

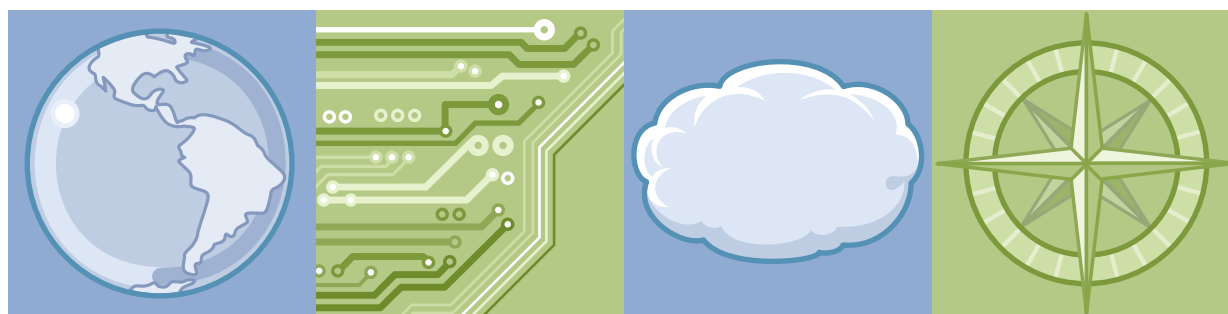


IBM Training

Student Exercises

IBM Integration Bus V9 Application Development II

Course code WM675 ERC 1.0



WebSphere Education

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:

DB™	DB2®	developerWorks®
Express®	Q®	Rational®
SurePOS™	Tivoli®	WebSphere®
z/OS®		

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

Lenovo and ThinkPad are trademarks or registered trademarks of Lenovo in the United States, other countries, or both.

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.

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

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

November 2013 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 2013.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Trademarks	v
Exercises description	vii
Exercise 1. Implementing message aggregation	1-1
Part 1: Start components and prepare the workspace	1-5
Part 2: Complete the fan-out message flow	1-5
Part 3: Complete the fan-in message flow	1-7
Part 4: Deploy and test the aggregation request/reply	1-8
Part 5: Clean up	1-11
Exercise 2. Extending a DFDL model	2-1
Exercise 3. Implementing an MRM message set	3-1
Part 1: Start components and import resources	3-4
Part 2: Model message and set properties for physical formats	3-4
Part 3: Extend the message flow	3-17
Part 4: Test with the Test Client	3-25
Part 5: Optional: Nonexistent reply-to queue	3-27
Part 6: Clean up the environment	3-28
Exercise 4. Implementing web services with IBM Integration Bus	4-1
Part 1: Start components and prepare the workspace	4-3
Part 2: Import and review the SOAP nodes sample	4-4
Part 3: Examine the SOAP Nodes Sample application	4-9
Part 4: Run the SOAP nodes sample	4-15
Part 5: Using the TCP/IP Monitor	4-20
Part 6: Set the reply message path with WS-Addressing	4-27
Part 7: Clean up the workspace	4-31
Exercise 5. Creating and testing an integration service	5-1
Part 1: Open a new workspace	5-3
Part 2: Create an integration service	5-3
Part 3: Define the service interface	5-6
Part 4: Implement the EmployeeService message flow	5-13
Part 5: Deploy the web service	5-23
Part 6: Test the web service with the Test Client	5-27
Part 7: Optional: Test the web service with soapUI test client	5-30
Part 8: Clean up	5-32
Exercise 6. Discovering WebSphere MQ and database services	6-1
Part 1: Preparing the environment	6-3
Part 2: Create a WebSphere MQ request/response service definition	6-5
Part 3: Publish the WebSphere MQ service to an Integration Registry	6-9

Part 4: Discover the database service	6-10
Part 5: Add an operation to the database service	6-16
Part 6: Implementing a WebSphere MQ service	6-19
Part 7: Calling a WebSphere MQ service	6-22
Part 8: Deploy and test the application	6-25
Exercise 7. Implementing a message flow that uses JMS	7-1
Part 1: Start components and prepare the workspace	7-3
Part 2: Import the JMSNodes sample	7-4
Part 3: Review the imported components	7-9
Part 4: Testing the sample	7-15
Part 5: Clean up	7-23
Exercise 8. Implementing IBM Integration Bus runtime security	8-1
Part 1: Start components and prepare the workspace	8-3
Part 2: Reviewing the message flows	8-3
Part 3: Testing the message flows	8-6
Part 4: Clean up	8-16
Exercise 9. Recording and replaying message flow data	9-1
Part 1: Set up the application for record and replay	9-3
Part 2: Set up the environment for Record and Replay	9-10
Part 3: View messages in the IBM Integration web console	9-13
Part 4: Replay messages	9-16
Part 5: Handle failed messages	9-19
Part 6: Clean up	9-20
Exercise 10. Creating and implementing a user-defined pattern	10-1
Part 1: Start components and prepare the workspace	10-3
Part 2: Create an exemplar message flow	10-3
Part 3: Create a pattern authoring project	10-7
Part 4: Configure the pattern authoring project	10-9
Part 5: Package the pattern authoring project for distribution	10-24
Part 6: (Optional) Advanced pattern authoring	10-25
Part 7: Clean up	10-27

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:

DB™	DB2®	developerWorks®
Express®	Q®	Rational®
SurePOS™	Tivoli®	WebSphere®
z/OS®		

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

Lenovo and ThinkPad are trademarks or registered trademarks of Lenovo in the United States, other countries, or both.

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.

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

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

Exercises description

This course includes the following exercises:

- Exercise 1: Implementing message aggregation
- Exercise 2: Extending a DFDL model
- Exercise 3: Implementing a message model
- Exercise 4: Implementing web services
- Exercise 5: Creating and testing an integration service
- Exercise 6: Discovering WebSphere MQ and database services
- Exercise 7: Implementing a message flow that uses JMS
- Exercise 8: Implementing IBM Integration Bus runtime security
- Exercise 9: Recording and replaying message flow data
- Exercise 10: Creating and implementing a user-defined pattern

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

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

Exercise 1. Implementing message aggregation

What this exercise is about

In this exercise, you send four requests to a back-end system and aggregate the replies into a single output message.

What you should be able to do

At the end of the exercise, you should be able to:

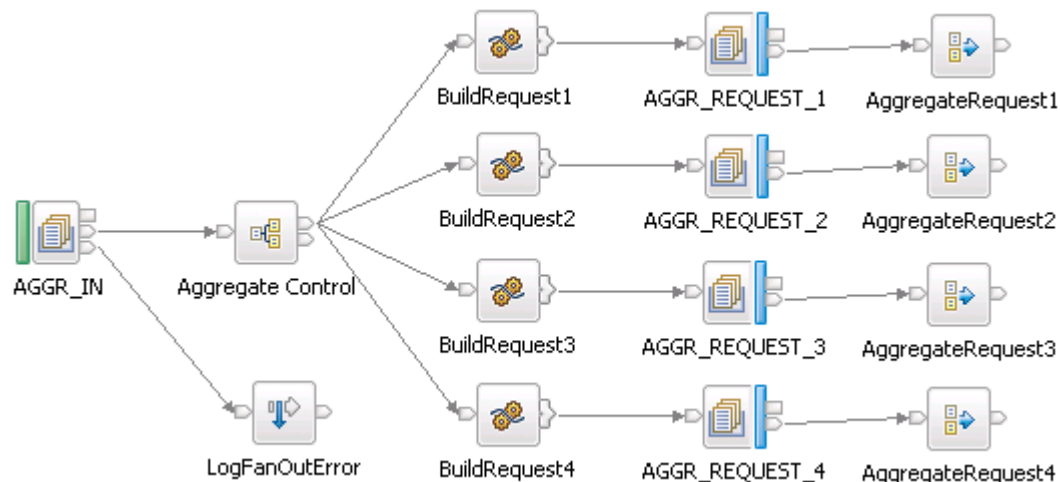
- Use the AggregateControl node and the AggregateRequest node to generate and concurrently fan-out related requests
- Use the AggregateReply node to aggregate messages into a single output message

Introduction

This exercise combines the generation and fan-out of four related requests with the fan-in of the corresponding replies. It also compiles those replies into a single aggregated reply message. The exercise uses the AggregateControl, AggregateRequest, and AggregateReply nodes to perform a basic four-way aggregation operation with simple fan-out and fan-in flows.

In the first part of this exercise, you start the IBM Integration Toolkit and prepare the workspace.

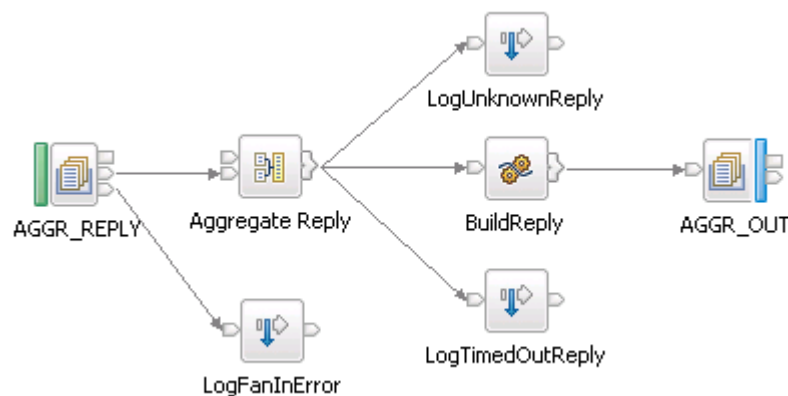
In the second part of the exercise, you configure the fan-out message flow to generate four different request messages and start the tracking of the aggregation operation.



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.



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 output message from the **AggregateReply** node does not contain an WebSphere MQ message descriptor (MQMD), so you use a **Compute** node to add an MQMD before sending the message to an **MQOutput** node. 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 fan-in flow does not aggregate the messages in any specific order.



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

Requirements

- A workstation with WebSphere MQ, IBM Integration Bus
- The IBM Integration Bus default configuration
- WebSphere MQ queues AGGR_REPLY, AGGR_OUT, AGGR_IN, and AGGR_REQUEST
- Lab exercise files in the C:\Labs\Lab01-Aggregate directory

Output message example

The example shows excerpts from the output message that is generated by the AggregateReply node.

```
<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>
        </Invoice>
      </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>
          </Item>
        </Invoice>
      </SaleList>
    </SaleEnvelope>
  </Request4>
</ComIbmAggregateReplyBody>
```

Exercise instructions

Part 1: Start components and prepare the workspace

- ___ 1. Make sure that all local IBM Integration Bus components are running.
 - ___ a. Start the IBM Integration Toolkit. If the IBM Integration Toolkit is already running, click **File > Switch Workspace > Other** from the toolbar.
 - ___ b. Wait for the toolkit to start. The **Workspace Launcher** dialog box is displayed after a few moments.
 - ___ c. For **Workspace**, enter an appropriate value, such as:
C:\workspace\aggregation, and then click **OK**.
 - ___ d. Wait for the workspace to open. This operation can take several minutes.
- ___ 2. When the **Welcome** page is displayed, close it. The **Integration Development** perspective is displayed.
- ___ 3. Import all the starting project from the project interchange file.
 - ___ a. From the menu bar, select **File > Import**.
 - ___ b. Expand **Other**, select **Project Interchange**, and then click **Next**.
 - ___ c. To the right of **From zip file**, click **Browse**.
 - ___ d. Go to C:\Labs\Lab01-Aggregate.
 - ___ e. Select AggregationLabStartingPoint_PI.zip and then click **Open**. The **Import Projects** dialog box is displayed.
 - ___ f. Ensure that **Aggregation Message Flows** is selected, and then click **Finish**.

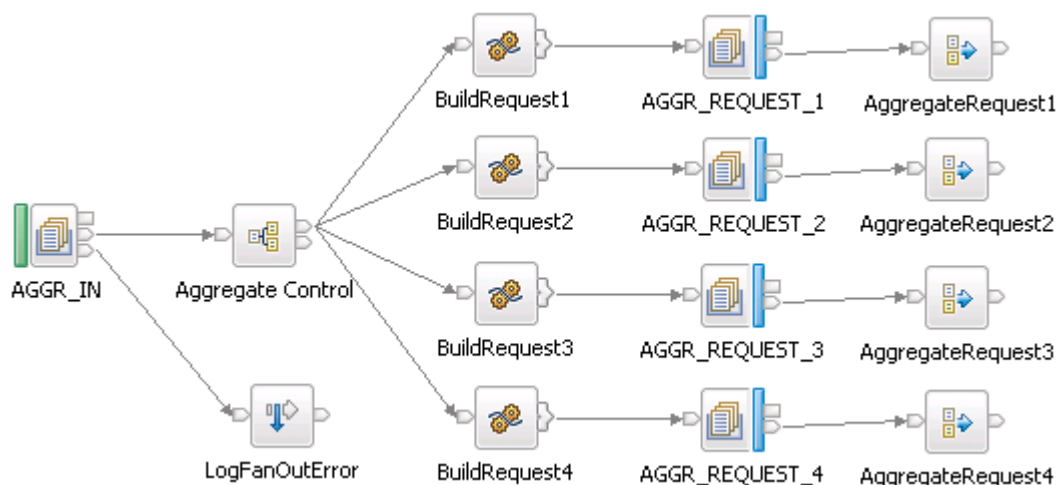
The projects are imported under the `Independent Resources` folder. Wait for the workspace build to complete. You resolve any errors that are reported in subsequent exercise steps.

Part 2: Complete the fan-out message flow

In this part of the exercise, you add and configure the aggregation nodes for the fan-out message flow.

- ___ 1. Open **FanOut.msgflow**.
 - ___ a. In the Application Development view, expand **Aggregation Message Flows > Flows**.
 - ___ b. Double-click **FanOut.msgflow** to open it in the Message Flow editor.
- ___ 2. Add the following nodes to the message flow from the **Routing** drawer:
 - One **AggregateControl** node
 - Four **AggregateRequest** nodes

- ___ 3. Wire the four new nodes as follows:
- ___ a. Wire the **Out** terminal of the **AGGR_IN** MQInput node to the **In** terminal of the **AggregateControl** node.
 - ___ b. Wire the **Out** terminal of the **AggregateControl** node to the **In** terminal of **BuildRequest1**, **BuildRequest2**, **BuildRequest3**, and **BuildRequest4** Compute nodes.
 - ___ c. Wire the **Out** terminal of the **AGGR_REQUEST_1** MQOutput node to the **In** terminal of the **Aggregate Request** node.
 - ___ d. Wire the **Out** terminal of the **AGGR_REQUEST_2** MQOutput node to the **In** terminal of the **Aggregate Request1** node.
 - ___ e. Wire the **Out** terminal of the **AGGR_REQUEST_3** MQOutput node to the **In** terminal of the **Aggregate Request2** node.
 - ___ f. Wire the **Out** terminal of the **AGGR_REQUEST_4** MQOutput node to the **In** terminal of the **Aggregate Request3** node.
- ___ 4. To prevent confusion, rename the Aggregate Request nodes so that the numbering matches the number of the node to which they are attached. Change:
- **Aggregate Request3** to `AggregateRequest4`
 - **Aggregate Request2** to `AggregateRequest3`
 - **Aggregate Request1** to `AggregateRequest2`
 - **Aggregate Request** to `AggregateRequest1`



- ___ 5. Configure the node properties for **Aggregate Control** node.
- For the **Aggregate name**, enter: `SalesAggr`
- ___ 6. Configure the **Folder name** property for each of the `AggregateRequest` nodes.

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

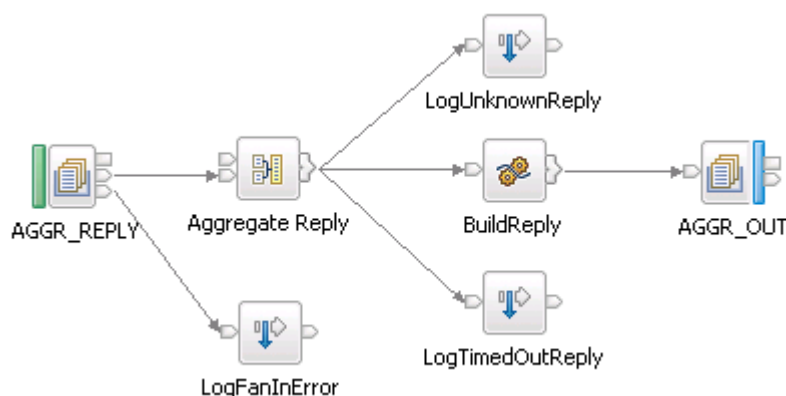
- ___ a. For **AggregateRequest1**, set **Folder name** to: Request1
- ___ b. For **AggregateRequest2**, set **Folder name** to: Request2
- ___ c. For **AggregateRequest3**, set **Folder name** to: Request3
- ___ d. For **AggregateRequest4**, set **Folder name** to: Request4
- ___ 7. Verify that the messages that are generated by the **FanOut.msgflow** are request messages.
 - ___ a. Select the **AGGR_REQUEST_1** MQOutput node.
 - ___ 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**.
- ___ 8. Save the message flow.
- ___ 9. 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 add and configure the aggregation nodes for the fan-in message flow.

- ___ 1. Open **FanIn.msgflow** in the Message Flow editor.
- ___ 2. Add an **Aggregate Reply** node to the message flow from the **Routing** drawer.
- ___ 3. Wire the **Aggregate Reply** 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: `SalesAggr`
 - ___ b. For the **Unknown message timeout**, enter: 10
 - ___ c. Clear the **Transaction mode** checkbox.
- ___ 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

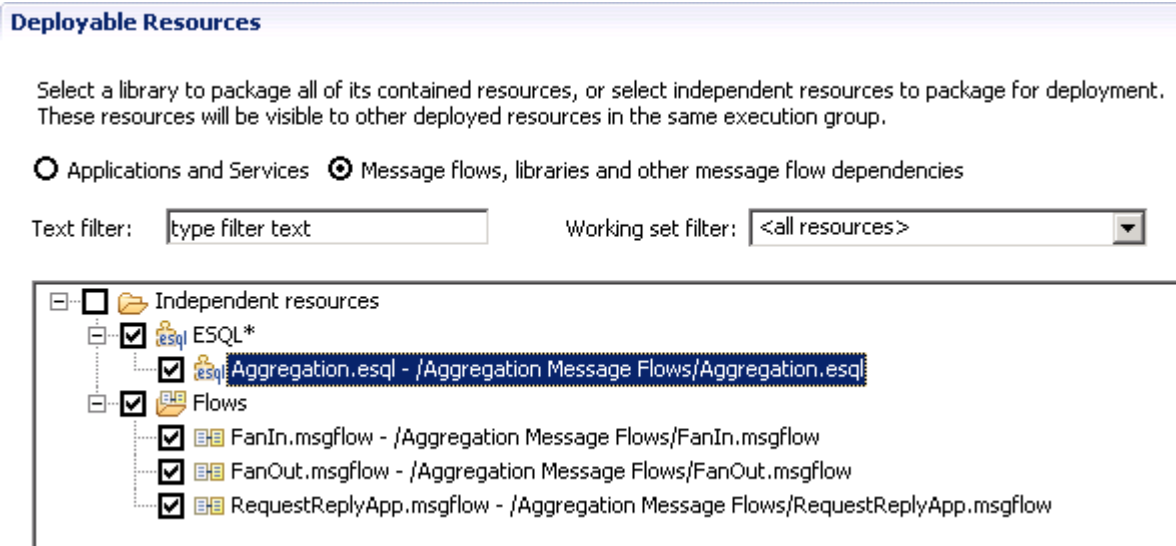
Now that the message flow is complete, you are ready to deploy and test.

You simulate the back-end application with a simple message flow that is named **RequestReplyApp.msgflow** that is supplied in **Aggregation Message Flow** project.

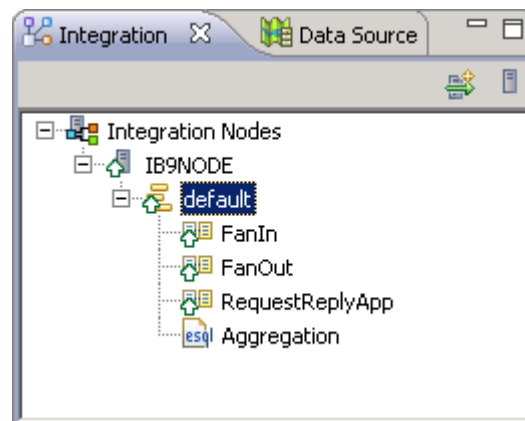


- ___ 1. Create a BAR file to contain the message flow artifacts.
- ___ a. Select **File > New > BAR File**. The **New BAR file** dialog box is displayed.
 - ___ b. For **Name**, enter `Aggregation`, and then click **Finish**. The BAR file editor is displayed.

- ___ c. In the BAR file editor, under **Independent Resources**, select **ESQL** and **Flows**. This action selects all available resources for deployment.



- ___ d. Click **Build and Save**. The BAR file is created.
- ___ 2. Deploy the **Aggregation.bar** to the integration server that is named **default** in the integration node that is named **IB9NODE**.



- ___ 3. Check the **Deployment Log** to ensure that the deployment was successful.
- ___ 4. Open the Test Client with the preconfigured test file.
- ___ a. Expand **Independent Resources > Aggregation Message Flows > Flow Tests** in the Application Development view.
- ___ b. Double-click **Aggregation8k.mbtest**. The Test Client view opens.
- ___ 5. Two message flow events are already configured:
- **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 and that the Sales message is preloaded in the Message section.

▼ **Detailed Properties**

Host:	localhost
Port:	0
Server channel:	SYSTEM.BKR.CONFIG
Queue manager:	IB9QMGR
Queue:	AGGR_IN

Message

► Header

Body: Edit as text

☐ Show in hexadecimal viewer (Read Only)

```
<SaleEnvelope><Header><SaleListCount>8</SaleListCount></Header><SaleList><Invoice><Initial>K<,
>Euros</Currency></Invoice></SaleList><SaleList><Invoice><Initial>K</Initial><Initial>h</Initial><Su
e></SaleList><SaleList><Invoice><Initial>q</Initial><Initial>F</Initial><Surname>TJnBitwOBVU</Surn
nvoice><Initial>b</Initial><Initial>P</Initial><Surname>yPXtKEWufTH</Surname><Item><Code>01</
l><Initial>o</Initial><Surname>AwqJMgJZpMj</Surname><Item><Code>01</Code><Code>05</Code
rname>tStYhUgRMOF</Surname><Item><Code>04</Code><Code>01</Code><Code>00</Code><De
e><Item><Code>03</Code><Code>02</Code><Code>04</Code><Description>IPmXBEv</Description
ode>01</Code><Code>00</Code><Description>mhPmVUH</Description><Category>Ixytw</Category>
```

- ___ 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 that are identified by their folder named (Request1, Request2, Request3, and Request4).

Message

► Header

Body: View as XML structure

Name	Value
[-] ComIbmAggregateReplyBody	
+ Request1	
+ Request2	
+ Request3	
[-] Request4	
ReplyIdentifier	X'414d5120494239514d47522020202008e2275220...
[-] SaleEnvelope	
[-] SaleList	
[-] Invoice	
Initial	S
Initial	e
Surname	rtrrMIfcYQt
[-] Item	
Code	03
Code	02
Code	04



Note

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

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

Part 5: Clean up

- ___ 1. Close all open editor windows by typing **Ctrl + Shift + W**. You do not need to save any changes, if you are prompted.
- ___ 2. In the **Integration nodes** view, right-click the **default** integration server, and then select **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.

End of exercise

Exercise 2. Extending a DFDL model

What this exercise is about

In this exercise, you learn how to model complex data in Data Format Description Language (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 message parsing.

What you should be able to do

At the end of this exercise, you should be able to:

- Extend a DFDL message model to add header and trailer records
- Reference a DFDL message model in a new DFDL message model
- Use discriminators in a DFDL model so that the parser can conditionally process elements and optimize message parsing

Introduction

The DFDL standard allows you to reference an existing message model in a new message model. In the first part of this lab, you extend an existing 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 used to 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, back track and make another attempt.

The use of discriminators can be used to assert that an object is 'known to exist', which prevents incorrect back-tracking. In the second part of this lab, you add discriminators to an existing DFDL model.

Requirements

- IBM Integration Toolkit
- Lab exercise files in C:\Labs\Lab02-DFDL

Exercise instructions

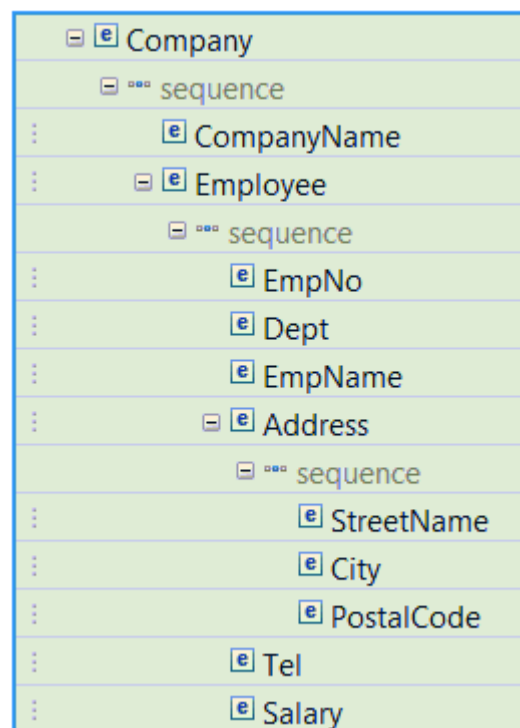
Part 1: Extending the message model

In this part of the lab, you are going to 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:\Labs\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)
]
```

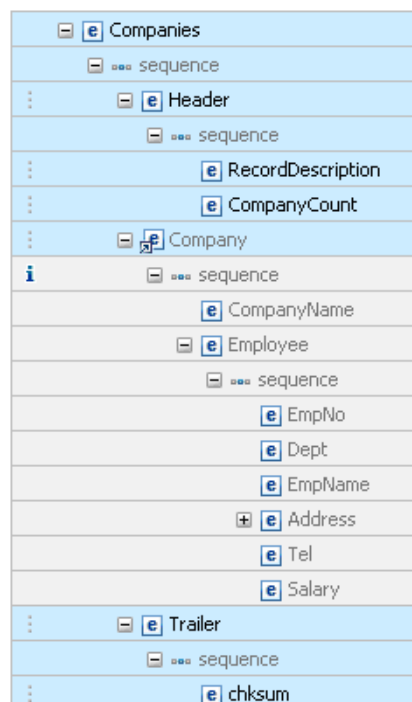
The DFDL model for the `Company.txt` file is defined in the `Company.xsd` schema.



In this exercise, you extend this DFDL structure so that it can parse the `Companies.txt` data file (in the `C:\Labs\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}
```

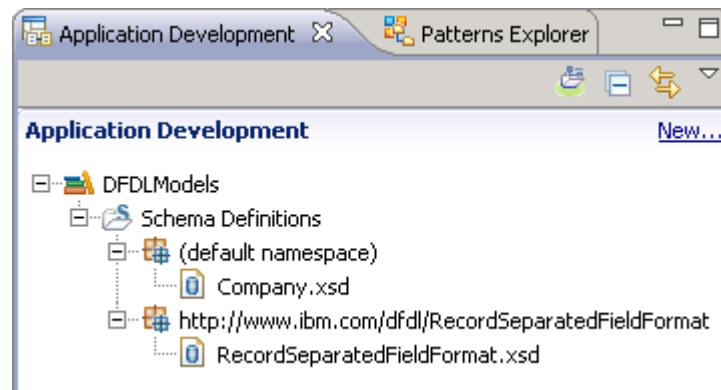
The header record contains a record description and a company count. The trailer record contains a checksum. You must modify the existing DFDL message structure to add the header and trailer records.



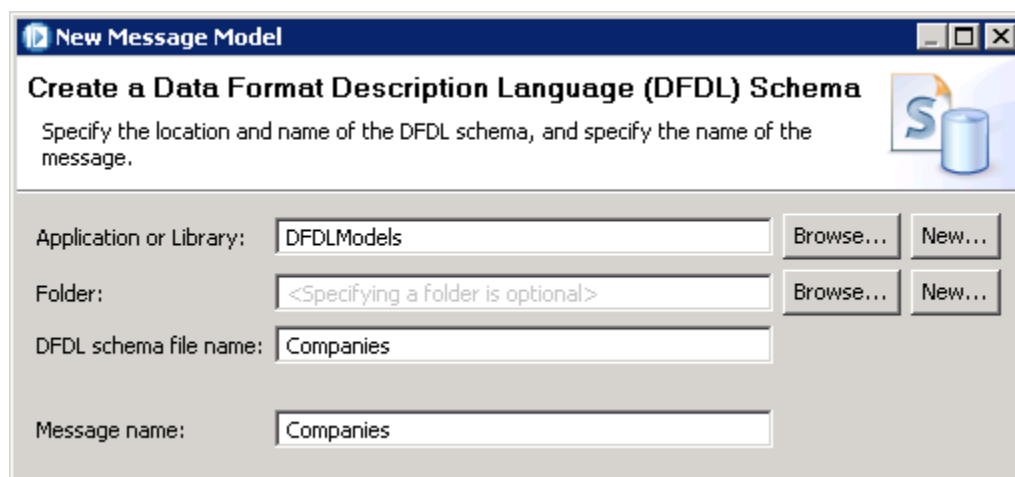
- ___ 1. Start the IBM Integration Toolkit and create a workspace.
- ___ 2. Import the Project Interchange file that contains the **DFDLModels** library.

The Project Interchange file is `DFDLLab_Startingpoint.zip` and is in the `C:\Labs\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`.

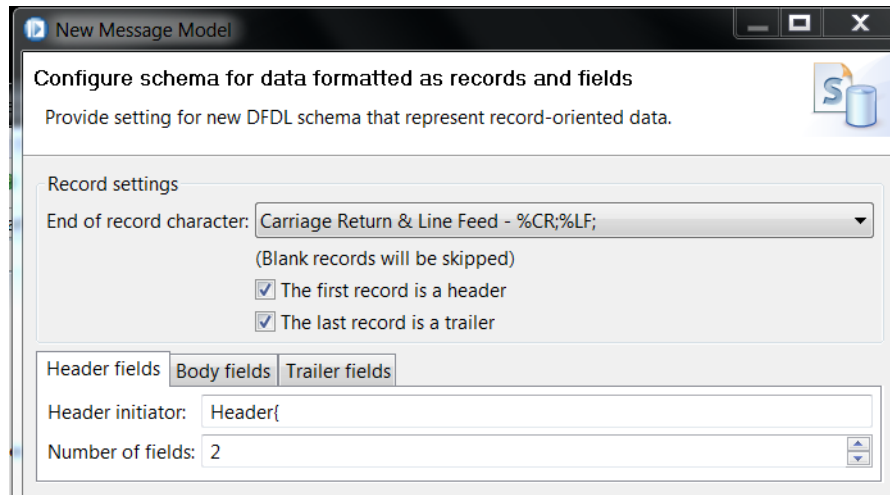


- ___ 3. Use the New Message Model wizard to create a record-oriented message model schema that is named **Companies** in the **DFDLModels** library that describes the header and trailer records.
 - ___ a. In the Application Development view, select **New > Message Model**.
 - ___ b. In the New Message Model wizard, select **Record-oriented text** and then click **Next**.
 - ___ c. From the wizard, select **Create a DFDL schema file using the wizard to guide you** and then click **Next**.
 - ___ d. Select **DFDLModels** for the **Application or Library** and then enter `Companies` for the DFDL schema file name. Click **Next**.

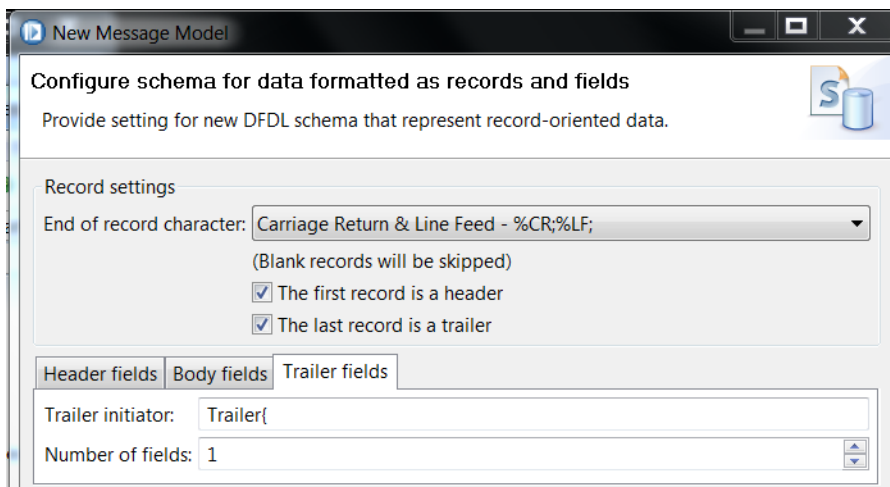


- ___ e. Do not change the **Record settings**.

- ___ f. In the **Header fields** tab, enter Header{ as the **Header initiator**, and 2 for the **Number of fields**.

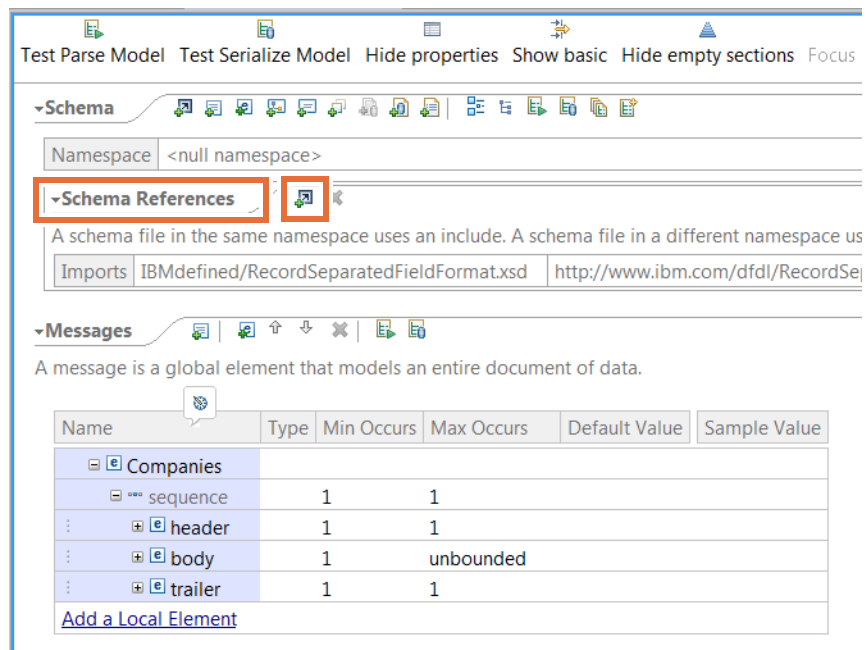


- ___ g. On the **Trailer fields** tab, enter Trailer{ as the **Trailer initiator** and 1 for the **Number of fields**.

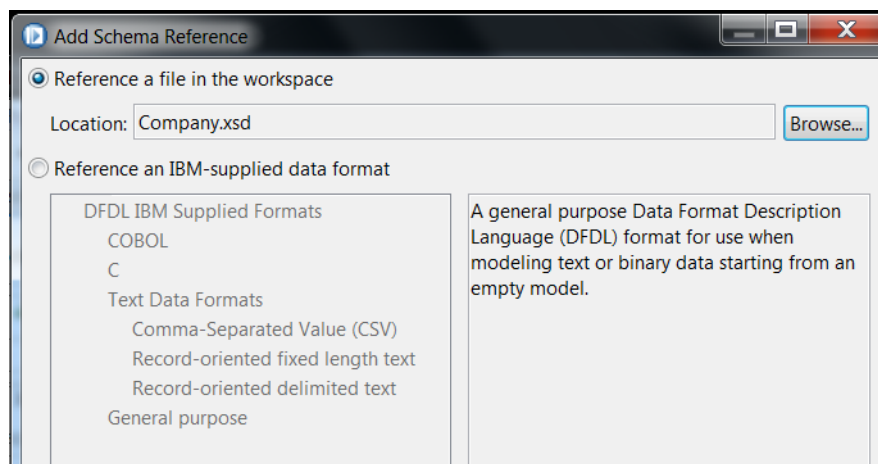


- ___ h. Click **Finish**. The DFDL Schema editor opens with the generated DFDL schema.
- ___ 4. Click the **Show all sections** icon in the DFDL Schema editor toolbar.
- ___ 5. Click the arrow next to the **Schema References** section to expand the references of the DFDL schema file.

- ___ 6. The `Companies.xsd` message model is going to build 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 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.

Schema

Namespace: <null namespace> [Change namespace](#)

Schema References

A schema file in the same namespace uses an include. A schema file in a different namespace uses an import.

Includes: Company.xsd

Imports: IBMdefined/RecordSeparatedFieldFormat.xsd <http://www.ibm.com/dfdl/RecordSeparatedFieldFormat>



Note

You now have two **Schema References**:

- Company.xsd, which you just added. You are going to reuse this message model to create a more complex one.
- RecordSeparatedFieldFormat.xsd, which the wizard automatically adds. It contains Record Separated specific defaults for DFDL properties.

___ 7. 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.

Test Parse Model Test Serialize Model Hide properties Show advanced Hide empty sections **Focus on selected**

Messages

A message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
[-] Companies					
[-] sequence		1	1		
[+] header		1	1		
[+] body		1	unbounded		
[+] trailer		1	1		

[Add a Local Element](#)

___ 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. Change the **CompanyCount Initiator** value from **iHead2** to **compCount:** (be sure to enter the colon after **compCount**).

Representation Properties (x)= Variables Asserts and Discriminators

CompanyCount (Element)

<type filter text>

Property	Value	
Comment		...
General		
Encoding (code page)	<dynamically set>	...
Byte Order	<dynamically set>	...
Content	integer	
Representation	text	
Length Kind	delimited	...
Default Value	<unset>	
Text Content		
Text Number Representation	standard	
Escape Scheme Reference	recSepFieldsFmt:RecordEscapeScheme	...
Occurrences		
Min Occurs	1	
Max Occurs	1	
Delimiters		
Initiator	compCount:	...
Terminator	<no terminator>	...
Validation	integer	

- ___ g. Change the **RecordDescription Initiator** value to **recDesc:**
- ___ 8. Select the **Header** element. In the **Delimiters** section of the Representation Properties, set the **Terminator** property to **}%CR;%LF;**
- ___ 9. Select **sequence** under the **Header** element.
- In the **Delimiters** section of the Representation Properties, set the **Separator** property to: **,** (comma).

- ___ 10. Modify the **trailer** record to define the checksum field.
- ___ a. Change the **trailer** element name to: Trailer
- ___ b. Change the element under **Trailer** to: chksum
- ___ c. Change the **chksum** element **Initiator** value to: chksum:

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
sequence		1	1
chksum	string	1	1

- ___ d. In the **Delimiters** section of the Representation Properties, set the **Terminator** property to: }

- ___ 11. Select the **sequence** element under the **Trailer** element.

In the **Delimiters** section of the Representation Properties, delete the **Separator** property value. Press Enter to make sure that the value is updated. The wizard automatically added the separator, but it is not required for this model.

- ___ 12. 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. The wizard automatically added the separator, but it is not required for this model.

- ___ 13. Replace the **body** portion of 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).
- ___ a. Right-click the **Companies** element and then select **Add Sequence**. A new sequence element is added to the bottom of the schema.

Name	Type	Min Occurs	Max Occurs	Default Value
Companies				
sequence		1	1	
Header		1	1	
sequence		1	1	
RecordDescription	string	1	1	
CompanyCount	string	1	1	
body		1	unbounded	
Trailer		1	1	
sequence		1	1	
chksum	string	1	1	

- ___ b. Right-click the new **sequence** element and then select **Add Element Reference**.
- ___ c. Select **Company** : from the list and then click **OK**.

▼Messages

A message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
[-] Companies					
[-] sequence		1	1		
[-] Header		1	1		
[-] sequence		1	1		
[-] RecordDescription	string	1	1		
[-] CompanyCount	string	1	1		
[-] Trailer		1	1		
[-] sequence		1	1		
[-] checksum	string	1	1		
[-] sequence		1	1		

[Add a Local Element](#)

Add Element Reference

Select an element: Company :

OK Cancel



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 DFDL schema where it was defined by clicking the yellow arrow that is displayed when you hover over the element.

Name	Type	Min Occurs	Max Occurs	Default Value
[-] Companies				
[-] sequence		1	1	
[-] Header		1	1	
[-] sequence		1	1	
[-] RecordDescription	string	1	1	
[-] CompanyCount	string	1	1	
[-] sequence		1	1	
[-] Company		1	1	
[-] sequence		1	1	
[-] CompanyName	string	1	1	
[-] Employee		1	unbounded	

- ___ d. Right-click the newly created **sequence** element and then select **Move Up** so that it is moved above the **Trailer** element.
- ___ 14. 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>.

Schema References (1 include, 1 import)
A schema file in the same namespace uses an include. A schema file in a different namespace uses an import.

Messages
A message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default
Companies				
sequence		1	1	
Header		1	1	
sequence		1	1	
Company		1	unbounded	
sequence		1	1	
CompanyName	string	1	1	
Employee		1	unbounded	
sequence		1	1	
EmpNo	integer	1	1	
Dept	integer	1	1	
EmpName	string	1	1	

Property	Value
General	
Data Format Reference	<default format>
Encoding (code page)	<dynamically set>
Byte Order	<dynamically set>
Ignore Case	no
Fill Byte	0
Content	
Alignment	
Delimiters	
Separator	<no separator>
Initiator	<no initiator>
Terminator	<no terminator>
Output New Line	%CR;%LF;

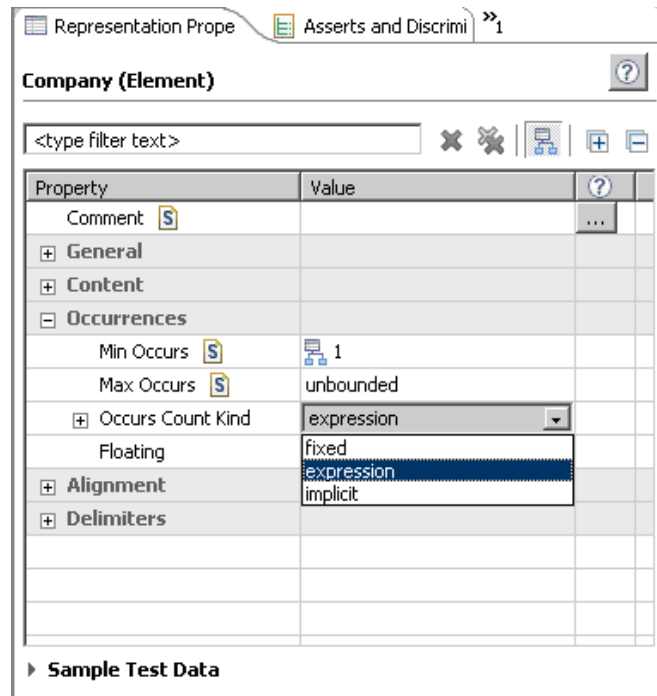
- ___ 15. Select **Max Occurs** column of the **Company** element reference, and change it to unbounded.

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

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Companies					
sequence		1	1		
Header		1	1		
sequence		1	1		
RecordDescription	string	1	1		head_value1
CompanyCount	integer	1	1		1
sequence		1	1		
Company		1	unbounded		
sequence		1	unbounded		
CompanyName	string	1	1		
Employee		1	unbounded		
Trailer		1	1		
sequence		1	1		
chksum	string	1	1		trailer_value1

- ___ 16. Modify the schema so that the **CompanyCount** element in the **Header** dictates the number of occurrences of the **Company** element.
- ___ a. Click the **Show Advanced** icon to show the advanced Representation Properties.

- ___ b. Select the **Company** element reference. In the **Occurrences** section of the Representation Properties, change the **Occurs Count Kind** from *fixed* to *expression*.



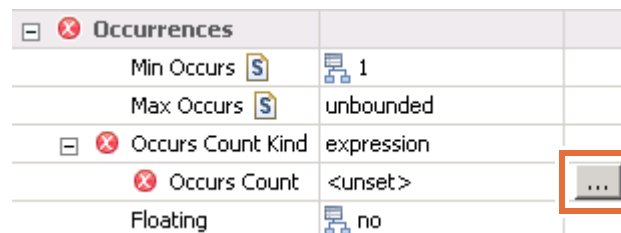
- ___ c. Expand the **Occurs Count Kind** property to show the **Occurs Count** property.



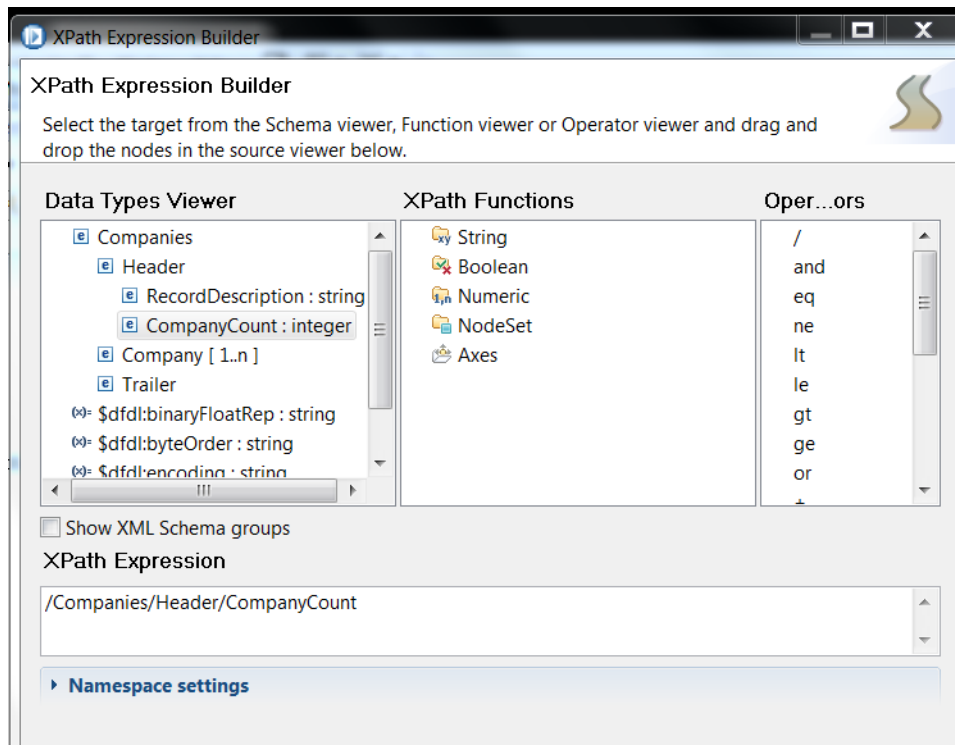
Note

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 ellipses next to the **Occurs Count** property to open the XPath Expression Builder.



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



- ___ f. Click **Finish**.
- ___ g. **Save** the changes in the editor.
- ___ h. Confirm that there are no errors in the Companies element.
- ___ 17. 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.

The screenshot shows the IBM DFDL Model Editor interface. The 'Data Formats' section is highlighted with a red box. The interface includes a top toolbar with icons for Test, Parse, Model, Test, Serialize, Model, Hide properties, Show basic, Show all sections, and Focus. Below the toolbar, the 'Schema' section is expanded, showing a 'Namespace' field with the value '<null namespace>'. The 'Schema References' section shows '(1 include, 1 import)' and a description: 'A schema file in the same namespace uses an include. A schema file in a different namespace...'. The 'Messages' section is expanded, showing a table with columns: Name, Type, Min Occurs, Max Occurs, Default Value, and Sample Value. The table contains one entry: 'Companies'. Below the table is a link 'Add a Local Element'. The 'Data Formats' section is highlighted with a red box and contains a table with columns: Name and Type. The table contains one entry: '<default format>' with the type 'Definition Format'. The 'Variables' section is expanded, showing '(4 variables)' and a description: 'A variable holds a value that can be used in DFDL expressions.'

- ___ 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**.

The screenshot shows the IBM Data Format Designer interface. On the left, the 'Schema' tab is active, showing a namespace of '<null namespace>' and a schema reference for 'Companies'. Below this, the 'Messages' tab shows a message named 'Companies' with a type of 'Definition Format'. The 'Data Formats' tab is also visible, showing a data format named '<default format>' with a type of 'Definition Format'. On the right, the 'Representation Properties' panel is open, showing a table of properties. The 'Delimiters' section is expanded, and the 'Document Final Terminator Can Be Missing' property is set to 'yes'.

Property	Value
General	
Content	
Text Content	
Binary Content	
Number	
Calendar	
Boolean	
Structural Content	
Occurrences	
Alignment	
Delimiters	
Separator	,
SeparatorSuppressionPolicy	anyEm
Separator Position	infix
Initiator	<no ini
Terminator	<no ter
Document Final Terminator Can Be Missing	yes
Nil Value Delimiter Policy	Initiato

- ___ 18. Save the schema.

Check the **Problems** view and verify that there are no errors.

- ___ 19. Test the model by parsing the `Companies.txt` file.

- Click the **Test Parse Model** icon.
- In the Parser Input section, select **Content from a data file** and click **Browse**.
- Check the **Select an input file from the file system**.
- Go to `C:\Labs\Lab02-DFDL\data`, select the `Companies.txt` file, and then click **Open**.
- Click **OK** on the File Selection window.
- Click **Yes** to switch to the DFDL Test perspective.
- Click **OK** on the Test Parse Model window.

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

- ___ h. Review the Parsed input and the Logical Instance view to verify that the parsing is correct.

DFDL Test - Logical Instance

Data source: <From 'DFDL Test - Parse' view>

Message: Companies (/Workspace/WM675_DFDL/DFDLModels/Companies.xsd)

Tree View XML View

Name	Type	Value	
Companies			
Header			
RecordDescription	xs:string	My Compan...	
CompanyCount	xs:integer	5	
Company			
CompanyName	xs:string	BBC	
Employee			
EmpNo	xs:integer	111111	
Dept	xs:integer	500	
EmpName	xs:string	Alice Wong	
Address			
Tel	xs:string	905-347-5649	
Salary	xs:decimal	135599.95	
Employee			
Employee			
Employee			
Employee			
Company			
Company			
Company			
Company			
Trailer			
chksum	xs:string	1234567890	

If there are errors, review the parsing errors and the **Trace** tab to identify the problem.

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, back track, and make another attempt.

Discriminators can be used to assert that an object is 'known to exist', which prevents incorrect back-tracking. 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:\Labs\Lab02-DFDL\discriminators` 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 the `CustomerCountry = 'Canada'`.
- The **CustomerCounty** redefines clause is used when the `CustomerCountry = 'UK' or 'Ireland'`.
- The **CustomerRegion** redefines clause is used when the `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 the `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.

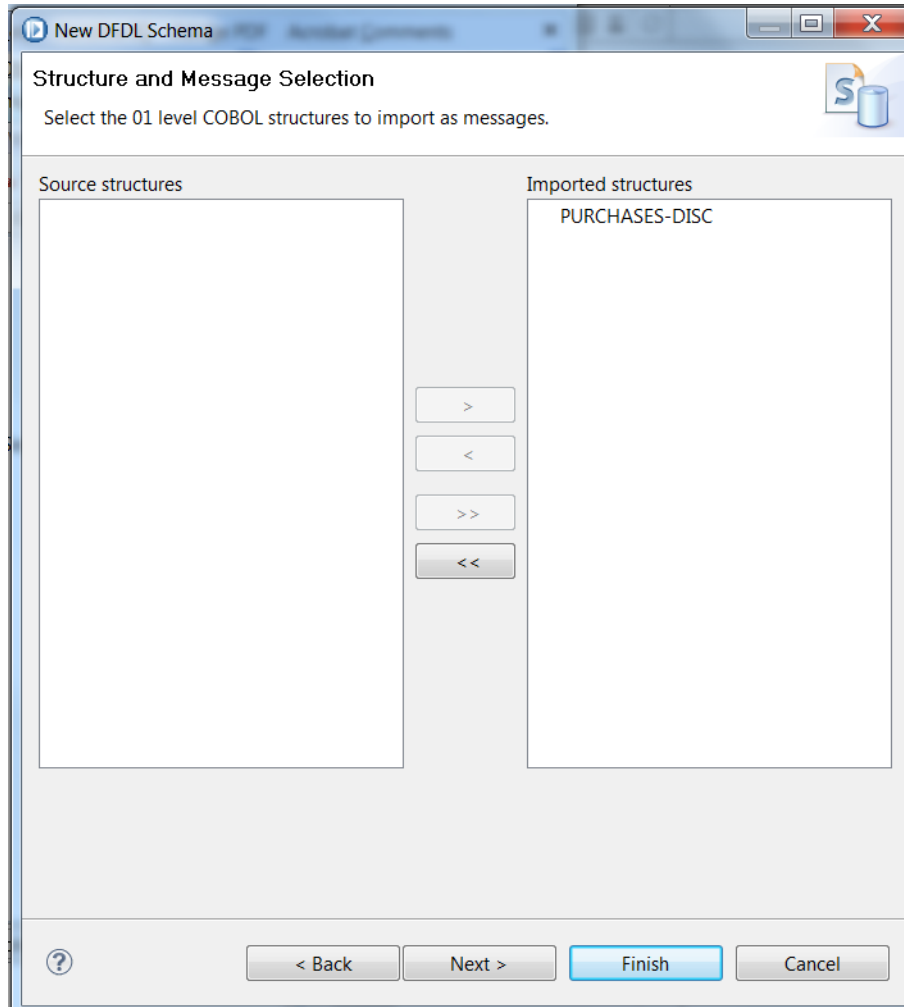
___ 1. Create a DFDL message model by importing the COBOL copybook that is named `PURCHASES-DISC.cpy` in the `C:\Labs\Lab02-DFDL\discriminators` directory.

- ___ a. In the **DFDLModels** library that you created in Part 1 of this lab, select **New > Message Model**.
- ___ b. In the New Message Model wizard, select **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**.

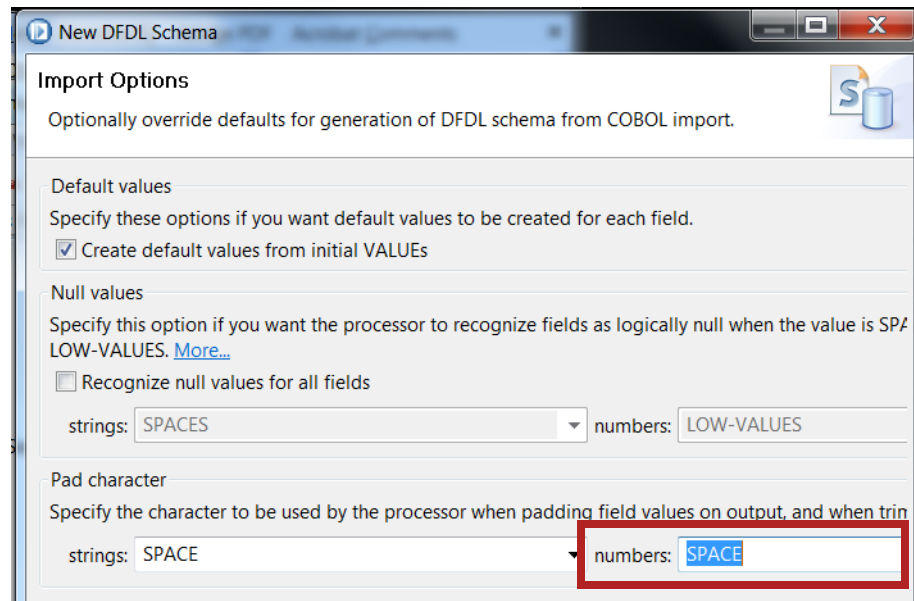
- ___ d. Select the option to **Select source from outside workspace**.
- ___ e. Click **Browse** and go to the `C:\Labs\Lab02-DFDL\discriminators` directory.
- ___ f. Select `PURCHASES-DISC.cpy` file and then click **OPEN**.
- ___ g. Click **Next**.

- ___ h. Select the **PURCHASES-DISC** source structure and 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, 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 by the COBOL REDEFINES clauses.

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**.

Name	Type
PURCHASESDISC	PURCHASESDISC
sequence	
REQUEST_TYPE	<PICX_string>
RET_CODE	<PICX_string>
CustomerId	<PICX_string>
CustomerLastName	<PICX_string>
CustomerFirstName	<PICX_string>
CustomerCompany	<PICX_string>
CustomerAddr1	<PICX_string>
CustomerAddr2	<PICX_string>
CustomerCity	<PICX_string>
CustomerCountry	<PICX_string>
choice	
CustomerArea	<PICX_string>
CustomerProvince	<PICX_string>
CustomerCounty	<PICX_string>
CustomerRegion	<PIC9-Display-Zoned_integer>
CustomerState	<PICX_string>
CustomerMailCode	<PICX_string>
CustomerPhone	<PICX_string>

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

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.

choice	
CustomerProvince	<PICX_string>
CustomerCounty	<PICX_string>
CustomerRegion	<PIC9-Display-Zoned_integer>
CustomerState	<PICX_string>
CustomerArea	<PICX_string>
CustomerMailCode	<PICX_string>

- ___ 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. Select **Content from a data file**,
- ___ c. Click **Select and input file from the file system**.

- ___ d. Browse to C:\Labs\Lab02-DFDL\discriminators, select 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 **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.

Tree View XML View		
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
CustomerProvince	xs:string	Texas
CustomerMailCode	xs:string	12312
CustomerPhone	xs:string	123-123-1234
CustomerLastUpdateDate	xs:string	04082008

CustomerProvince was selected because the branches of the choice are tried in the declared order until one parses successfully. Because **CustomerState** and **CustomerProvince** are both declared as PIC X(30), 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. Select the **CustomerProvince** element, and then click the **Asserts and Discriminators** tab.
- ___ d. Select **Discriminator**.

The screenshot shows the IBM DFDL editor interface. On the left, the 'Data Types Viewer' displays a tree structure of elements. The 'CustomerProvince' element is highlighted with a red box. On the right, the 'Asserts and Discriminators' tab is active. The 'Discriminator' radio button is selected. Below it, the 'Test Condition' field is empty, and the 'Add discriminator' link is visible. The 'Test Kind' and 'Test Condition' columns are also shown.

- ___ 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 PURCHASESDISC : PURCHASESDISC in the **Data Types Viewer**, and 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 **CustomerCountry**.

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

The XPath expression for the element **CustomerCountry** 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.

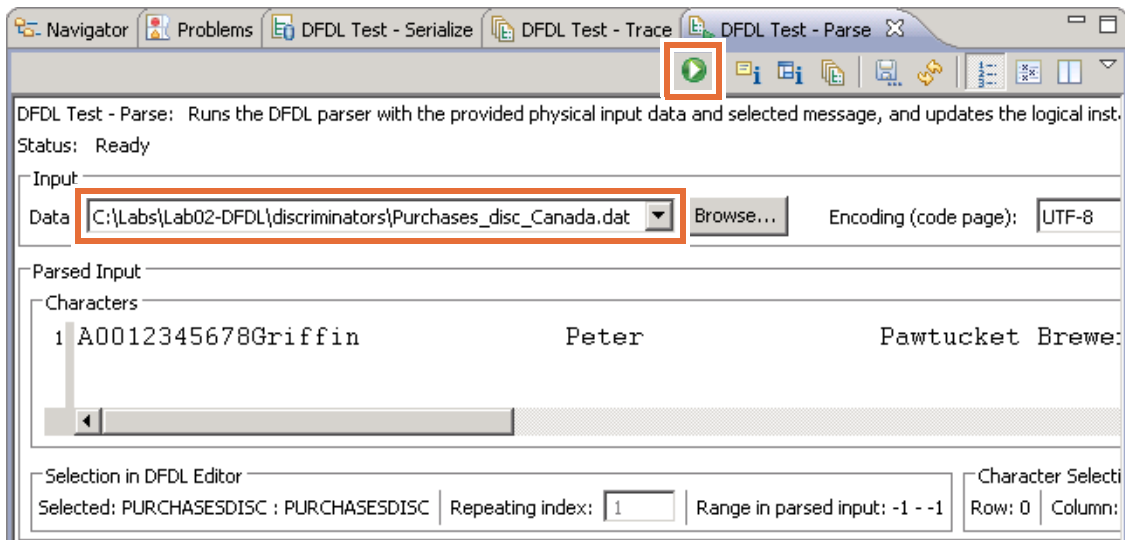
- ___ 8. Retest the model with discriminators.

- ___ 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. Now select the input file `Purchases_disc_Canada.dat`, and 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 sets the value to `Quebec`.

- ___ f. Now select the input file `Purchases_disc_UK.dat` and click the **Run Parser** icon.

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

In this case, the parser determines 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`.

- ___ g. Run the parser with input file `Purchases_disc_Russia.dat`. The file that contains data from Russia has the data in the field **Region** 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 uses 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. 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.

Exercise 3. Implementing an MRM message set

What this exercise is about

In this exercise, you create a message set to define a message. You also use a Compute node or JavaCompute node to propagate multiple physical messages by referencing a single message set.

What you should be able to do

At the end of the exercise, you should be able to:

- Create a message set
- Configure the logical and physical properties of the message
- Propagate multiple messages with different physical formats from a single compute node

Introduction

In this exercise, you create one message set that can be used to create three output files with a different physical format: XML, binary, and text.

Requirements

- A workstation with WebSphere MQ and IBM Integration Toolkit
- The IBM Integration Bus default configuration
- Lab files in the C:\Labs\Lab03-MRM directory
- The following WebSphere MQ queues: COMPLAINT_IN, COMPLAINT_OUT, COMPLAINT_REPLY, COMPLAINT_FALSE, COMPLAINT_FAILURE

Input message

The input message is the same as in the previous exercise:

```
<CUSTOMERCOMPLAINT>
  <VERSION>1</VERSION>
  <CUSTOMER_NAME>
    <N_FIRST>Ed</N_FIRST>
    <N_LAST>Fletcher</N_LAST>
  </CUSTOMER_NAME>
  <CUSTOMER_ADDRESS>
    <A_LINE>Mail Point 135</A_LINE>
    <A_LINE>Hursley Park</A_LINE>
    <TOWN>Winchester</TOWN>
    <ZIP>SO21 2JN</ZIP>
    <COUNTRY>UK</COUNTRY>
  </CUSTOMER_ADDRESS>
  <COMPLAINT>
    <C_TYPE>Delivery</C_TYPE>
    <C_REF>XYZ123ABC</C_REF>
    <C_TEXT>My order was delivered in time, but the package was torn and
      dirty.</C_TEXT>
  </COMPLAINT>
</CUSTOMERCOMPLAINT>
```

Output message (Reply): The same content in three different wire formats

1. XML:

```
<Complaint_Reply>
  <YourComplaint Type="Delivery">
    <YourReference>XYZ123ABC</YourReference>
    <Text>My order was delivered in time, but the package was torn and
dirty. </Text>
  </YourComplaint>
  <Reply>
    <OurReference>XYZ123ABC</OurReference>
    <Text>Your complaint has been received. </Text>
  </Reply>
</Complaint_Reply>
```

2. Binary:

Source	XML Source	Hexadecimal (Read Only)	
0000:	44 65 6C 69 76 65 72 79	20 20 58 59 5A 31 32 33	Delivery XYZ123
0010:	41 42 43 20 4D 79 20 6F	72 64 65 72 20 77 61 73	ABC My order was
0020:	20 64 65 6C 69 76 65 72	65 64 20 69 6E 20 74 69	delivered in ti
0030:	6D 65 2C 20 62 75 74 20	74 68 65 20 70 61 63 6B	me, but the pack
0040:	61 67 65 20 77 61 73 20	74 6F 72 6E 20 61 6E 64	age was torn and
0050:	20 64 69 72 74 79 2E 20	20 20 20 20 20 20 20 20	dirty.
0060:	20 20 20 20 20 20 20 20	20 20 20 20 20 20 20 20	

3. Text:

Source	XML Source	Hexadecimal (Read Only)	
0000:	44 65 6C 69 76 65 72 79	2B 2B 2B 58 59 5A 31 32	Delivery+++XYZ12
0010:	33 41 42 43 2B 2B 2B 4D	79 20 6F 72 64 65 72 20	3ABC+++My order
0020:	77 61 73 20 64 65 6C 69	76 65 72 65 64 20 69 6E	was delivered in
0030:	20 74 69 6D 65 2C 20 62	75 74 20 74 68 65 20 70	time, but the p
0040:	61 63 6B 61 67 65 20 77	61 73 20 74 6F 72 6E 20	ackage was torn
0050:	61 6E 64 20 64 69 72 74	79 2E 2B 2B 2B 43 30 31	and dirty.+++C01
0060:	2D 43 4F 4D 36 38 34 61	32 64 32 61 2D 33 38 34	-COM684a2d2a-384
0070:	38 2D 34 39 37 38 2D 61	35 64 33 2D 66 31 34 31	8-4978-a5d3-f141
0080:	32 61 30 38 33 38 31 61	2B 2B 2B 59 6F 75 72 20	2a08381a+++Your
0090:	63 6F 6D 70 6C 61 69 6E	74 20 68 61 73 20 62 65	complaint has be
00A0:	65 6E 20 72 65 63 65 69	76 65 64 2E	en received.

The structure of message Complaint_Reply:

<input type="checkbox"/>	<input checked="" type="checkbox"/>	Complaint_Reply
<input type="checkbox"/>	<input checked="" type="checkbox"/>	YourComplaint
	<input checked="" type="checkbox"/>	Type
	<input checked="" type="checkbox"/>	Reference
	<input checked="" type="checkbox"/>	Text
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Reply
	<input checked="" type="checkbox"/>	OurReference
	<input checked="" type="checkbox"/>	Text

Exercise instructions

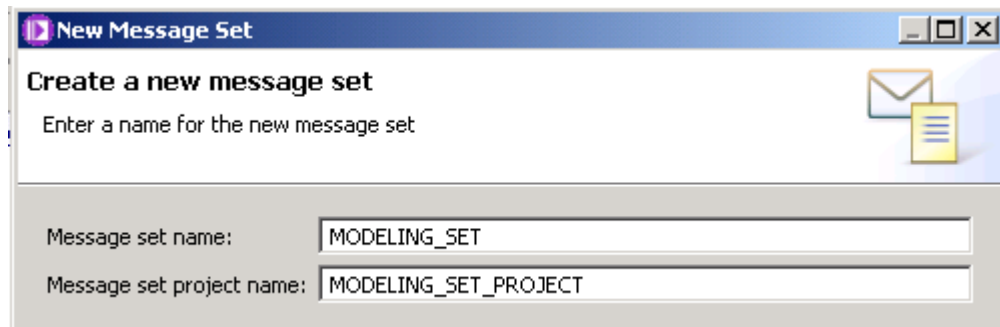
Part 1: Start components and import resources

- ___ 1. Make sure that all local IBM Integration Bus components are running:
Start the IBM Integration Toolkit, if it is not already open.
- ___ 2. Start a new workspace that is named C:\Workspace\MRM.
To start a new workspace:
 - ___ a. Click **File > Switch Workspace > Other** from the toolbar. The Workspace Launcher is displayed.
 - ___ b. For Workspace, enter C:\Workspace\MRM, and then click **OK**.
- ___ 3. When the new workspace opens, close the Welcome page.
- ___ 4. Import the MRMStartingPoint_PI.zip project interchange file that is the starting point for this lab.
 - ___ a. Click **File > Import > Other > Project Interchange**, and then click **Next**.
 - ___ b. To the right of **From zip file**, click **Browse**.
 - ___ c. Browse to C:\Labs\Lab03-MRM\MRMStartingPoint_PI.zip, and then click **Open**.
 - ___ d. Ensure that the RouteComplaint application is selected, and then click **Finish**. The project interchange file is opened in the Application Development view and the workspace is built.

Part 2: Model message and set properties for physical formats

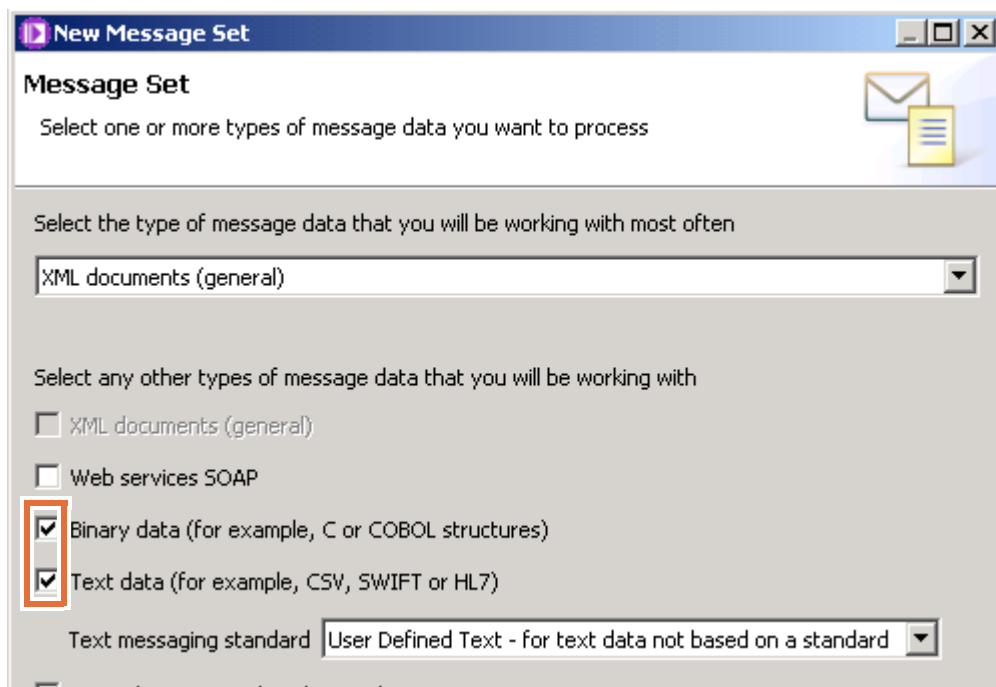
- ___ 1. Create a message set, MODELING_SET, in the message set project MODELING_SET_PROJECT with the physical formats XML1, Binary1, and Text1.
 - ___ a. Select **File > New > Other**. The New wizard is shown.
 - ___ b. Expand **Integration Bus - Message Set Development**, and then click **Message Set**.
 - ___ c. Click **Next**.
 - ___ d. For the **Message set name**, enter: MODELING_SET

- ___ e. For the **Project Name**, enter: MODELING_SET_PROJECT



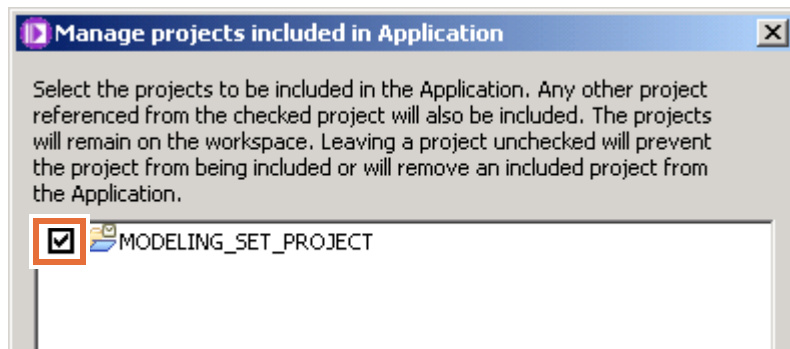
- ___ f. Click **Next**.
- ___ g. The XML document is already selected as the type of message data that you work with most often.

Select the **Binary data** and **Text data** options so that this message model also supports binary data and text data.



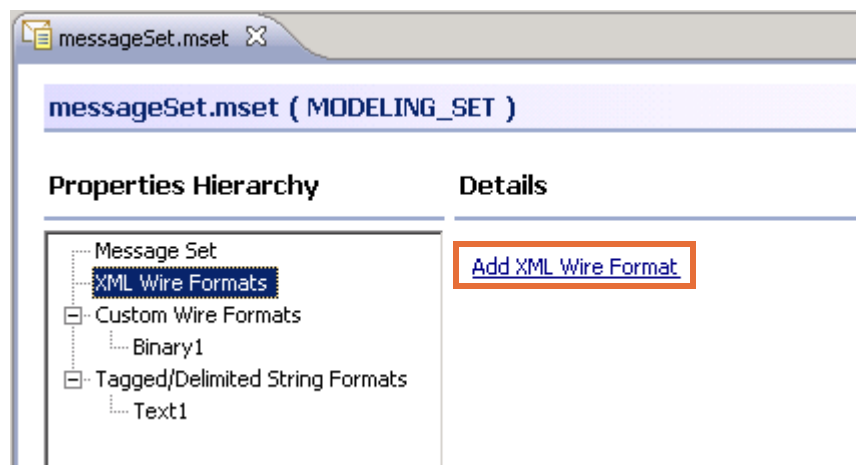
- ___ h. Click **Finish**. The Message Set editor opens with the file messageSet.mset for the MODELING_SET message set.
- ___ i. Click **OK** on the **Tip** window.
- ___ 2. Modify the **RouteComplaint** application to reference the MODELING_SET_PROJECT.
- ___ a. In the Application Development view, right-click the **RouteComplaint** application and then select **Manage included projects**. The **Manage projects included in Application** window opens.

- ___ b. Select `MODELING_SET_PROJECT`, and then click **OK**.



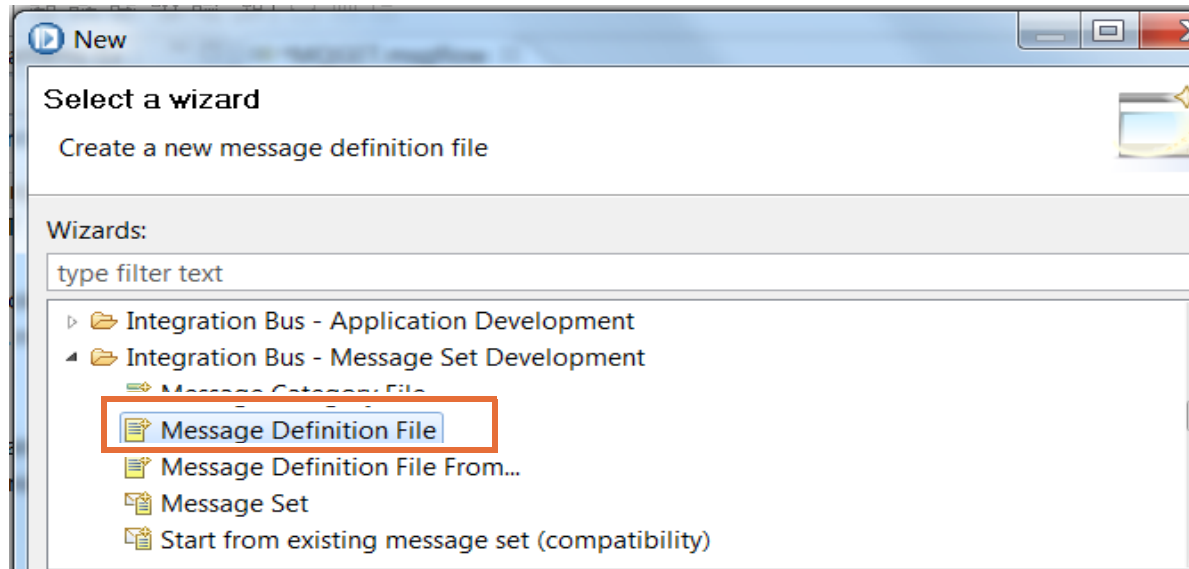
The message set project is moved into the **RouteComplaint** application under the **Message Sets** folder.

- ___ 3. The message flow in the **RouteComplaint** application converts a COBOL document to an XML document. The next step is to add the XML1 wire format to the message set.
- ___ a. In the Message set editor for the file `messageSet.mset`, click **XML Wire Formats** in the **Properties Hierarchy** column.
- ___ b. Click the **Add XML Wire Format** link.

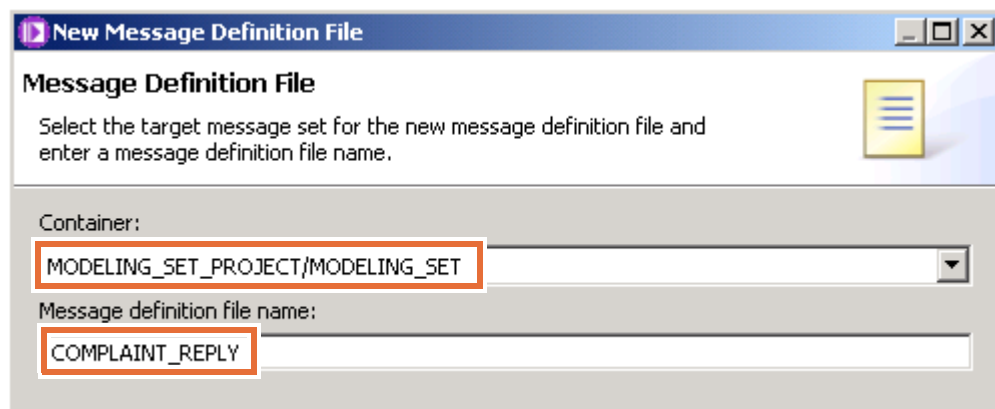


- ___ c. Click **OK** to accept the default XML wire format name of `XML1`.
- ___ d. Save the changes to the `messageSet.mset` file by typing `Ctrl + S` in the message set editor.
- ___ 4. Create a message definition file `COMPLAINT_REPLY.mxsd`.
- ___ a. From the toolbar, click **File > New > Other**. The New wizard is displayed.

- ___ b. Expand **Integration Bus - Message Set Development**, and select **Message Definition File**.



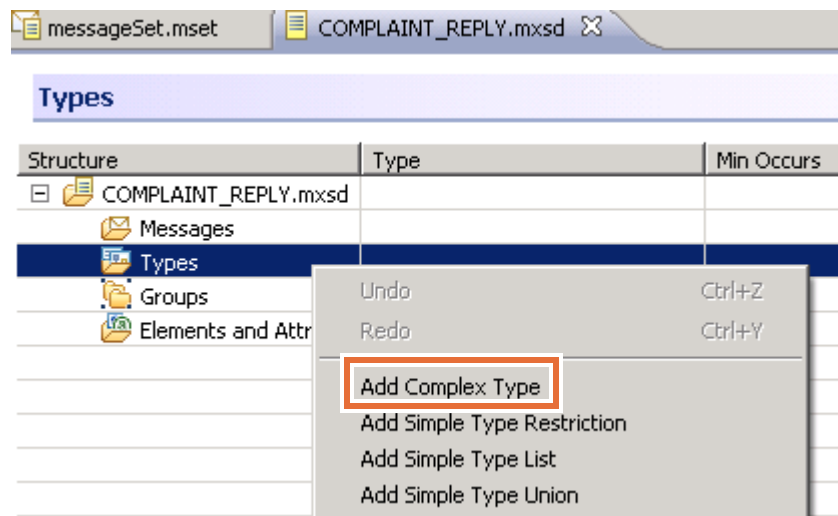
- ___ c. Click **Next**. The New Message Definition File window is displayed.
- ___ d. Verify that **Container** is set to MODELING_SET_PROJECT/MODELING_SET.
- ___ e. For **Message definition file name**, enter: COMPLAINT_REPLY



- ___ f. Click **Finish**. The Message Definition File editor opens with the file COMPLAINT_REPLY.mxsd.
- ___ 5. Create three complex types:
- **t_Complaint_Reply**
 - **t_YourComplaint**
 - **t_Reply**
- t_Complaint_Reply** is a complex type whose elements are **t_YourComplaint** and **t_Reply**.

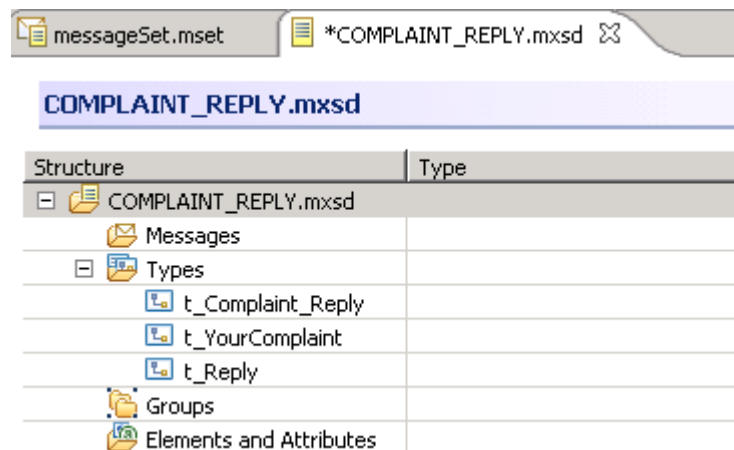
t_YourComplaint and **t_Reply** are complex types whose elements are simple types.

- ___ a. In the Message Definition File editor window for **COMPLAINT_REPLY.mxsd**, right-click **Types**, and then select **Add Complex Type**.



A row is added to the table with the default name **complexType**.

- ___ b. Type over the default name with: **t_Complaint_Reply**
- ___ c. Add another complex type: **t_YourComplaint**
- ___ d. Add a third complex type: **t_Reply**



- ___ 6. Add local elements to the complex types.
- ___ a. Right-click **t_YourComplaint** and click **Add Local Element** from the menu. A row is added to the table with the default name **localElement**.

**Note**

Be sure that you are adding the element to `t_YourComplaint`, not `t_Complaint_Reply`.

- ___ b. Overwrite the default name with: `Type`
The element is added with the default data type of `xsd:string`
- ___ c. Similarly, add two more elements to **`t_YourComplaint`**: `Reference` and `Text`.

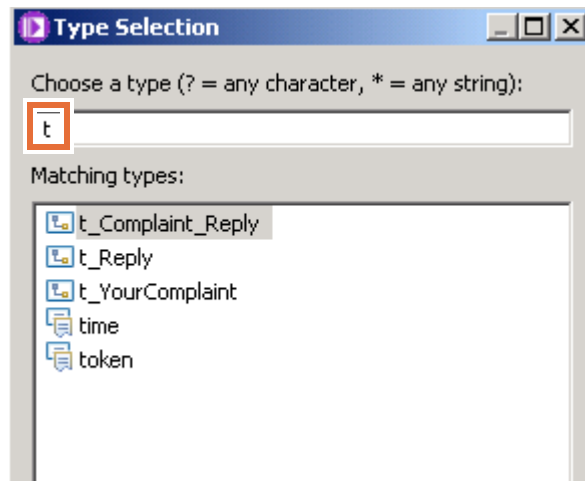
Structure	Type	Min Occurs	Max Occurs
COMPLAINT_REPLY.mxsd			
Messages			
Types			
t_Complaint_Reply			
t_YourComplaint			
Type	xsd:string	1	1
Reference	xsd:string	1	1
Text	xsd:string	1	1
t_Reply			
Groups			
Elements and Attributes			

- ___ d. For the complex type **`t_Reply`**, add the local elements: `OurReference` and `Text`.

Structure	Type	Min Occurs	Max Occurs
COMPLAINT_REPLY.mxsd			
Messages			
Types			
t_Complaint_Reply			
t_YourComplaint			
Type	xsd:string	1	1
Reference	xsd:string	1	1
Text	xsd:string	1	1
t_Reply			
OurReference	xsd:string	1	1
Text	xsd:string	1	1
Groups			
Elements and Attributes			

- ___ e. For the complex type **`t_Complaint_Reply`**, add the local elements `YourComplaint` and `Reply`.

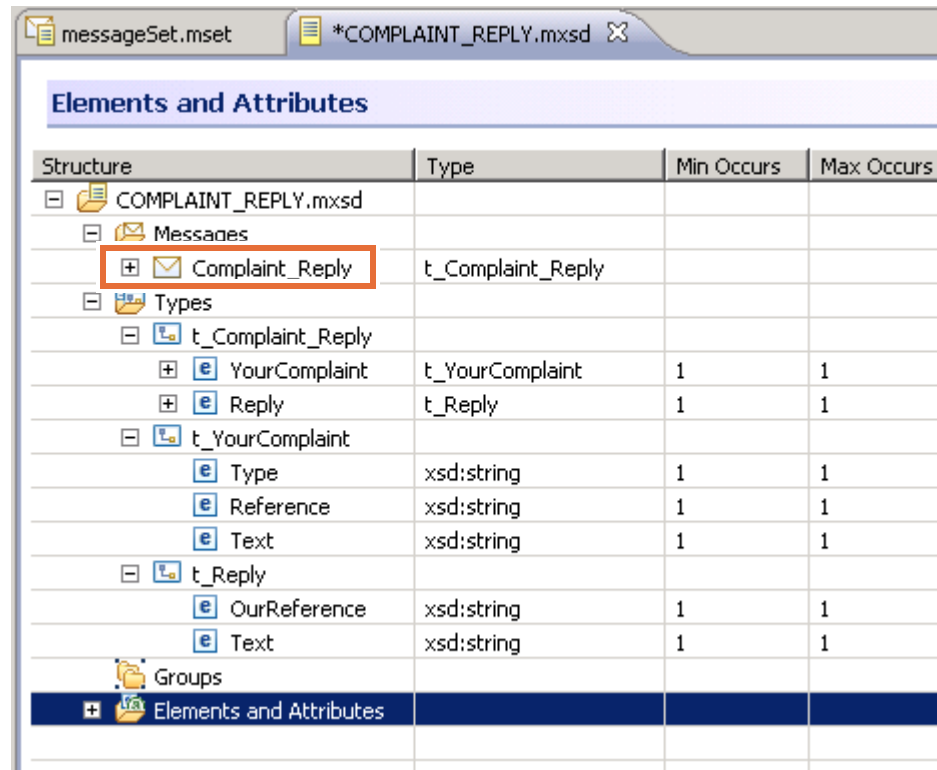
- ___ 7. The elements within **t_Complaint_Reply** were created with the default type `xsd:string`. Change these elements to their complex types `t_YourComplaint` and `t_Reply`.
- ___ a. Under **t_Complaint_Reply**, click **xsd:string** in the **Type** column for the local element **YourComplaint**. A list box is displayed.
- ___ b. Select **(More...)** from the list. The **Type Selection** menu opens.
- ___ c. For **Choose a type**, enter `t` to limit the choices of the custom types to types that begin with the letter `t`.



- ___ d. Select **t_YourComplaint**, and then click **OK**.
- ___ e. Similarly, for the element **Reply**, replace the default type of `xsd:string` with `t_Reply`.

messageSet.mset *COMPLAINT_REPLY.mxsd			
Elements and Attributes			
Structure	Type	Min Occurs	Max Occurs
COMPLAINT_REPLY.mxsd			
Messages			
Types			
t_Complaint_Reply			
YourComplaint	t_YourComplaint	1	1
Reply	t_Reply	1	1
t_YourComplaint			
Type	xsd:string	1	1
Reference	xsd:string	1	1
Text	xsd:string	1	1
t_Reply			
OurReference	xsd:string	1	1
Text	xsd:string	1	1

- ___ 8. Create a message that is named `Complaint_Reply`.
- ___ a. In the Message Definition File editor, right-click **Messages**, and then select **Add Message From Global Type**.
 - ___ b. For **Select a global complex type**, select `t_Complaint_Reply`.
 - ___ c. Click **OK**.
 - ___ d. Rename the new message to `Complaint_Reply` by clicking the message name and deleting `t_`.



Structure	Type	Min Occurs	Max Occurs
COMPLAINT_REPLY.mxsd			
Messages			
Complaint_Reply	t_Complaint_Reply		
Types			
t_Complaint_Reply			
YourComplaint	t_YourComplaint	1	1
Reply	t_Reply	1	1
t_YourComplaint			
Type	xsd:string	1	1
Reference	xsd:string	1	1
Text	xsd:string	1	1
t_Reply			
OurReference	xsd:string	1	1
Text	xsd:string	1	1
Groups			
Elements and Attributes			

- ___ 9. Expand the message **Complaint_Reply** to verify the structure of the message.

Structure	Type
COMPLAINT_REPLY.mxsd	
Messages	
Complaint_Reply	t_Complaint_Reply
YourComplaint	t_YourComplaint
Type	xsd:string
Reference	xsd:string
Text	xsd:string
Reply	t_Reply
OurReference	xsd:string
Text	xsd:string

- ___ 10. Save all open editors by typing Shift + Ctrl + S (or from the toolbar, click **File > Save > All**).

You see a number of warnings in the **Problems** view. They go away as soon as you define the logical and physical properties for each object.

- ___ 11. Set the physical properties for the message set.
- ___ a. In the Application Development view, double-click **messageSet.mset** under **MODELING_SET** to open the Message Set editor (if it is not already open).
 - ___ b. In the **Properties Hierarchy** column, select **XML1** under **XML Wire Formats**.
 - ___ c. Expand **Namespace settings** and set **Output Namespace Declaration** to `As required`.

- ___ d. Expand **XML declaration** and select **Suppress XML Declaration**.

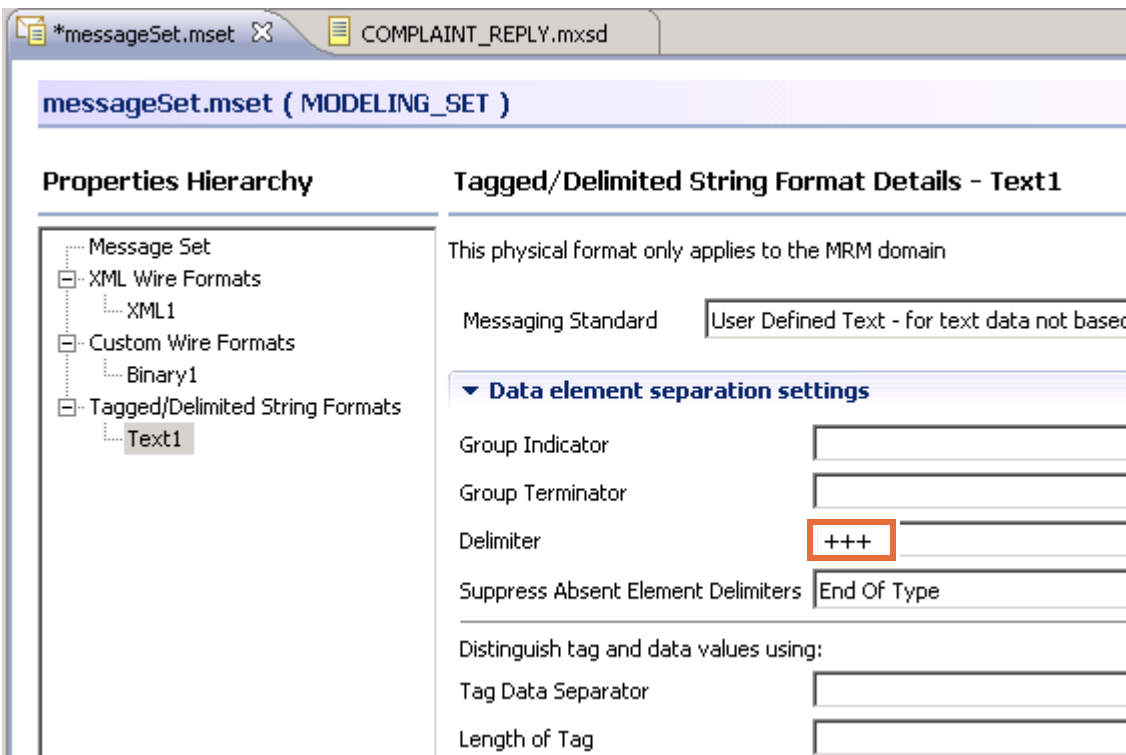
The screenshot shows the IBM WebSphere Message Designer interface. The top bar displays two tabs: `*messageSet.mset` and `*COMPLAINT_REPLY.mxsd`. Below the tabs, the title bar reads `messageSet.mset (MODELING_SET)`.

The interface is divided into two main sections:

- Properties Hierarchy:** A tree view on the left showing the structure of the message set. It includes:
 - Message Set
 - XML Wire Formats
 - XML1** (selected)
 - Custom Wire Formats
 - Binary1
 - Tagged/Delimited String Formats
 - Text1
- XML Wire Format Details - XML1:** A panel on the right showing configuration options for the selected XML1 format.
 - A note states: "This physical format only applies to the MRM domain".
 - Namespace settings:**
 - Namespace declarations: A table with columns "Namespace URI" and "Prefix".
 - Namespace schema locations: A table with columns "Namespace URI" and "Schema location". The first row contains the value "<no target namespace>".
 - Output Namespace Declaration: A dropdown menu set to "As required".
 - XML declaration:**
 - ☒ Suppress XML Declaration
 - XML Version: 1.0
 - XML Encoding: Null
 - Standalone Document: Null

- ___ e. While still in **Properties Hierarchy**, select **Text1** under **Tagged/Delimited String Formats**.

- ___ f. Expand **Data element separation settings** and enter +++ for the **Delimiter**.



- ___ 12. Save all.
- ___ 13. Set the physical properties for type **t_YourComplaint**.
- ___ a. If it is not already open, open the **ComplaintReply.mxsd** file by double-clicking it in the Application Development view.
- ___ b. In the **Outline** view, expand **Types** and select **t_YourComplaint**.
- ___ c. At the bottom of the Message Definition File editor for **COMPLAINT_REPLY.mxsd**, click the **Properties** tab.
- ___ d. In **Properties Hierarchy**, expand **Physical Properties > Text1 > Complex Type**.

- ___ e. Under **Details**, for **Data Element Separation**, select **All Elements Delimited**.

messageSet.mset *COMPLAINT_REPLY.mxsd

t_YourComplaint

Properties Hierarchy

- Logical properties
 - Complex Type
- Physical properties
 - XML1
 - Complex Type
 - Binary1
 - Complex Type
 - Text1
 - Complex Type
- Documentation
 - Complex Type

Details

These physical format properties only apply to the MRM domain

Field identification

Data Element Separation	All Elements Delimited
Group Indicator	
Group Terminator	
Delimiter	+++
Suppress Absent Element Delimiters	End Of Type
<input type="checkbox"/> Observe Element Length	



Note

Verify that **Delimiter** is set to +++, which you specified for the entire message set. If **Delimiter** is empty, verify that you saved the `messageSet.mset` file in previous step.

- ___ f. Perform steps b through e again for **t_Reply** to set **Data Element Separation** to **All Elements Delimited**, and that **Delimiter** is set to +++.
- ___ g. Perform steps b through e again for **t_Complaint_Reply** to set **Data Element Separation** to **All Elements Delimited**, and that **Delimiter** is set to +++.
- ___ 14. In the same way that you set the properties for **messageset.mset** and **t_YourComplaint**, set the logical and physical properties for message, types, and elements within these types, according to the table shown here. Only those values that differ from the default are listed.

The properties for the elements in the first two rows in the table were already set in Steps 12 and 14.

Object	Logical	Binary1	XML1	Text1
messageset.mset			Suppress XML Declaration. Output Namespace Declaration = As required	Delimiter = +++
t_YourComplaint, t_Reply, and t_Complaint_Reply				Data Element Separation = All Elements Delimited
t_YourComplaint > Type	MinOccurs = 0 Default = Other	Length = 10 PadChar = SPACE	Render = XMLAttribute	
t_YourComplaint > Reference	MinOccurs = 0 Default = NOREF	Length = 10 PadChar = SPACE	XML Name = YourReference	
t_YourComplaint > Text	Default = (type a space character)	Length = 200 PadChar = SPACE		
t_Reply > OurReference	Default = (type a space character)	Length = 50		
t_Reply > Text	Default = (type a space character)	Length = 200		



Important

Debugging parser errors is time-consuming; be sure to follow the instructions closely. Pay attention to the field that is being updated:

1. Highlight the field in the **Outline** view.
2. Select the **Properties** tab in the Message Definition File editor to modify the field attributes, paying attention to the field name that is shown at the top of the **Properties** tab.
3. As you complete each update, put a check mark or other indicator in each box.

A common error is forgetting to change the length of elements to a value other than zero. An element cannot contain a zero-length string.

4. Save all open editors (Shift + Ctrl + S).

Most of the warnings in the **Problems** view are now gone. You can safely ignore the rest because you are not using this message for input, or validating it.

Part 3: Extend the message flow

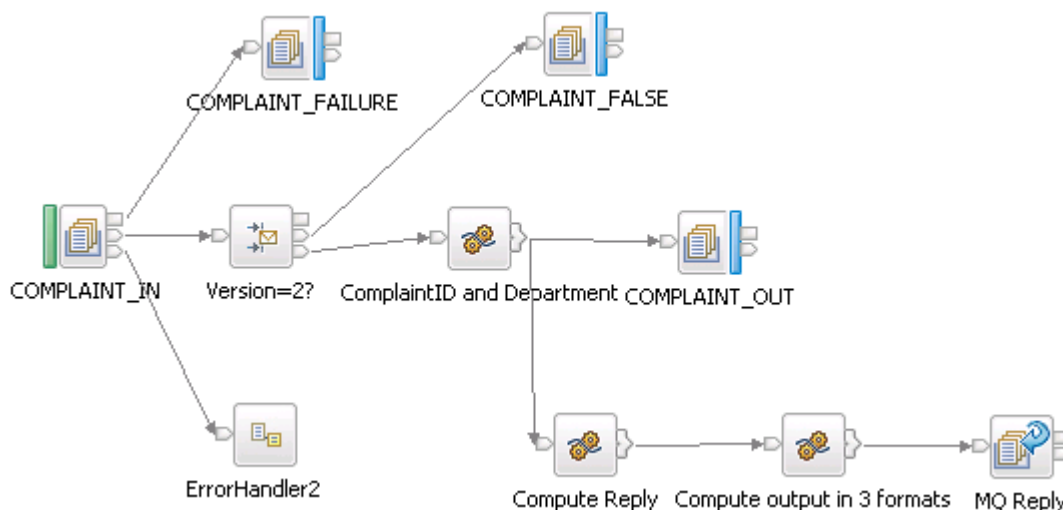
- ___ 1. In the Application Development view, expand **Flows**.
- ___ 2. Double-click **MyFlow.msgflow** to open the flow in the Message Flow editor.
- ___ 3. Add two Compute (or JavaCompute) nodes and one MQReply node.
- ___ 4. Rename the two new Compute (JavaCompute) nodes to **Compute Reply** and **Compute output in 3 formats**.
- ___ 5. Wire the new nodes as follows:
 - ___ a. **Complaint ID and Department** **Compute** node **Out** terminal to the **Compute Reply** **Compute** node **In** terminal.

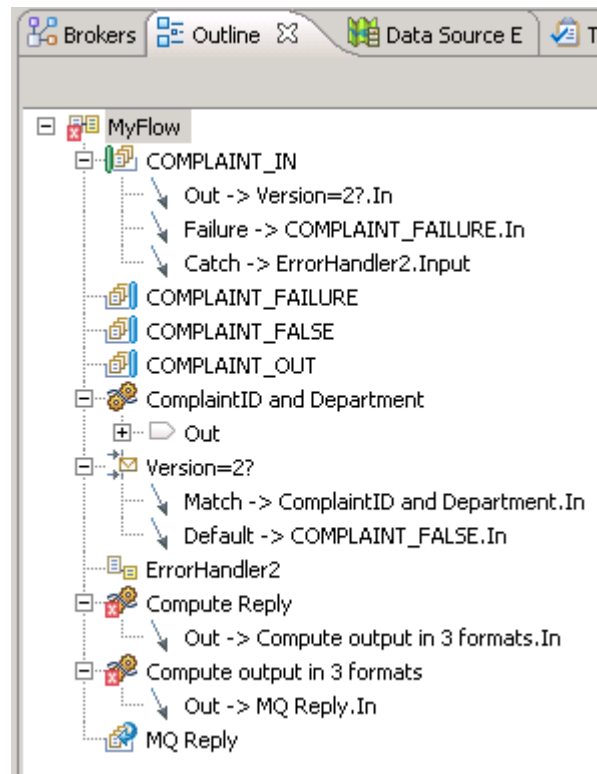


Note

Use care when attaching the wire to the **Compute Reply** Compute node. Since the **Out** terminal is already in use, the message flow editor selects **Out1** as the default terminal. Verify that you wired the **Out** terminal.

- ___ b. **Compute Reply** Compute node **Out** terminal to the **Compute output in 3 formats** Compute node **In** terminal.
- ___ c. **Compute output in 3 formats** Compute node **Out** terminal to the **MQReply In** terminal.





Option 1: Write a transformation in ESQL

- ___ 1. If you used Compute nodes (not JavaCompute nodes), configure the node properties.
 - ___ a. For the **Compute Reply** node, set the **Basic > ESQL module** property to `Compute_Reply`.
 - ___ b. For the **Compute output in 3 formats** node, set the **Basic > ESQL module** property to `Output_in_3_formats`.
- ___ 2. Write extended structured query language (ESQL) for `Compute_Reply`.

You can either create the ESQL based on the following information, or cut and paste it from the `Cut&Paste.txt` file in the `C:\Labs\Lab03-MRM\resources` folder.

- ___ a. Double-click the `Compute_Reply` node. The ESQL editor opens on a new module.

Write ESQL to create a message body in the MRM domain (copy headers only).

- ___ b. Set `OutputRoot.Properties` for message set `MODELING_SET` and message `Complaint_Reply`.

Create the message body. Use content assist (Ctrl + Space) to fill the message path for the MRM part.

- ___ c. The structure of `YourComplaint` is filled with matching fields from `InputBody.CUSTOMERCOMPLAINT.COMPLAINT`.
- ___ d. `Reply.OurReference` is a concatenation of `InputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT` and `'--'` and `InputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE`.
- ___ e. `Reply.Text` is a literal text.

```
CREATE COMPUTER MODULE Compute_Reply
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    CALL CopyMessageHeaders();
    -- Set Properties for MRM domain
    SET OutputRoot.Properties.MessageSet = 'MODELING_SET';
    SET OutputRoot.Properties.MessageType = 'Complaint_Reply';
    -- Fill Complaint_Reply message field by field
    SET OutputRoot.MRM.YourComplaint.Type =
    InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE;
    SET OutputRoot.MRM.YourComplaint.Reference =
    InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_REF;
    SET OutputRoot.MRM.YourComplaint.Text =
    InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TEXT;

    SET OutputRoot.MRM.Reply.OurReference =
    InputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT
    || '-' ||
    InputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE;
    Set OutputRoot.MRM.Reply.Text = 'Your Complaint has been received.';
    RETURN TRUE;
  END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
  DECLARE I INTEGER 1;
  DECLARE J INTEGER;
  SET J = CARDINALITY(InputRoot.*[]);
  WHILE I < J DO
    SET OutputRoot.*[I] = InputRoot.*[I];
    SET I = I + 1;
  END WHILE;
END;

END MODULE;
```

___ 3. Write ESQL for **Compute Output in 3 formats.**

You can either create the ESQL based on the following information, or cut and paste it from the `Cut&Paste.txt` file in the `C:\Labs\Lab03-MRM\resources` folder.

- ___ a. Copy the entire message.
- ___ b. Set the physical format in `OutputRoot.Properties` to `'XML1'`.
- ___ c. Propagate the message without clearing the logical message.
- ___ d. Repeat these steps for the other two physical formats: `Binary1` and `Text1`.

- ___ e. Do not use PROPAGATE for the last message because RETURN propagates automatically to the next node.

```
CREATE COMPUTE MODULE Output_in_3_formats
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    SET OutputRoot = InputRoot;
    SET OutputRoot.Properties.MessageFormat = 'XML1';
    PROPAGATE DELETE NONE;
    SET OutputRoot.Properties.MessageFormat = 'Binary';
    PROPAGATE DELETE NONE;
    SET OutputRoot.Properties.MessageFormat = 'Text1';
    RETURN TRUE;
END;
END MODULE;
```

- ___ 4. Save all changes.

You might notice some Unresolvable message field reference warning messages in the **Problems** view. You can safely ignore these warnings.

Option 2: Write a transformation in the JavaCompute node

- ___ 1. Configure the node properties for the new JavaCompute nodes.
 - ___ a. For the JavaCompute **Compute Reply** node, set the **Basic > Java class** property to `ComputeReply`.
 - ___ b. For the JavaCompute **Output in 3 formats** node, set the **Basic > Java class** property to `OutputIn3Formats`.
- ___ 2. Create a JavaCompute node class `ComputeReply` with the template for **Creating a message class**.
 - ___ a. Double-click the JavaCompute node.
 - ___ b. The wizard guides you. Accept the defaults and select **Creating message class**.
 - ___ c. Write Java code to set `Properties` for the message set to `MODELING_SET` and the message to `Complaint_Reply`.
 - ___ d. Create a message body in the `MRM` domain.
 - ___ e. Create the message body. The structure of `YourComplaint` is filled with matching fields from the input message `CUSTOMERCOMPLAINT.COMPLAINT`.
 - ___ f. `Reply.OurReference` is a concatenation of input `CUSTOMERCOMPLAINT.ADMIN.DEPT` and `'-'` and `CUSTOMERCOMPLAINT.ADMIN.REFERENCE`.
 - ___ g. `Reply.Text` is a literal text.

```
// Add user code below
// Set Properties for MRM domain
outMessage.getRootElement().evaluateXPath(
    "Properties/?MessageSet[set-value('MODELING_SET')]");
outMessage.getRootElement().evaluateXPath(
    "Properties/?MessageType[set-value('Complaint_Reply')]");

// Create message body with MRM domain
outMessage.getRootElement().createElementAsLastChild("MRM");

// Fill Complaint_Reply message body
outMessage.evaluateXPath("?YourComplaint/?Type[set-value('"
    + inMessage.evaluateXPath("string(CUSTOMERCOMPLAINT/COMPLAINT/C_TYPE)") + "')]);
outMessage.evaluateXPath("?YourComplaint/?Reference[set-value('"
    + inMessage.evaluateXPath("string(CUSTOMERCOMPLAINT/COMPLAINT/C_REF)") + "')]);
outMessage.evaluateXPath("?YourComplaint/?Text[set-value('"
    + inMessage.evaluateXPath("string(CUSTOMERCOMPLAINT/COMPLAINT/C_TEXT)") + "')]);

outMessage.evaluateXPath("?Reply/?OurReference[set-value('"
    + inMessage.evaluateXPath("string(CUSTOMERCOMPLAINT/ADMIN/DEPT)") + "-"
    + inMessage.evaluateXPath("string(CUSTOMERCOMPLAINT/ADMIN/REFERENCE)") + "')]);
outMessage.evaluateXPath("?Reply/?Text[set-value('Your complaint has been
received.')]");
// End of user code
```



Note

You can copy the Java code from the C:\Labs\Lab03-MRM\resources\Cut&Paste.txt file.

- ___ 3. Create a Java Compute Node class `OutputIn3Formats` with the template for **Modifying message class**.
 - ___ a. Set the physical format in `Properties` to `'XML1'`.
 - ___ b. Finalize the message and propagate it to out.

___ c. Repeat these steps for the other two physical formats: Binary1 and Text1.

```
// Add user code below
outMessage.getRootElement().evaluateXPath(
    "Properties/?MessageFormat[set-value('XML1')]");
outMessage.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);

outMessage.getRootElement().evaluateXPath(
    "Properties/?MessageFormat[set-value('Binary1')]");
outMessage.finalizeMessage(MbMessage.FINALIZE_NONE);
out.propagate(outAssembly);

outMessage.getRootElement().evaluateXPath(
    "Properties/?MessageFormat[set-value('Text1')]");
outMessage.finalizeMessage(MbMessage.FINALIZE_NONE);
// End of user code
```


Part 4: Test with the Test Client

- ___ 1. Start the Test Client for `Route.msgflow`.
 - ___ a. Right-click the **COMPLAINT_IN MQInput** node and select **Test** to start the Test Client.
 - ___ b. When you are prompted to confirm that **The owning application or library will be deployed**, click **OK**. The Test Client starts.
- ___ 2. The input node expects an XML message, but this message flow is not associated with a message definition. You must import the source data from the following file:
`C:\Labs\Lab03-MRM\data\Complaint_2d.xml`.
 - ___ a. Under **Detailed Properties**, in the **Message** section, change **Body** to **Import from external file**.

and Message to run.

► **General Properties**

▼ **Detailed Properties**

Message Flow: /RouteComplaint/MyFlow.msgflow

Input node: COMPLAINT_IN

Message

► **Header**

Body: Import from external file

File name: Workspace... File system... Edit...

☐ Show in hexadecimal viewer (Read Only)

Send Message

- ___ b. Click **File system**, and then browse to `C:\Labs\Lab03-MRM\data` and import the `Complaint_2d.xml` file.
- ___ 3. Configure the Test Client for this message flow.

You can expect four output messages (one on the `COMPLAINT_OUT` queue, and three on the `COMPLAINT_REPLY` queue) so you must change the Test Client settings to allow for multiple messages.

 - ___ a. Click the **Configuration** tab of the Test Client.
 - ___ b. Click **MQ Settings**.

- ___ c. Clear the **Stop when the first MQ message is received** option.

▼ **MQ Settings**

Stop when the first MQ message is received

The Test Client listens to MQ queues defined in the MQOutput nodes in the message flows being tested. Checking the option on will instruct the Test Client to stop testing when the first message reaches any of the MQ queues being listened to.

☐ Stop when the first MQ message is received



Note

Now the Test Client waits indefinitely for output messages, rather than timing out. You must explicitly stop it to rerun or reinvoke the test.

- ___ 4. The message flow contains an MQReply node so you must configure the MQMD **Reply to queue name** in the Test Client.
- ___ a. In the **Configuration** tab of the Test Client, click **MQ Message Header "Default Header"**.
- ___ b. In the **Reply to queue name** field, enter: COMPLAINT_REPLY

▼ **MQ Message Header "Default Header"**

Accounting token:		Message type:	8
Application origin data:		Message sequence number:	1
Application id data:		Offset:	0
Backout count:	0	Original length:	-1
Character set:	0	Persistence:	2
Correlation id:		Priority:	-1
Encoding:	0	Put application name:	
Expiry:	-1	Put application type:	0
Feedback:	0	Put date/time:	
Format:		Report:	0
Group id:		Reply to queue manager name:	
Message id:		Reply to queue name:	COMPLAINT_REPLY
Message flags:	0	User id:	

☐ Include RFH V2 header

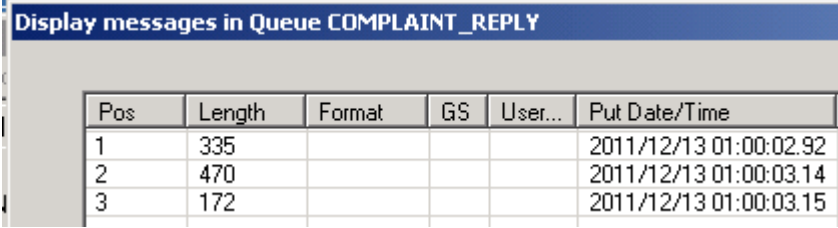
- ___ 5. Send the test message.
- ___ a. Click the **Events** tab.
- ___ b. Click **Send Message**. The Select Deployment Location is displayed.

- ___ c. Click **Finish**.
- ___ 6. Inspect the outcome of this test. You should see that the message was sent to the COMPLAINT_OUT queue.

Use IBM Integration Explorer or RFHUtil to verify that three output messages were sent to the COMPLAINT_REPLY queue and that they are in the correct format as described in the exercise introduction.

To use RFHUtil to review the queues, complete the following steps:

- ___ a. Start RFHUtil.
- ___ b. For **Queue Manager**, select **IB9QMGR**.
- ___ c. For **Queue Name**, select **COMPLAINT_REPLY**.
- ___ d. Click **Display Q**. Three messages should be listed in the display list.



Pos	Length	Format	GS	User...	Put Date/Time
1	335				2011/12/13 01:00:02.92
2	470				2011/12/13 01:00:03.14
3	172				2011/12/13 01:00:03.15

- ___ e. Select the first message, and then click **Browse Q**. You are returned to the main RFHUtil page.
- ___ f. Click the **Data** tab.
- ___ g. Under **Data Format**, click **XML** to format the message correctly. Verify that the content is correct.
- ___ h. Click the **Main** tab.
- ___ i. Click **Display Q** again, and select the second record. Repeat the previous steps (changing the Data Format selection as appropriate) to review the second message.
- ___ j. Repeat the steps for the third message on the queue.
- ___ 7. If necessary, correct the source code and test again.

Part 5: Optional: Nonexistent reply-to queue

If you completed the exercise ahead of schedule, try the optional exercise.

Test your message flow with a test message (modify the RFHUtil page MQMD) that:

1. Contains **no** reply-to queue specification.
2. Contains **no** reply-to queue specification and Message Type = 1 (Request).
3. Contains a reply-to queue specification that does **not exist** in your queue manager.

- What happens?
- Can you explain it?

Part 6: Clean up the environment

- ___ 1. Close all open editors. You can right-click any editor tab and click **Close All**.
- ___ 2. In the **Integration Nodes** view, right-click the **default** integration server and click **Delete > All Flows and Resources**.

End of exercise

Solution

You are not able to put a message with **no** reply-to queue because IBM WebSphere MQ checks this field if MQMD.MSGTYPE = Request was specified. The MQPut returns a reason code of 2027 (Missing ReplyToQ).

If you change MSGTYPE to Datagram (8), the message flow is rolled back and the original message is written to the FAILURE queue.

The same happens when you specify an unknown reply-to queue.

End of solution

Exercise 4. Implementing web services with IBM Integration Bus

What this exercise is about

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

What you should be able to do

At the end of the exercise, you should be able to:

- Import a WSDL file
- 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 monitor message traffic

Introduction

In this exercise, you implement the SOAP nodes example that is included in the IBM Integration Bus samples gallery. This sample application provides a foundation example on how to implement web services within a set of message flows. The message flows act as a producer of web services and a consumer of them. You review the components of the message flows and how they are configured. You then test the message flows with a simple message.

Next, you start the TCP/IP Monitor that is contained within the IBM Integration Toolkit. The monitor allows you 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.

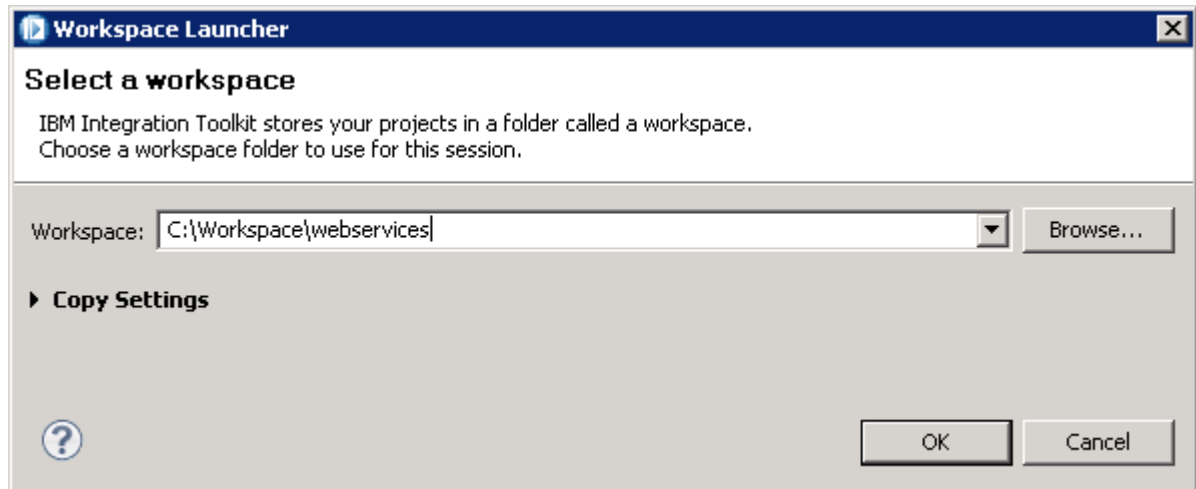
Requirements

- A workstation with the WebSphere MQ and IBM Integration Bus components installed
- The IBM Integration Bus default configuration

Exercise instructions

Part 1: Start components and prepare the workspace

- ___ 1. Make sure that all local IBM Integration Bus components are running.
- ___ 2. Start the IBM Integration Toolkit if it is not already running.
- ___ 3. Open a new workspace, such as: C:\Workspace\webservices, and then click **OK**.

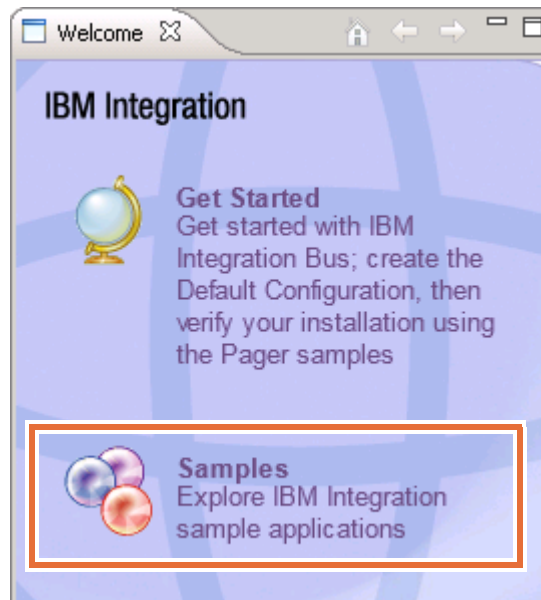


- ___ 4. When the IBM Integration Toolkit opens, go to the **Application Development** perspective.

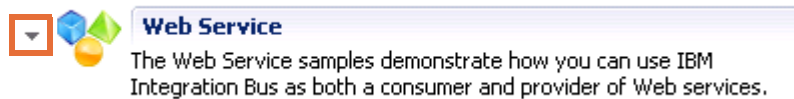
Part 2: Import and review the SOAP nodes sample

In this part of the exercise, you import the SOAP nodes web service application. Review the documentation for the SOAP nodes sample application in the Information Center.

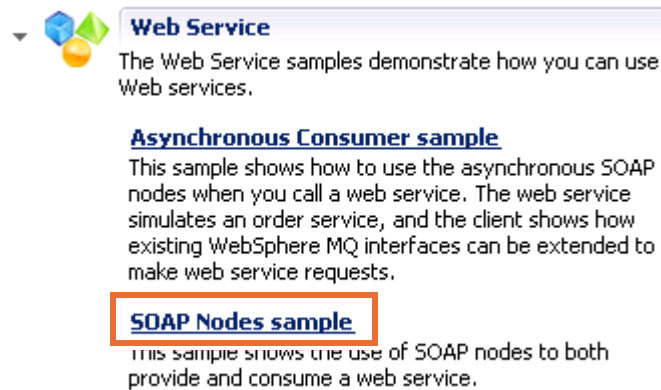
- ___ 1. Review the SOAP node sample application from the samples gallery.
 - ___ a. In the **Quick Start** section in the right pane, click **Start from Samples**. The **Samples and Tutorials** gallery is displayed.



- ___ b. Scroll to the bottom of the window, and expand **Web Service**.



- ___ c. Click **SOAP Nodes sample**.



The Information Center opens, and displays a description about the sample and the message flows it contains.

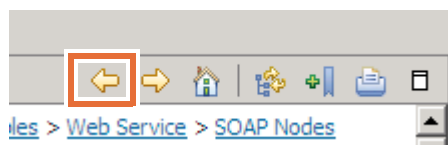
- ___ d. Click the **Read about the sample** link, and review the information about the sample.



Information

The SOAP node example creates two message flows: a web service consumer and a web service provide. Both message flows consist of a main flow and a subflow. You review the two flows in greater detail later in the exercise.

- ___ e. After you finish reading about the sample, click the **Back** arrow in the toolbar at the top of the page. The main sample page is displayed.

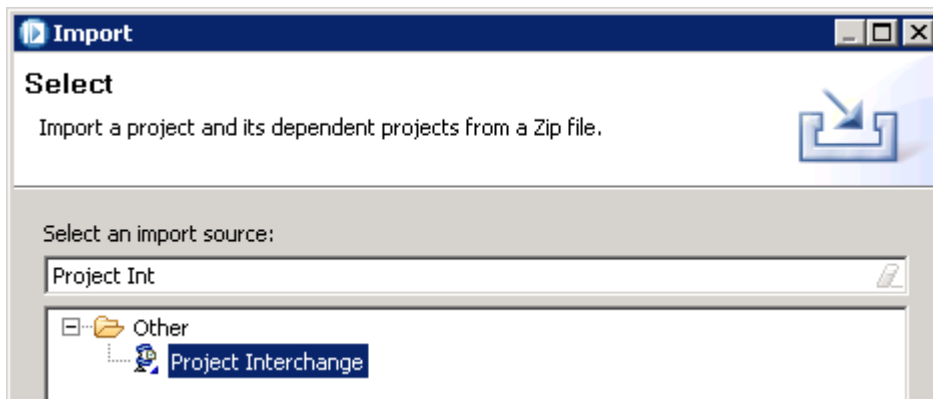


Note

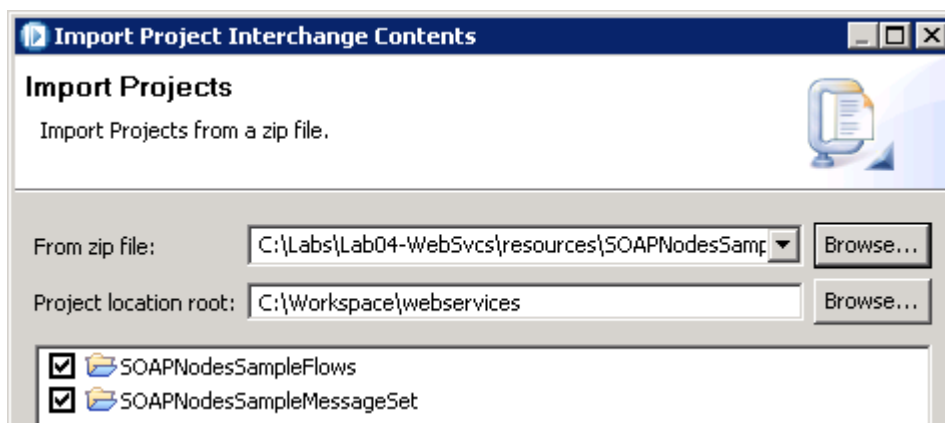
For this exercise, use the copy of the SOAP nodes sample application from the lab files instead of the copy in the samples gallery.

- ___ 2. Import the SOAP node sample application from the lab files directory.
- ___ a. From the main menu bar, select **File > Import**.

- ___ b. In the Import wizard, select **Other > Project Interchange**.

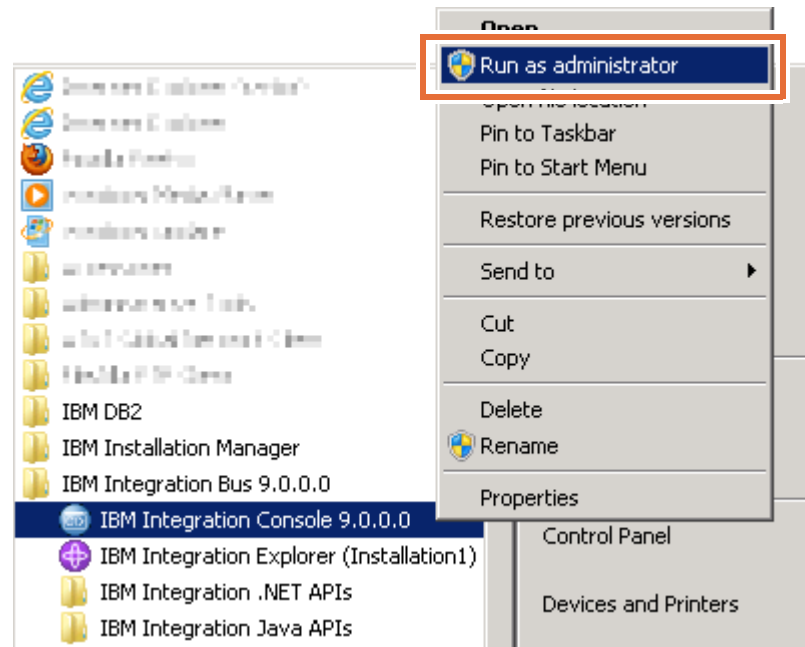


- ___ c. Click **Next**.
- ___ d. In the Import Projects wizard page, select **Browse** next to the **From Zip File** field.
- ___ e. Go to the `C:\Labs\Lab04-WebSvc\resources\` folder and select `SOAPNodesSample.zip`.
- ___ f. Back in the Import Projects page, select **Select All** to import both of the listed projects.



- ___ g. Click **Finish**.
- ___ 3. Define three local message queues for the SOAP nodes sample application: `SOAPSAMPLE_FAULT`, `SOAPSAMPLE_IN`, and `SOAPSAMPLE_OUT`.
- ___ a. From the Windows Start menu, expand **All Programs > IBM Integration Bus 9.0.0.0 > IBM Integration Console 9.0.0.0**.

- ___ b. Right-click **IBM Integration Console** and click **Run as administrator**.

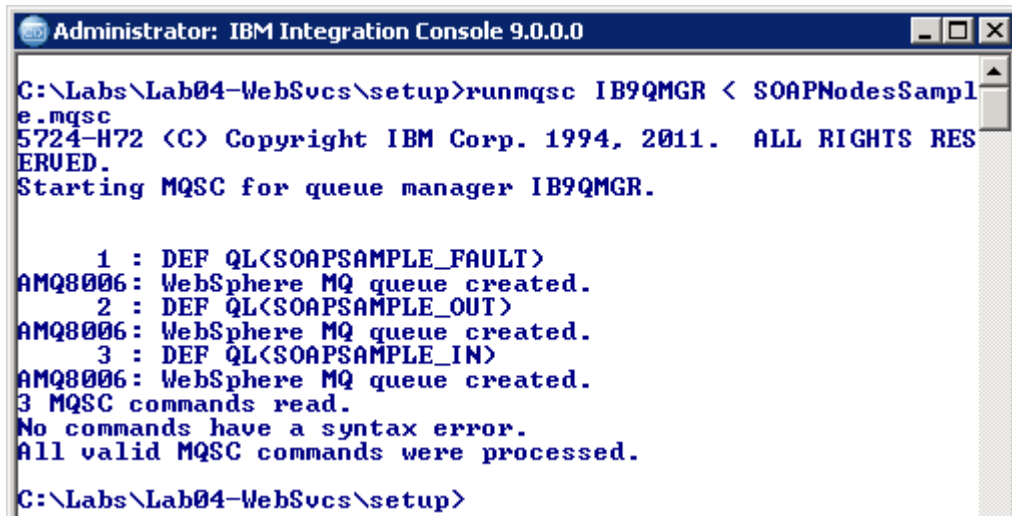


Note

If you attempt to use a regular command prompt to enter an `mqsi` command, the command fails because the appropriate environment variables are not set. Always use the IBM Integration Console when you enter an `mqsi` command.

- ___ c. In the User Account Control dialog box, click **Yes** to run the application as an administrator.
- ___ d. In the command prompt, type the command:
`runmqsc IB9QMGR < SOAPNodesSamples.mqsc`

- ___ e. Confirm that the script created three local queues for the application.



```

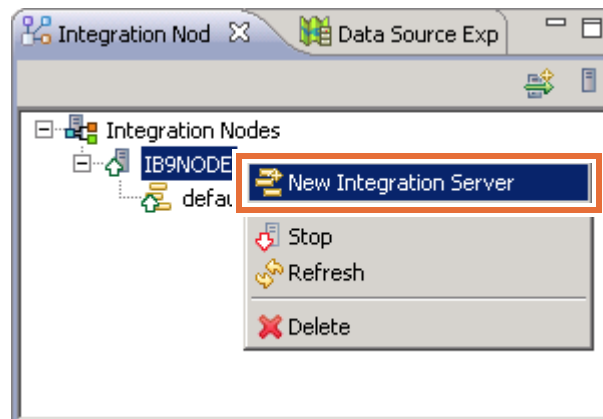
Administrator: IBM Integration Console 9.0.0.0
C:\Labs\Lab04-WebSvc\setup>runmqsc IB9QMGR < SOAPNodesSample.mqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2011. ALL RIGHTS RESERVED.
Starting MQSC for queue manager IB9QMGR.

      1 : DEF QL(SOAPSAMPLE_FAULT)
AMQ8006: WebSphere MQ queue created.
      2 : DEF QL(SOAPSAMPLE_OUT)
AMQ8006: WebSphere MQ queue created.
      3 : DEF QL(SOAPSAMPLE_IN)
AMQ8006: WebSphere MQ queue created.
      3 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

C:\Labs\Lab04-WebSvc\setup>

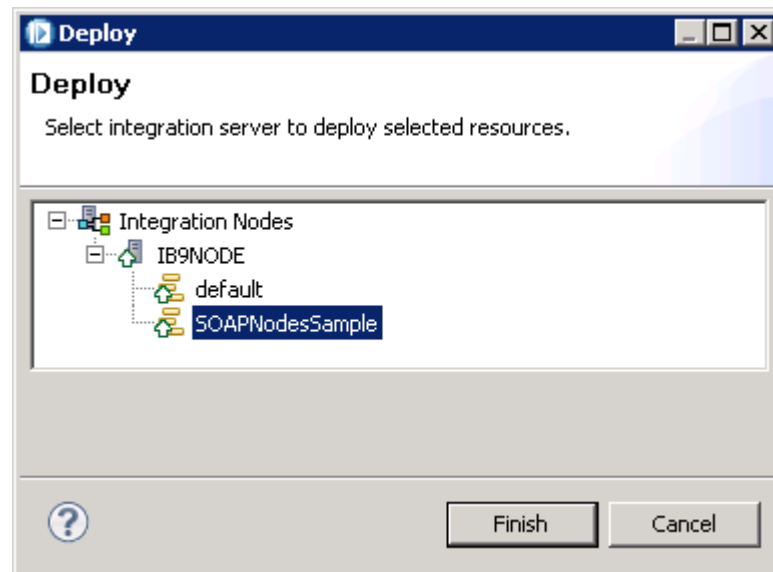
```

- ___ 4. Create an integration server, SOAPNodesSample, to manage the application.
- ___ a. Switch to the IBM Integration Toolkit.
- ___ b. In the **Integration Nodes** view, select the **IB9Node** integration node.
- ___ c. Right-click the selection and click **New Integration Server**.



- ___ d. Enter **SOAPNodesSample** as the integration server name.
- ___ e. Click **OK**.
- ___ 5. Deploy the SOAP nodes sample application to the SOAPNodesSample integration server.
- ___ a. From the Application Development view, click **BARs > SOAPNodesSampleBAR.bar > SOAPNodesSampleFlows/SOAPNodesSampleBAR.bar**.
- ___ b. Right-click the selection and click **Deploy**.

- ___ c. Click **SOAPNodesSample** as the integration server.



- ___ d. Click **Finish**.
- ___ e. Verify that the application deployed successfully in the **Deployment Log** view.

Part 3: Examine the SOAP Nodes Sample application

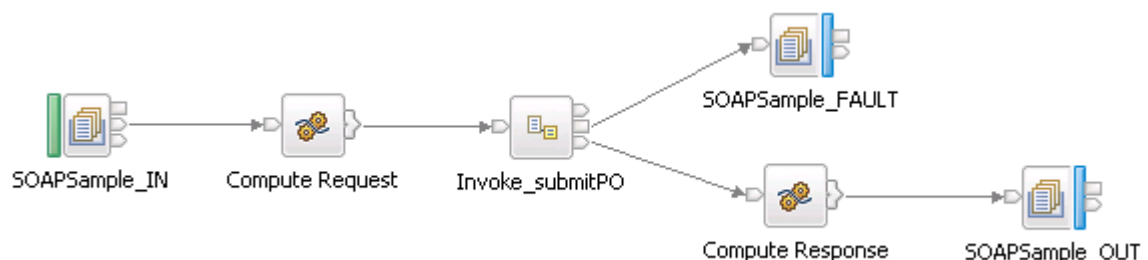
Before you test the sample application, examine the message flows for the web service consumer and web service provider parts of the application.

- ___ 1. Review the web service consumer message flow.
- ___ a. Expand **Independent Resources > SOAPNodesSampleFlows > (default broker schema)**.
- ___ b. Double-click **SOAPNodesSampleConsumerFlow.msgflow** to open the message flow in the message flow editor.



Information

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

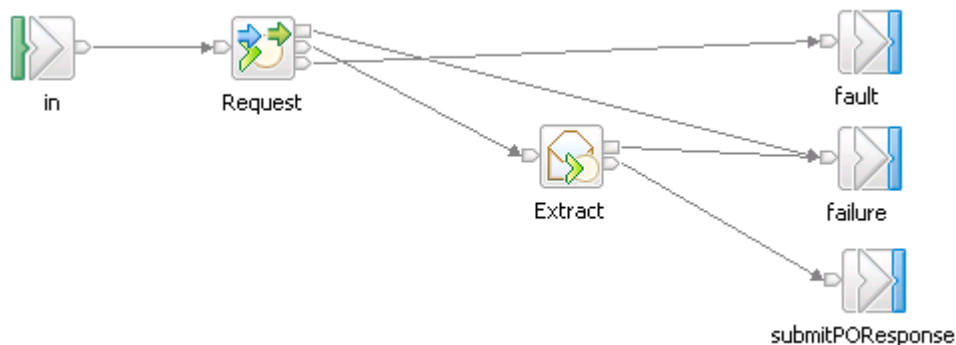


- **SOAPSAMPLE_IN** is an MQInput node. It accepts an XML message from the SOAPSAMPLE_IN queue to start the message consumer flow.

- The Compute node **Compute Request** modifies the incoming XML message in preparation for sending it to the web service.
- **Invoke_submitPO** calls the subflow that issues the SOAP request. This subflow is explained further in an upcoming step.
- **SOAPSAMPLE_FAULT** is a WebSphere MQ Output node that writes a fault message to the SOAPSAMPLE_FAULT queue if a failure occurs in the subflow.
- The Compute note **Compute Response** formats an output XML message that was returned from successful invocation of the SOAP service in the subflow.
- The MQOutput node **SOAPSAMPLE_OUT** writes the formatted message to queue SOAPSAMPLE_OUT.

___ 2. Examine the **submitPO** message flow.

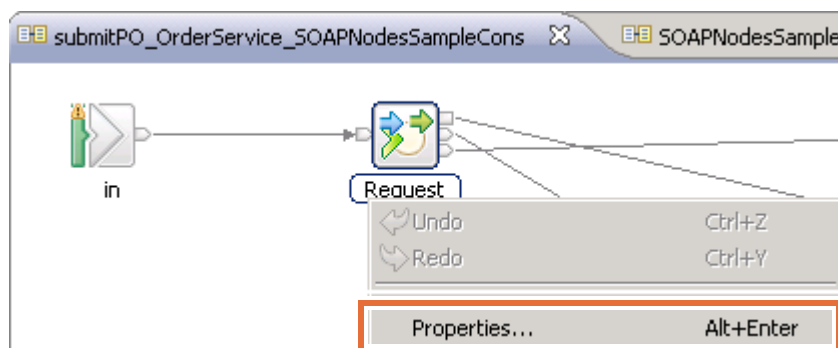
- ___ a. Double-click the **Invoke_submitPO** subflow node in the message flow. This operation opens the subflow of the message consumer in the message flow editor.



- ___ b. Examine the **Request** node in the **submitPO** subflow.

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

- ___ c. Right-click the node and select **Properties**.



___ d. Examine the properties in the **Basic** property category.

Operation mode	
WSDL Properties	
WSDL file name*	SOAPNodesSampleMessageSet/com/acmeorders/www/orderservice/OrderService.wsdl
Port type*	OrderService
Imported binding*	OrderServiceSOAP
Binding operation*	submitPO
Service port*	OrderServiceHTTPSOAP
Target namespace:	http://www.acmeOrders.com/OrderService
Transport	HTTP



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.

___ e. Click the **HTTP Transport** tab and note the **Web service URL** value.

Web service URL *	http://localhost:7800/acmeOrders/WADDR/ProcessOrders <i>e.g. http://server/path/to/service</i>
Request timeout (in seconds)	120
HTTP(S) proxy location	<enter your proxy server (if any)>
Protocol (if using SSL)	TLS
Allowed SSL ciphers (if using SSL)	<enter any specific SSL Ciphers you wish to use>
Use compression	none
Accept compressed responses by default	<input type="checkbox"/>
Enable SSL Certificate Hostname Checking	<input type="checkbox"/>

The SOAPRequest node sends a web service request message to the service at the specified address.

- ___ f. Click the **Response Message Parsing** property tab and note the message model that is displayed.

This message model is the message set definition (`messageSet.mset`) that is displayed in the **SOAPNodesSampleMessageSet** project in the **Application Development** perspective.

Settings for working with the response message parsers.

Message domain	SOAP : For SOAP Web Services (WS-Standards support)
Message model	SOAPNodesSampleMessageSet (NUV4DUS002001)
Message	
Physical format	

You can verify the property by double-clicking **messageSet.mset** in the **SOAPNodesSampleMessageSet** project to open it in the message set editor. The unique identifier that is displayed in the **ResponseMessageParsing** property tab matches the one shown in the **messageSet.mset**.

- ___ g. Examine the **Extract** node in the submitPO subflow.

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

- ___ h. Examine the **ws_submitPOResponse** Label node in the submitPO subflow. It receives the message from the SOAPExtract node, and then exits the subflow.

- ___ 3. Review the web service provider message flow.

- ___ a. In the Application Development view, expand **Independent Resources > SOAPNodesSampleFlows > (default broker schema)**.
- ___ b. Double-click **SOAPNodesSampleProviderFlow.msgflow** to open the message flow.



Information

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

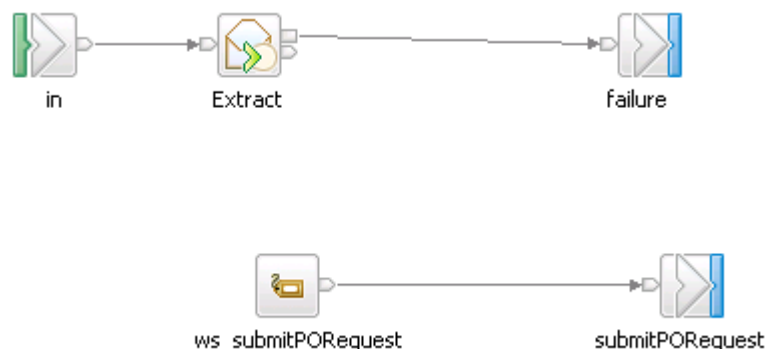


- **Input** is a SOAPInput node. It receives an incoming SOAP message, validates it, and then passes it on.
- **OrderService_Extract** calls the subflow that extracts the SOAP envelope from the message. The subflow is explained further in an upcoming step.

- The **Compute Response** Compute node 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 SOAPReply 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.

___ 4. 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.



Information

The SOAPExtract node **Extract** removes the SOAP envelope. As you saw previously, removing the SOAP envelope leaves an XML message that can be altered or transformed, by using a Compute or Mapping node, for example.

___ 5. Examine the **Extract** node properties.

- ___ a. Review the properties of the **Extract** node. Right-click the node, click **Properties**, and then click the **Basic** tab.

Remove envelope	<input checked="" type="checkbox"/>
Envelope Destination	<input type="text" value="\$LocalEnvironment/SOAP/Envelope/InRequest/OrderServiceSOAP"/>
Destination path mode	<input checked="" type="radio"/> Create path <input type="radio"/> XPath location of existing element
Route to 'operation' label	<input checked="" type="checkbox"/>
Label Prefix	<input type="text" value="ws_"/>

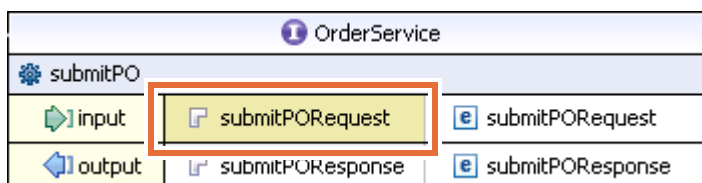


Information

When you select the **Route to 'operation'** label option, the message node routes the message to a Label node that matches the operation name in the SOAP request message.

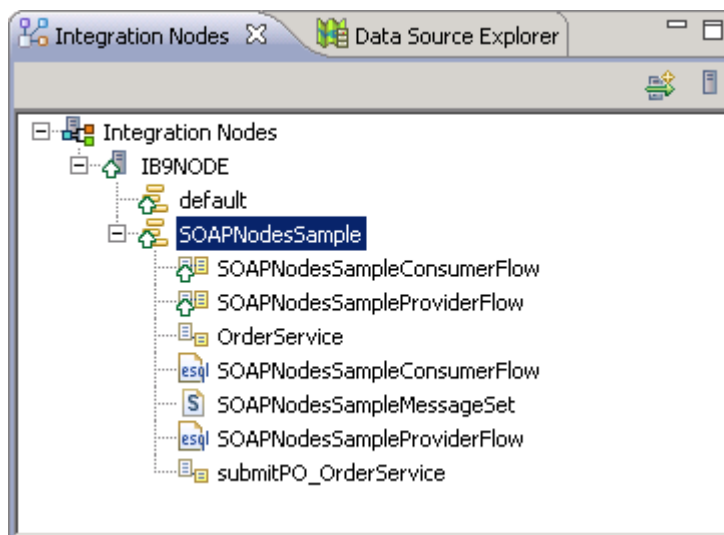
The **Label Prefix** concatenates a value to the operation name. In this example, the prefix that is attached to the operation name is **'ws_'**.

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 case, the request message name for the submitPO operation is **subPORequest**.



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

- ___ b. The Label node **ws_subPORequest**, as explained in the preceding item, is the target of the flow of control from the SOAPEXtract node.
- ___ 6. Confirm that the SOAP node message flows and message sets are deployed in the **Integration Nodes** view.



Part 4: Run the SOAP nodes sample

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

- ___ 1. Review the instructions for configuring and running the SOAP nodes sample application.
 - ___ a. Return to the SOAP Nodes sample in the IBM Integration Bus Information Center.
 - ___ b. Click **Run the sample**.



[Run the sample](#)

The wizard displays the instructions for configuring the environment and running the sample.



Note

You are using the SOAP over HTTP transport for this exercise, although the sample allows you to use either SOAP/HTTP or SOAP/JMS. As the instructions state, the default port that IBM Integration Bus uses is 7800. The port assignment that the SOAPRequest node uses must match the port assignment that the integration server uses.

To verify that these port assignments match, type the `mqsireportproperties` command against the integration server to show the port assignment. You also must verify that the port that the SOAPRequest node uses is set correctly.

- ___ 2. Check the HTTP Connector port number that is assigned to the SOAPNodesSample application.

In the IBM Integration Console, issue the command:

```
mqsireportproperties IB9NODE -e SOAPNodesSample -o HTTPConnector -n
port
```

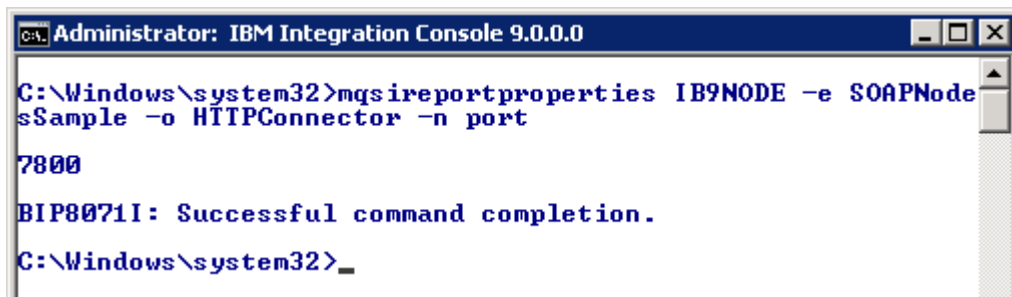
The command queries the integration node that is named **IB9NODE** with the integration server **SOAPNodesSample** for the value of the **HTTPConnector port**.



Important

When you use `mqsi` commands, remember:

- The commands are case-sensitive.
- Enter the `mqsi` command as a single line; that is, do not type a `Return` in the command. Allow the command to wrap in the command window if necessary.



```
Administrator: IBM Integration Console 9.0.0.0
C:\Windows\system32>mgsireportproperties IB9NODE -e SOAPNode
sSample -o HTTPConnector -n port
7800
BIP8071I: Successful command completion.
C:\Windows\system32>
```

In this case, the default port (7800) is in use.



Information

The SOAPRequest node in the web service consumer 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.

To review the setting for the web service, refer to **Part 3, step 2e** in this exercise.

___ 3. Run the sample.

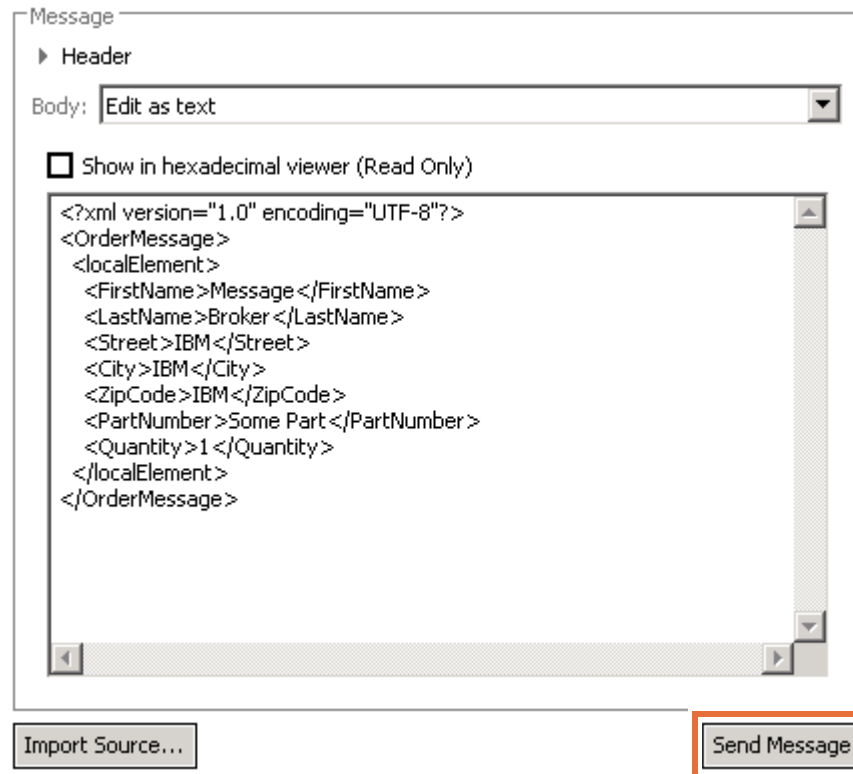
- ___ a. In the **Application Development** view, expand **Independent Resources > SOAPNodesSampleFlows > Flow tests**.
- ___ b. Double-click **SOAPNodesSampleConsumerFlow.mbtest**. The Test Client window opens.
- ___ c. Review the message that you are sending. You can enlarge the window to do so.



Note

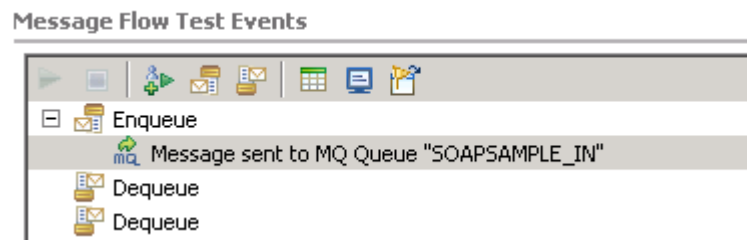
To maximize the Test Client, double-click the tab. Double-click its tab again to return to the default size.

- ___ 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 issue the request.



- ___ 4. If the application runs successfully, a message is enqueued on the SOAPSAMPLE_OUT queue. Use the WebSphere MQ Explorer to view the queues.
- ___ a. Start WebSphere MQ Explorer by clicking **All Programs > WebSphere MQ > WebSphere MQ Explorer (Installation 1)**.
- ___ b. Expand **IBM WebSphere MQ > Queue Managers > IB9QMGR**.
- ___ c. Double-click **Queues**.

- ___ d. In the right pane (**WebSphere MQ Explorer - Content**), check the **Current queue depth** column.

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

A message is on the queue.

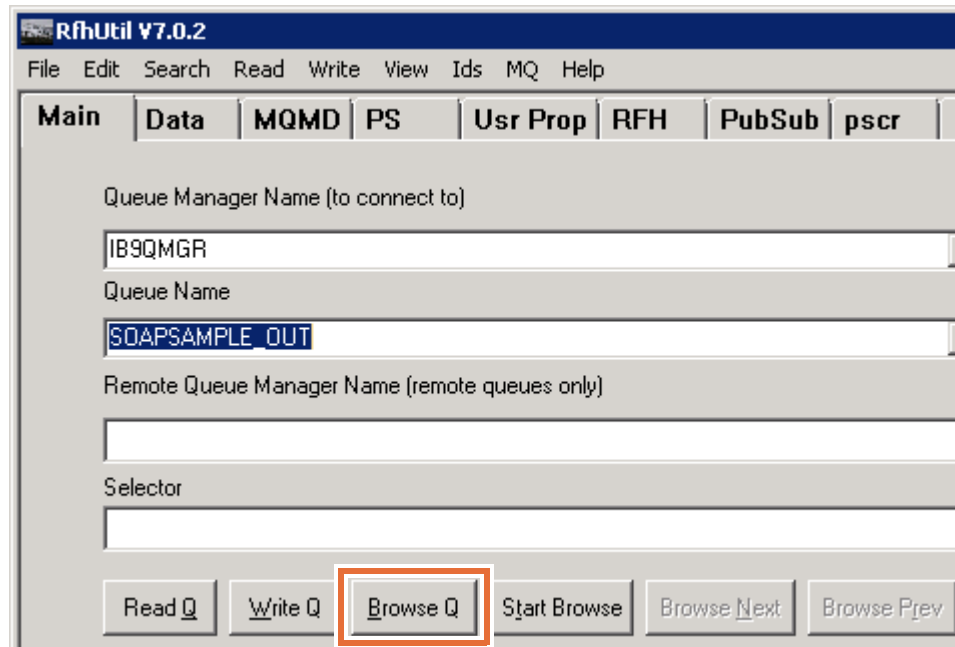
- ___ e. To browse the message, right-click **SOAPSAMPLE_OUT**, and then select **Browse Messages**.
- ___ f. To view the detailed message, right-click the row in the table that contains message 1, and then click **Properties**.

Message browser				
Queue Manager Name: IB9QMGR				
Queue Name: SOAPSAMPLE_OUT				
Position	Put date/time	User identifier	Put application name	Format
1	Aug 16, 2013 1:14:44 PM	SYSTEM	0.0.0\bin\DataFlowEngine.exe	MQSTR
Compare with...				
Properties...				

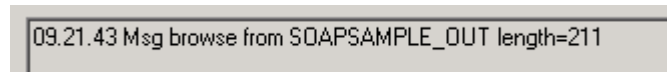
The **Properties** window is displayed.

- ___ g. Select **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 WebSphere MQ Explorer.
- ___ 5. Examine the SOAPSAMPLE_OUT queue with RFHUtil.
- ___ a. Start RFHUtil by clicking the **RFHUtil** icon on the Windows desktop.
- ___ b. On the **Main** tab, select **Queue Manager Name IB9QMGR**, and **Queue Name SOAPSAMPLE_OUT**.

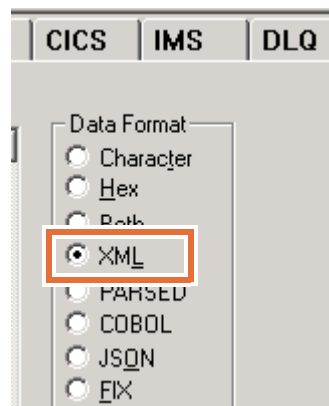
- ___ c. Click **Browse Q**.



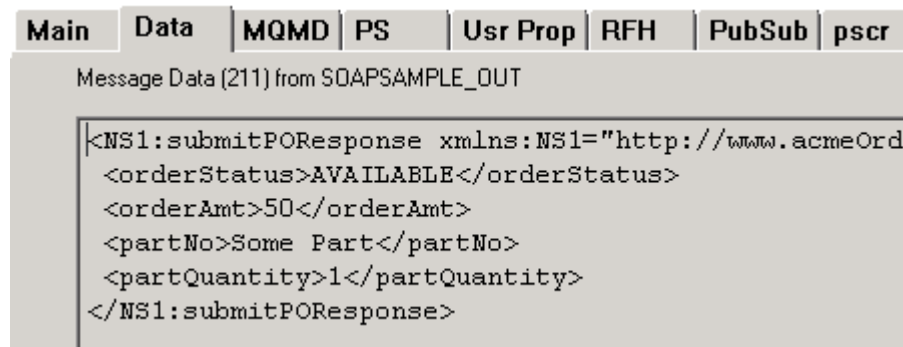
- ___ d. Confirm that the application read a message from the queue.



- ___ e. Click the **Data** tab at the top of the application.
- ___ f. In the **Data Format** section at the right, select **XML** to display the data in the XML format.



- ___ g. Inspect the contents of the message from the SOAPSAMPLE_OUT queue. The data represents the response from the SubmitPO web service operation.

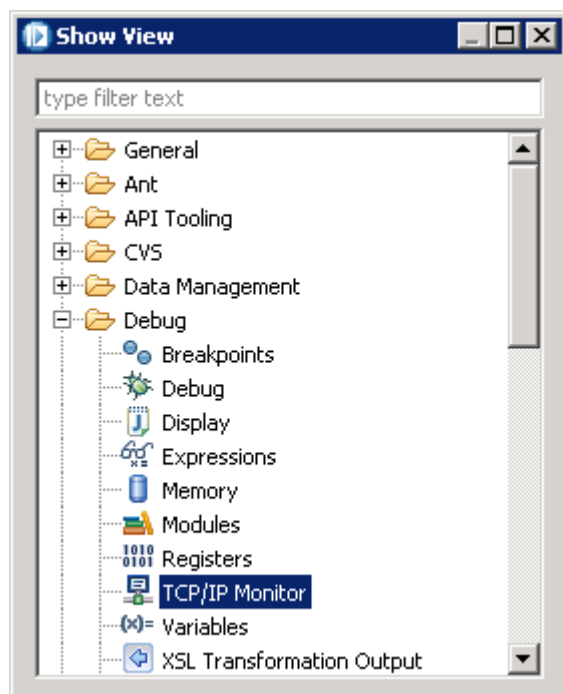


- ___ h. Close the RFHUtil window.
- ___ 6. You have finished following the instructions for the sample. You can close the Information Center window.

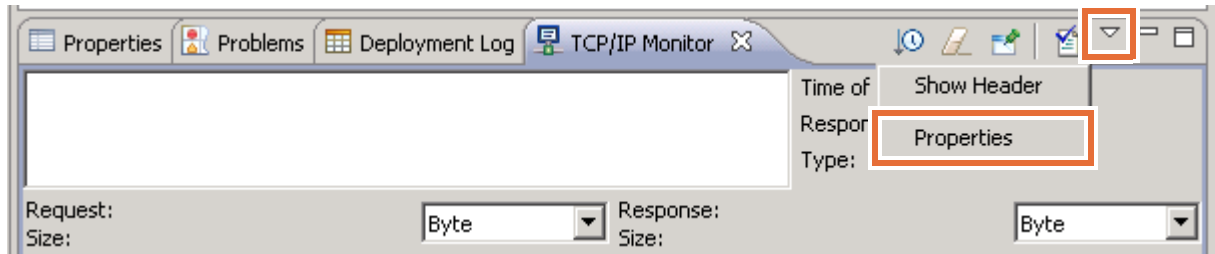
Part 5: Using the TCP/IP Monitor

In this part of the exercise, you extend the sample by using the TCP/IP Monitor to view message data as it passes through the SOAP/HTTP message transport. Specifically, you see the message that is sent from the SOAPRequest node to the SOAPInput node, and then the message that is returned from the SOAPReply node to the SOAPRequest node.

- ___ 1. From the IBM Integration Toolkit toolbar, click **Window > Show View > Other**. The **Show View** menu is displayed.
- ___ 2. Expand **Debug**, click **TCP/IP Monitor**, and then click **OK**.

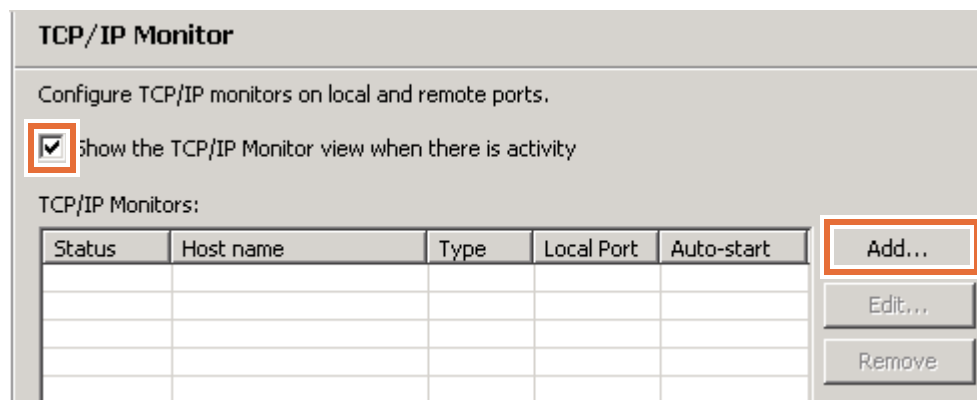


- ___ i. Confirm that Integration Toolkit opens the TCP/IP Monitor view.
- ___ 3. Configure the TCP/IP Monitor settings.
 - ___ a. In the **TCP/IP Monitor** view, click **View menu**. The button is a downward-pointing triangle in the view toolbar.
 - ___ b. Click **Properties**.



The **Preferences** dialog box is displayed.

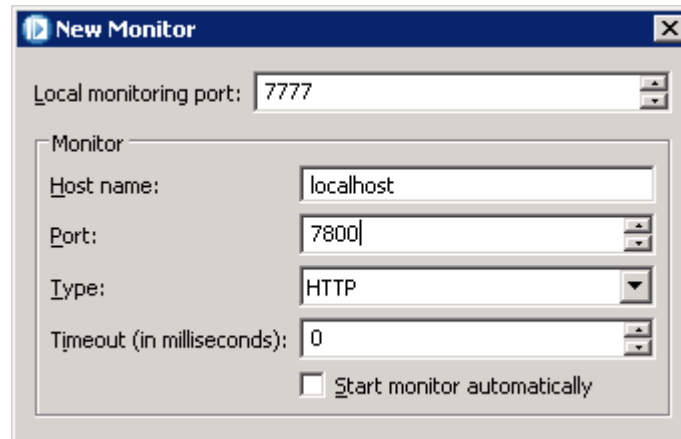
- ___ c. In the **Preferences** dialog box, verify that **Show TCP/IP Monitor view when there is activity** is selected.
- ___ d. Click **Add**.



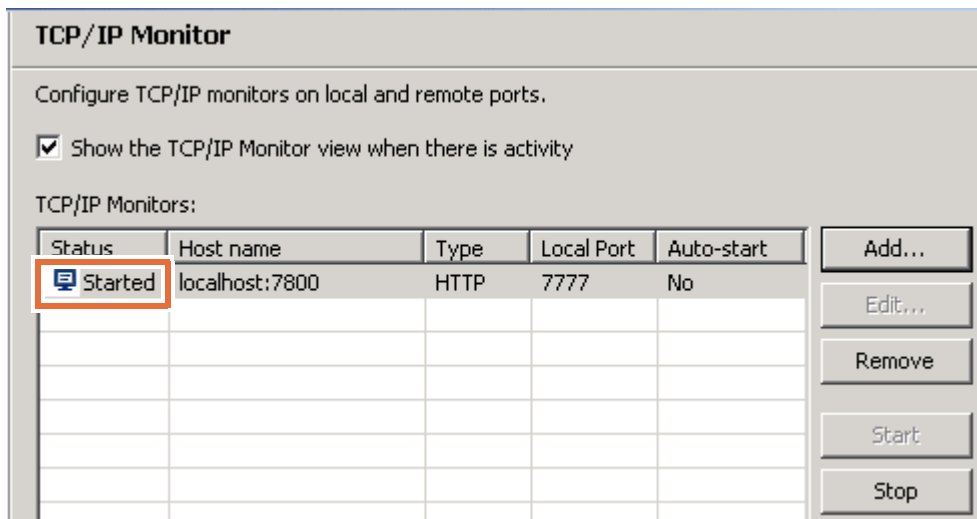
The **New Monitor** dialog box is displayed.

- ___ e. Set the following parameters:
 - **Local monitoring port:** 7777
 - **Host name:** localhost

- **Port:** 7800



- ___ 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.
- ___ 4. Rerun the application test.
 - ___ a. Switch to the Test Client window.
 - ___ b. In the **Message Flow Test Events** window, click the **Enqueue** operation to highlight it.
 - ___ c. 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.

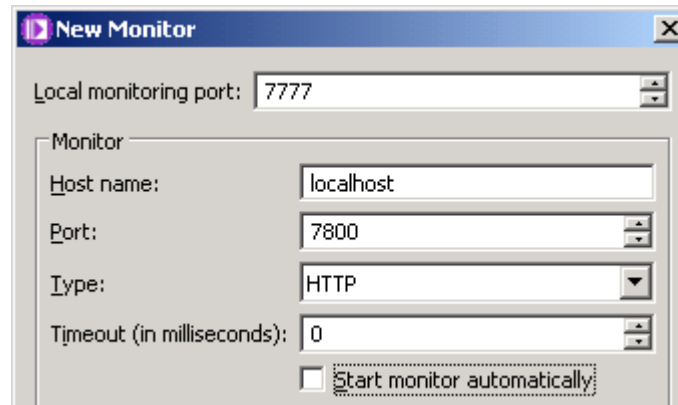


Questions

Why do you not see any activity on the TCP/IP Monitor?

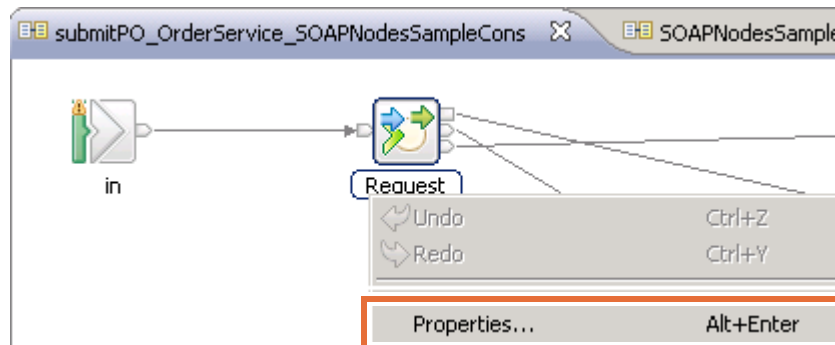
The SOAPRequest node in the message flow is not sending the request through the TCP/IP Monitor.

You configured the TCP/IP Monitor to use port 7777 as the local monitoring port.



You cannot set the local monitoring port to the same value as the port that is being monitored. Instead, you change the port that the SOAPRequest node uses so that it sends the message through the monitored port.

- ___ 5. Configure the SOAPRequest node to send messages to the monitored port.
 - ___ a. From the **Application Development** view, expand **Independent Resources > SOAPNodesSampleFlows > Flows > (default broker schema)**.
 - ___ b. Open **SOAPNodesSampleConsumerFlow.msgflow**.
 - ___ c. In the message flow editor, double-click the **Invoke_submitPO** subflow node.
 - ___ d. In the **SubmitPO** message flow, right-click the SOAPRequest node **Request**.
 - ___ e. Click **Properties**.

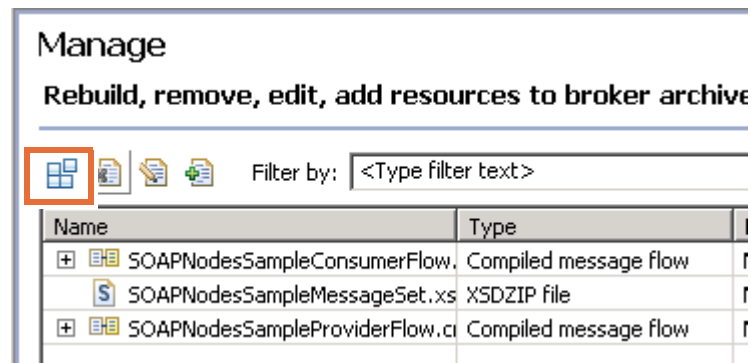


- ___ f. Click the **HTTP Transport** tab.

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

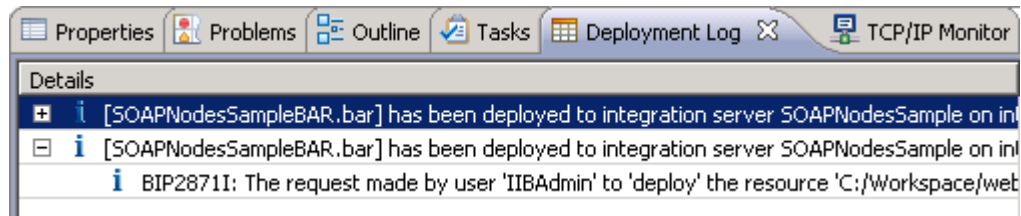
Web service URL*

- ___ h. Save all unsaved changes in the workspace by typing Ctrl + Shift + S.
- ___ 6. Rebuild and redeploy the BAR file.
- ___ a. In the **Application Development** view, expand **BARs**.
- ___ b. Double-click **SOAPNodesSampleBAR.bar** to open it in the BAR File editor.
- ___ c. Click the **Build** icon in the BAR File editor toolbar (the first icon in the toolbar).



- ___ d. If you receive messages about modified resources, click **Yes** to save them.
- ___ e. 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**.
- ___ 7. Deploy the BAR file to the SOAPNodesSample integration server.
- ___ a. In the **Application Development** view, right-click the **SOAPNodesSampleBAR.bar** file.
- ___ b. Click **Deploy**.
- ___ c. In the Deploy dialog box, click **SOAPNodesSample** as the integration server to deploy.
- ___ d. Click **Finish**.

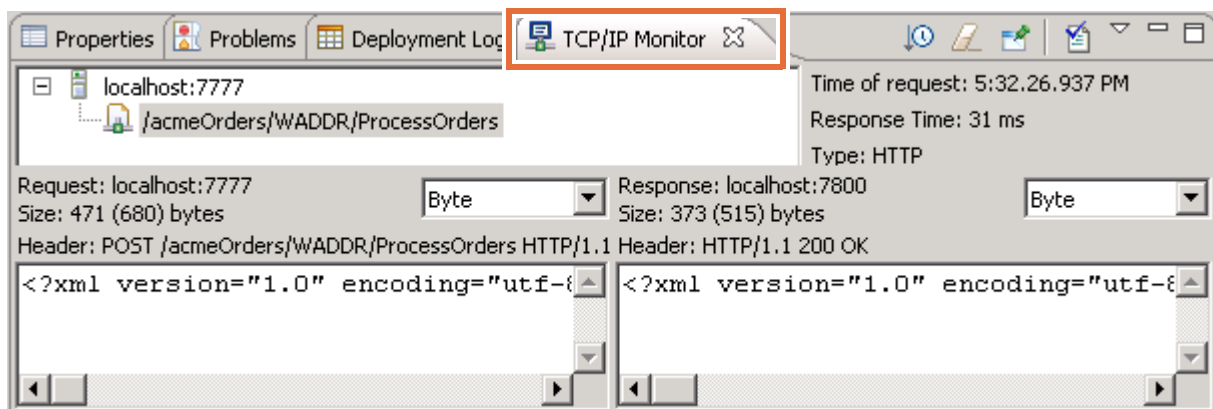
- ___ e. Wait for the deployment operation to complete. Verify that the deployment was successful by reviewing the **Deployment** tab for a success message for the current date and time.



- ___ 8. Run the application test.
- ___ a. Switch to the Test Client window.
- ___ b. In the Message Flow Test Events window, click the **Enqueue** operation to highlight it.
- ___ c. 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.

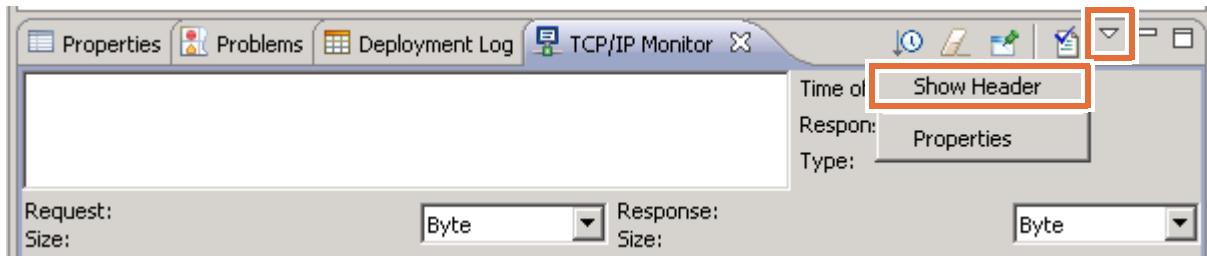
- ___ 9. Review the web service request and response messages in the TCP/IP Monitor.
- ___ a. Click the **TCP/IP Monitor** tab.



In the top part of the monitor, you see the monitored port (7777) and the URL that was 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.

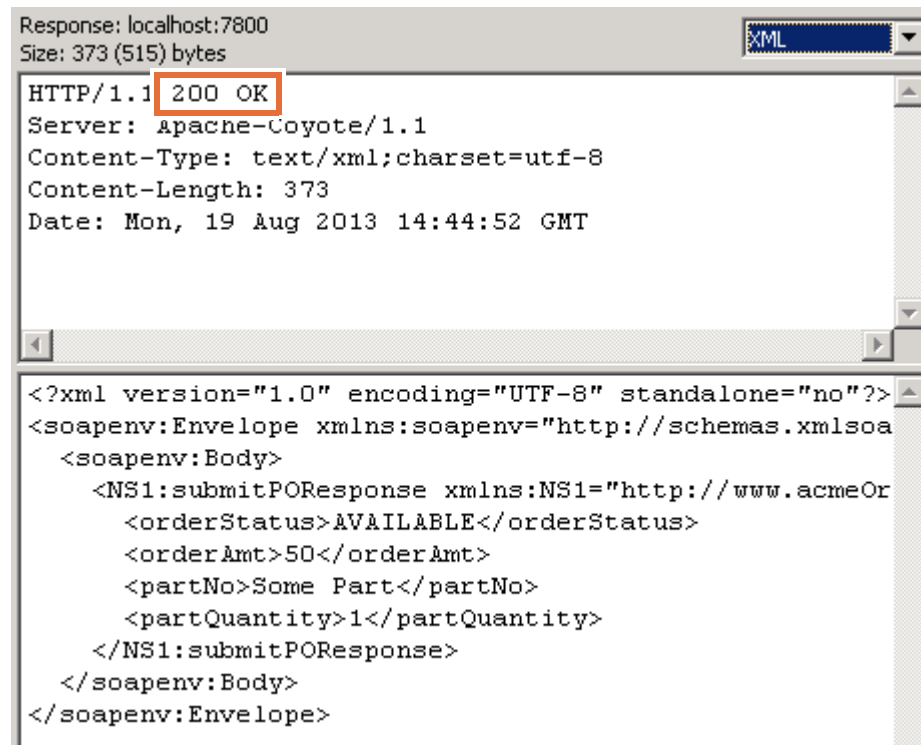
```
Request: localhost:7777
Size: 471 (680) bytes
XML

POST /acmeOrders/WADDR/ProcessOrders HTTP/1.1
Content-Length: 471
Content-Type: text/xml; charset=utf-8
Host: localhost:7800
SOAPAction: "http://www.acmeOrders.com/OrderService"
Connection: keep-alive

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <NS1:submitPORequest xmlns:NS1="http://www.acmeOrders.com/OrderService">
      <partNo>Some Part</partNo>
      <partQuantity>1</partQuantity>
      <personName>
        <firstName>Message</firstName>
        <lastName>Broker</lastName>
      </personName>
      <address>
        <street>IBM</street>
        <city>IBM</city>
        <zipCode>IBM</zipCode>
      </address>
    </NS1:submitPORequest>
  </soapenv:Body>
</soapenv:Envelope>
```

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.

- ___ 10. Restore the **TCP/IP Monitor** pane to its normal size by double-clicking the tab.

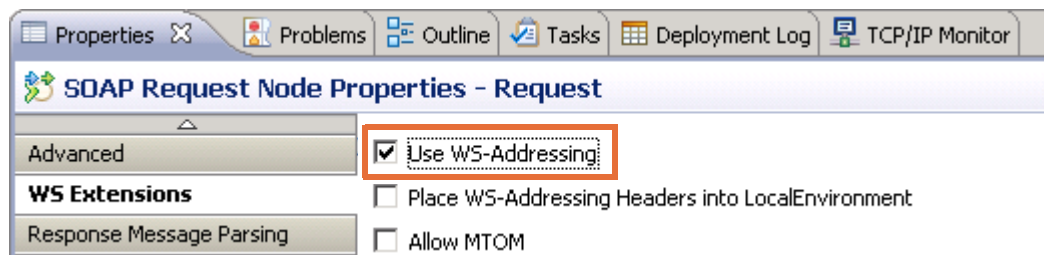
Part 6: Set the reply message path with WS-Addressing

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

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

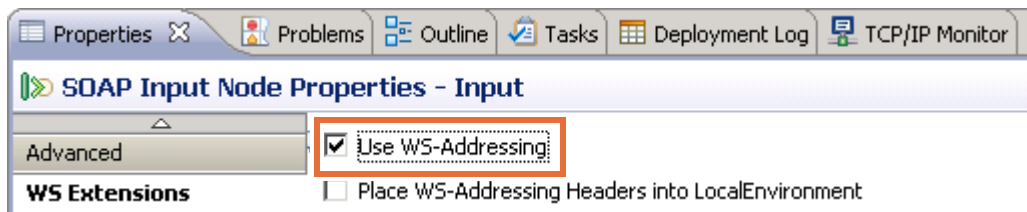
- ___ 1. Configure the web service consumer flow.
- ___ a. If the **Invoke_submitPO** subflow for the message consumer is not already open in the editor, expand **Independent Resources > SOAPNodesSampleFlows > Flows > (default broker schema)**.
 - ___ b. Double-click **SOAPNodesSampleConsumerFlow.msgflow** to open the main message flow, and then double-click the **Invoke_submitPO** subflow.
 - ___ c. Right-click the SOAPRequest node **Request**, and then click **Properties**.
 - ___ d. Click the **WS Extensions** tab.

- ___ e. Select **Use WS-Addressing**.



- ___ 2. Configure the web service provider flow.

- ___ a. If the **SOAPNodesSampleProviderFlow** for the message provider is not already open in the editor, expand **Independent Resources > SOAPNodesSampleFlows > Flows > (default broker schema)**, and then double-click **SOAPNodesSampleProviderFlow.msgflow** to open the message flow.
- ___ b. Right-click the SOAPInput node **Input**, and then click **Properties**.
- ___ c. Click the **WS Extensions** tab.
- ___ d. Select **Use WS-Addressing**.



- ___ 3. Save all unsaved changes in the workspace by typing Ctrl + Shift + S.
- ___ 4. Rebuild and redeploy the BAR file.
- ___ a. In the **Application Development** view, expand **BARs**.
- ___ b. Double-click **SOAPNodesSampleBAR.bar** to open it in the BAR File editor.
- ___ c. Click the **Build** icon in the BAR File editor toolbar (the first icon in the toolbar).
- ___ d. If you receive messages about modified resources, click **Yes** to save them.
- ___ e. If you receive a message about overriding configurable properties, click **OK** to allow the override to occur.
The BAR file is rebuilt. This operation can take a few moments.
- ___ f. When the **Operation completed successfully** message is displayed, click **OK**.
- ___ g. Redeploy the BAR file to the SOAPNodesSample integration server.
- ___ h. 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.

- ___ a. Switch to the Test Client window.
- ___ b. In the Message Flow Test Events window, click the **Enqueue** operation to highlight it.
- ___ c. 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.

___ 6. 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.

```

Request: localhost:7777
Size: 856 (1027) bytes
XML

POST /acmeOrders/WADDR/ProcessOrders HTTP/1.1
Content-Length: 856
Content-Type: text/xml; charset=utf-8
Host: localhost:7800
SOAPAction: ""
Connection: Keep-Alive

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:To>http://localhost:7777/acmeOrders/WADDR/ProcessOrders</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>urn:uuid:23E44846DF7DFE5AF11376924153767</wsa:MessageID>
    <wsa:Action>http://www.acmeOrders.com/OrderService</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <NS1:submitPORequest xmlns:NS1="http://www.acmeOrders.com/OrderService">
      <partNo>Some Part</partNo>
      <partQuantity>1</partQuantity>
      <personName>
        <firstName>Message</firstName>
      </personName>
    </NS1:submitPORequest>
  </soapenv:Body>
</soapenv:Envelope>

```

There are a number of differences between this message and the message that was sent before WS-Addressing was enabled:

- **SOAPAction** value is now null.
- A number of WS-Addressing fields are listed in the **SOAPenv:Header** section of the message. These fields are prefixed by **wsa**.
- 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.

```

Response: localhost:7800
Size: 920 (1062) bytes
XML

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 920
Date: Mon, 19 Aug 2013 14:55:53 GMT

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous</wsa:Add
    </wsa:ReplyTo>
    <wsa:MessageID>urn:uuid:319fefd0-f839-4fa9-839f-683828f5a98d</wsa:Mess
    <wsa:Action>http://www.acmeOrders.com/OrderService/OrderService/submit
    <wsa:RelatesTo RelationshipType="http://www.w3.org/2005/08/addressing,
  </soapenv:Header>
  <soapenv:Body>
    <NS1:submitPOResponse xmlns:NS1="http://www.acmeOrders.com/OrderServic
      <orderStatus>AVAILABLE</orderStatus>
  </soapenv:Body>
</soapenv:Envelope>
  
```

- 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 WS-Addressing **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.

Part 7: Clean up the workspace

- ___ 1. Close the TCP/IP Monitor tab.
- ___ 2. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, if you are prompted.
- ___ 3. In the **Integration Nodes** view, right-click the **SOAPNodesSample** integration server, and then click **Delete > Integration server**. When the **Confirm Deletion** dialog box is displayed, click **OK**.

This action deletes the integration server that the imported sample created and all of its contents. (If the Information Center is still open, you can use the link within the SOAP Nodes sample instructions to remove the deployed objects instead, if you want.)

End of exercise

Exercise 5. Creating and testing an integration service

What this exercise is about

In this exercise, you design, implement, and test an integration service.

What you should be able to do

At the end of this exercise, you should be able to:

- Create a container for an integration service
- Design the operations, bindings, and messages for a web service interface in a graphical editor
- Implement a web service operation with a database SQL call
- Test a web service with the IBM Integration Toolkit
- Review service invocation statistics with the IBM Integration Explorer

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. Review the integration service with the Integration Explorer.

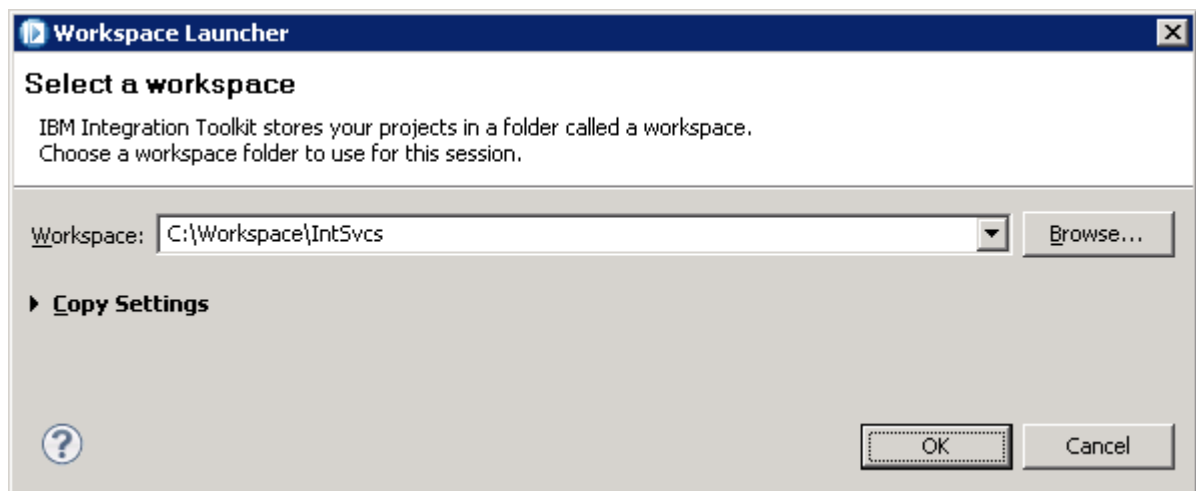
Requirements

- A workstation with WebSphere MQ and the IBM Integration Bus components installed
- The IBM Integration Bus default configuration
- IBM DB2 and SAMPLE database
- IBM WebSphere MQ SupportPac IH03, RfhUtil
- Lab files in the C:\Labs\Lab05-IntSvcs directory

Exercise instructions

Part 1: Open a new workspace

- ___ 1. Make sure that all local IBM Integration Bus components are running.
- ___ 2. Start the IBM Integration Toolkit if it is not already running.
 - ___ a. From the desktop, open the **Integration Toolkit 9.0.0.0** shortcut.
- ___ 3. Open a new workspace.
 - ___ a. In the Workspace Launcher dialog box, enter **C:\Workspaces\IntSvc**.

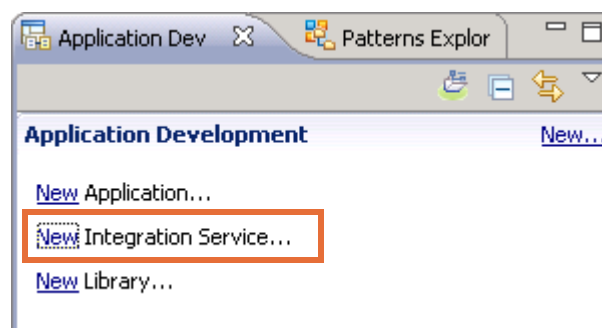


- ___ b. Click **OK**.
- ___ 4. Close the Welcome view to go to the **Integration Development** perspective.

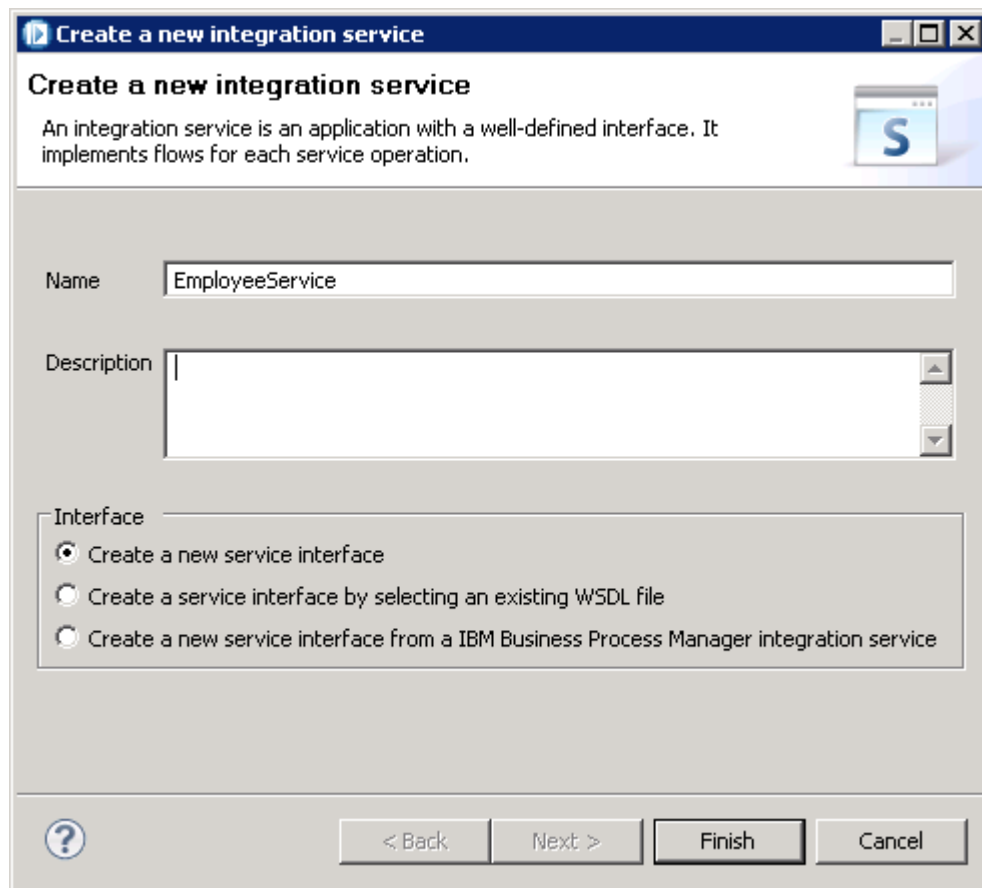
Part 2: Create an integration service

Before you design your web service, you must create an integration service to hold the components of the web service.

- ___ 1. Create an integration service named **EmployeeService**.
 - ___ a. In the Application Development view, click **New Integration Service**.



- ___ b. In the wizard, enter **EmployeeService** as the integration service name.
- ___ c. Select **Create a new service interface**.



- ___ d. Click **Finish**.
- ___ 2. Examine the integration service project structure.
- In the Application Development view, expand **Employee Services** components.



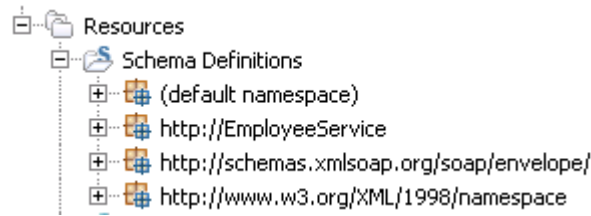
Information

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

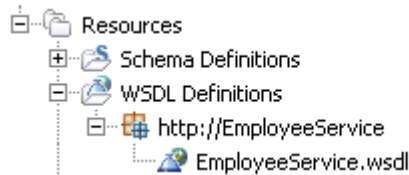


- **Integration Service Description** is a shortcut to the WSDL document that defines the service interface.

- **Resources > Schema Definitions** collects all of the XML schema definitions that the WSDL document uses.



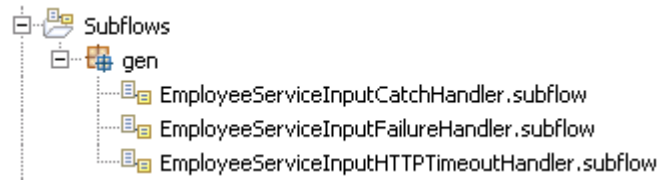
- XML elements that are not explicitly linked to an XML namespace belong to the **default namespace**.
 - **http://EmployeeService** is the namespace for the Employee Service WSDL document. You can change the namespace declaration in the WSDL document.
 - **http://schemas.xmlsoap.org/soap/envelope/** 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** contain the message flows that implement web service operations. 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:



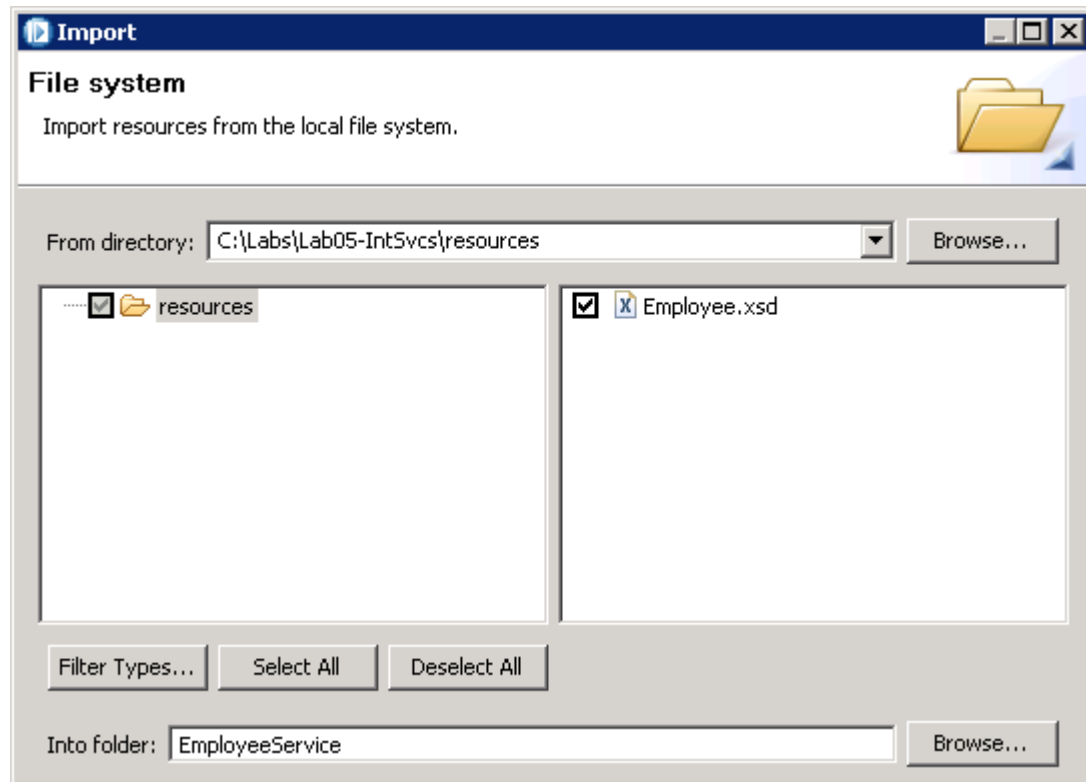
- A Catch error handler
- A Failure error handler
- An HTTP timeout error handler

Part 3: Define the service interface

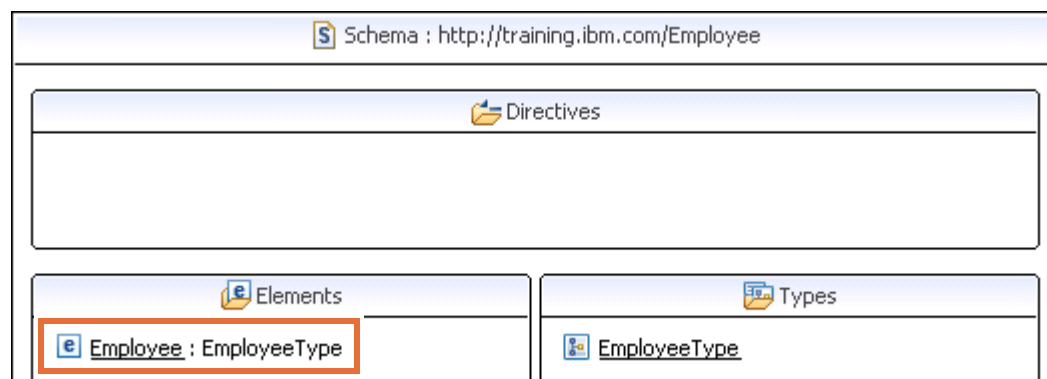
Start your design with a top-down approach and design a service interface first. Import the `Employee.xsd` XML schema document to use the data types in your service interface.

- ___ 1. Import the Employee data XML schema file.
 - ___ a. From the main menu, click **File > Import**.
 - ___ b. In the Import wizard, click **General > File System**.
 - ___ c. In the File System import dialog box, enter `C:\Labs\Lab05-IntSvcs\resources` in the **From Directory** field.
 - ___ d. Click **Employee.xsd** from the resources directory.

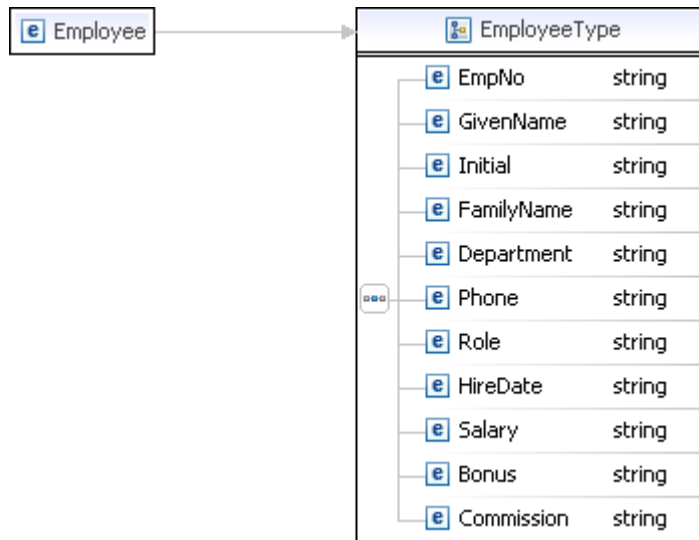
- ___ e. In the **Into folder** field, select **EmployeeService**.



- ___ f. Click **Finish**.
- ___ 2. Examine the Employee data XML schema.
- ___ a. From the Application Development view, expand **EmployeeServices > Resources > Schema Definition > http://training.ibm.com/Employee**.
- ___ b. Open the **Employee.xsd** file in an 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.

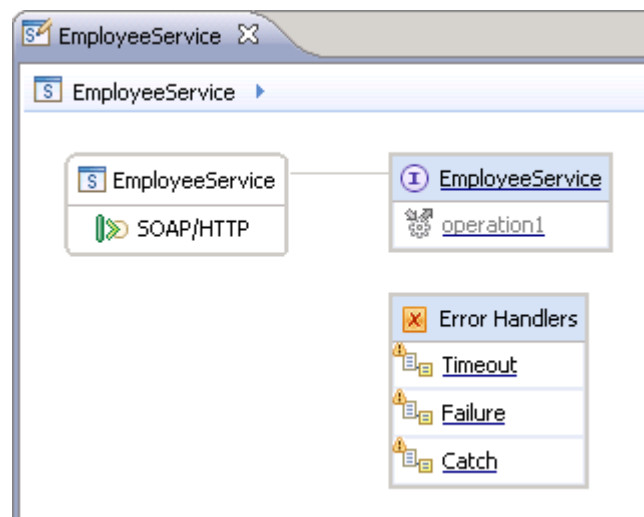
In this exercise, you design and implement a web service operation that retrieves an employee record from a database. The web service response message returns the record fields as an **<Employee>** element.

- ___ e. Close the XML schema editor.
- ___ 3. Examine the Integration Service Description.
- ___ a. From the Application Development view, double-click **EmployeeService > Integrated Service Description**.
- ___ b. Examine the **EmployeeService** service diagram on the **Service** tab.



Information

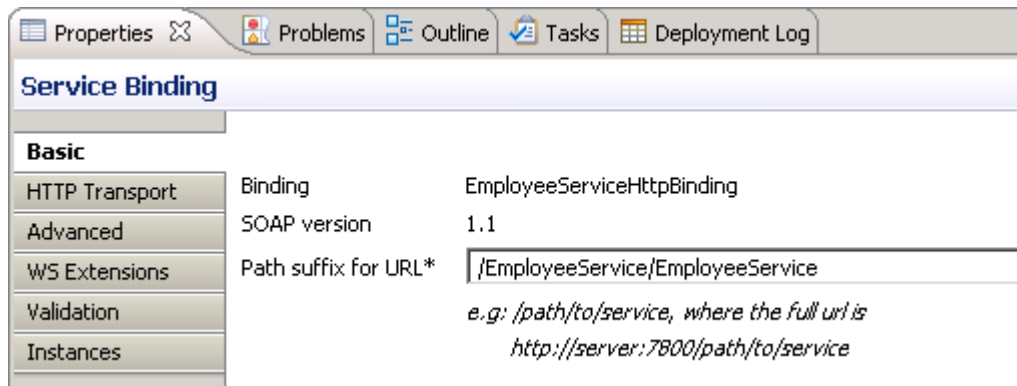
The EmployeeService integrated service description has two parts: a visual diagram of the service implementation on the **Service** tab, and a service interface on the **Interface** tab.



- 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.

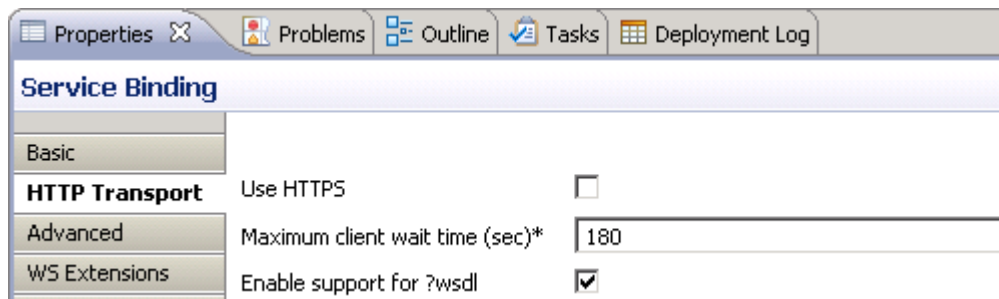
___ c. Select the **SOAP/HTTP** service binding.

- ___ d. In the **Properties** view, click **Basic**.



The integration service binding supports SOAP specification version 1.1. 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/EmployeeService**.

- ___ e. Click **HTTP Transport** in the Properties view.



- To enable SSL/TLS security for the web service endpoint, select **Use HTTPS**.
- The default 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/EmployeeService?wsdl**.

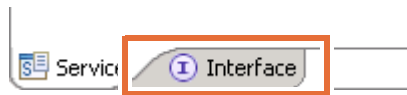
- ___ 4. Create a web service operation, **getEmployee**, that accepts an employee XML element and returns an employee XML element in the reply message.

**Note**

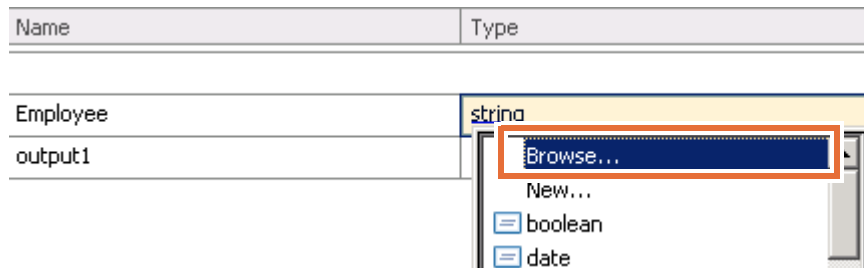
For this exercise, the `getEmployee` web service operation reads the `EmpNo` field only. However, future versions of the `getEmployee` operation can search the `EMPLOYEE` database table with other fields.

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.

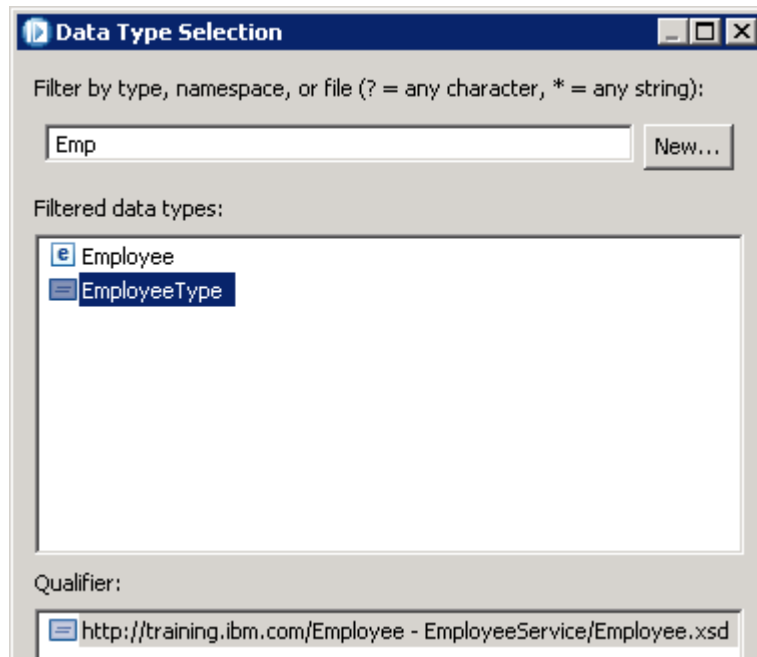


- ___ b. In the **Operations** section, double-click **operation1**.
- ___ c. Change the name of the operation to `getEmployee`.
The name of the operation input and output messages changes to `getEmployee` and `getEmployeeResponse`.
- ___ d. In the **getEmployee** input field, enter `Employee` as the input message name.
- ___ e. Select the **string** type in the input field.



- ___ f. Click **Browse**.
- ___ g. In the Data Type Selection dialog box, enter `emp` as the filter by type setting.

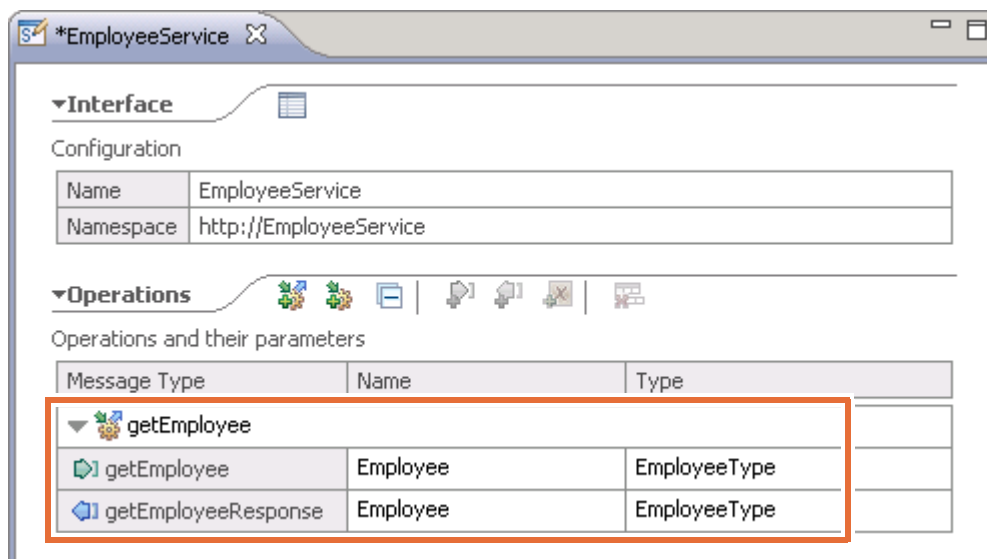
- ___ h. Click **EmployeeType** as the data type for the input message.
Click **OK**.



Note

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

- ___ i. Change the getEmployeeResponse message name to **Employee**.
___ j. Set the getEmployeeResponse message type to **EmployeeType**.

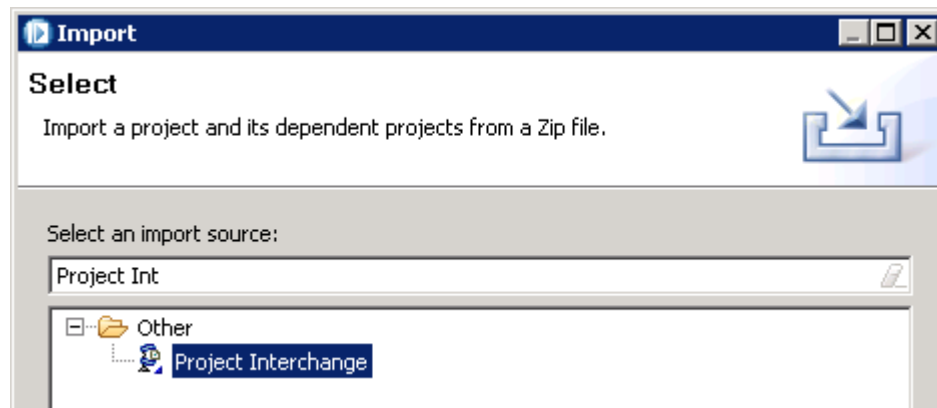


- ___ k. Save the **EmployeeService** interface.

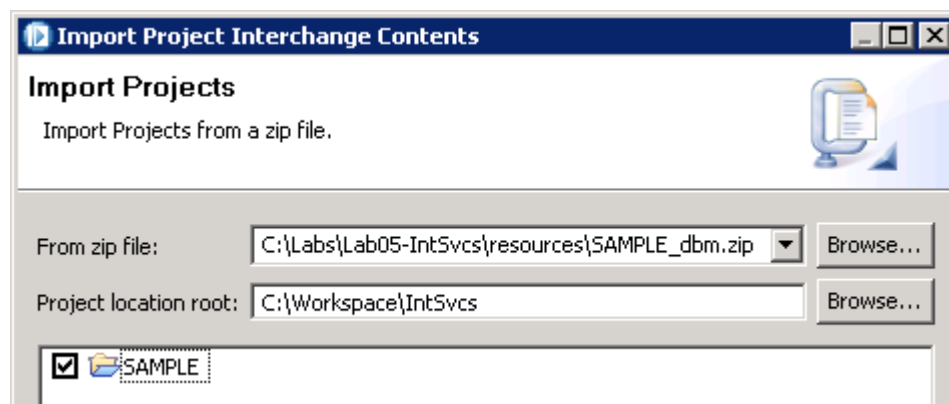
Part 4: Implement the EmployeeService message flow

In this section, provide an implementation for the service interface that you defined. Complete the EmployeeService message flow to accept requests for the `getEmployee` web service operation. Use a mapping node to retrieve an employee record from the database and write the results in the `getEmployeeResponse` web service output message.

- ___ 1. Import a database definition file for the SAMPLE database.
- ___ a. From the main menu, click **File > Import**.
- ___ b. Click **Other > Project Interchange** from the Import wizard.



- ___ c. Click **Next**.
- ___ d. In the Import Projects page, click **Browse** and go to **C:\Labs\Lab05-IntSvcs\resources**
- ___ e. Select **SAMPLE_dbm.zip** and then click **Open**.
- ___ f. Select the **SAMPLE** project.

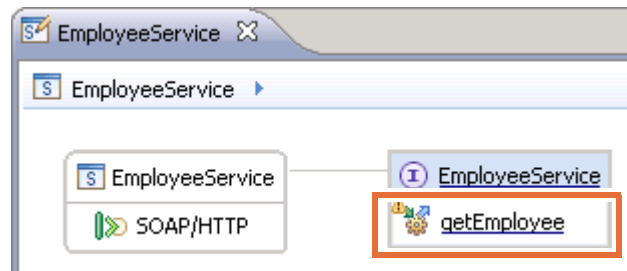


- ___ g. Click **Finish**.

- ___ h. Confirm that the Application Development view lists the **SAMPLE** database model file.



- ___ 2. Open the **getEmployee** web service operation message flow.
- ___ a. In the Application Development view, open the EmployeeService **Integration Service Description**.
- ___ b. Select the **getEmployee** web service operation.

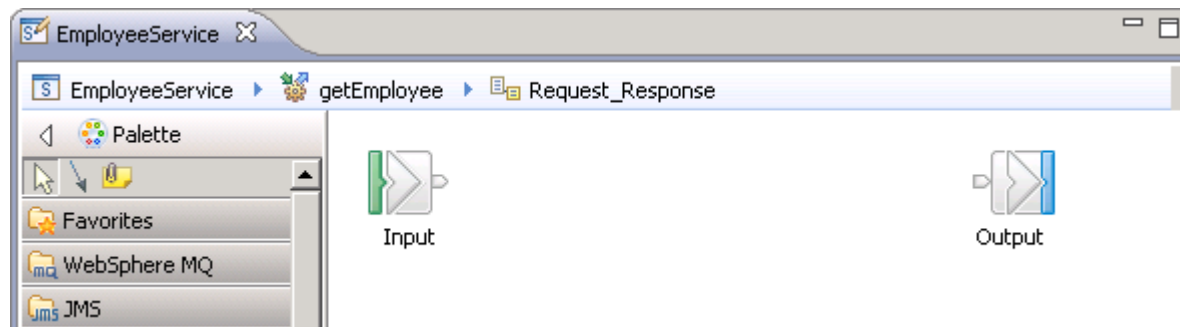


- ___ c. Examine the message flow for the **getEmployee** operation.



Information

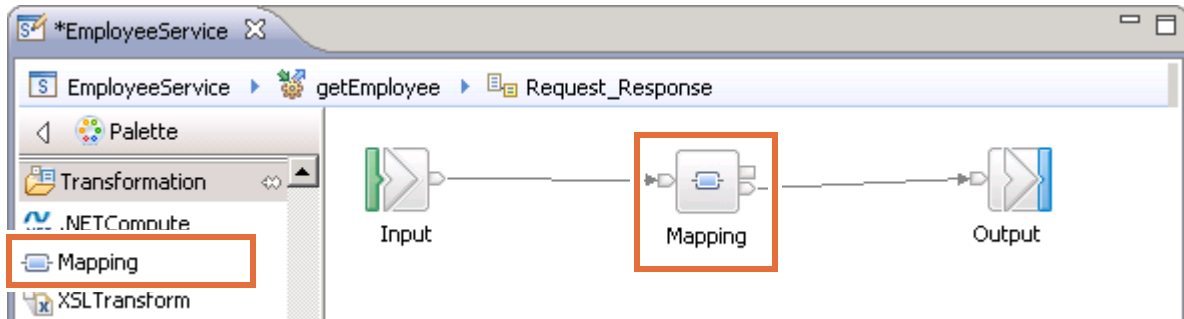
The **Request_Response** message flow implements the **getEmployee** service operation.



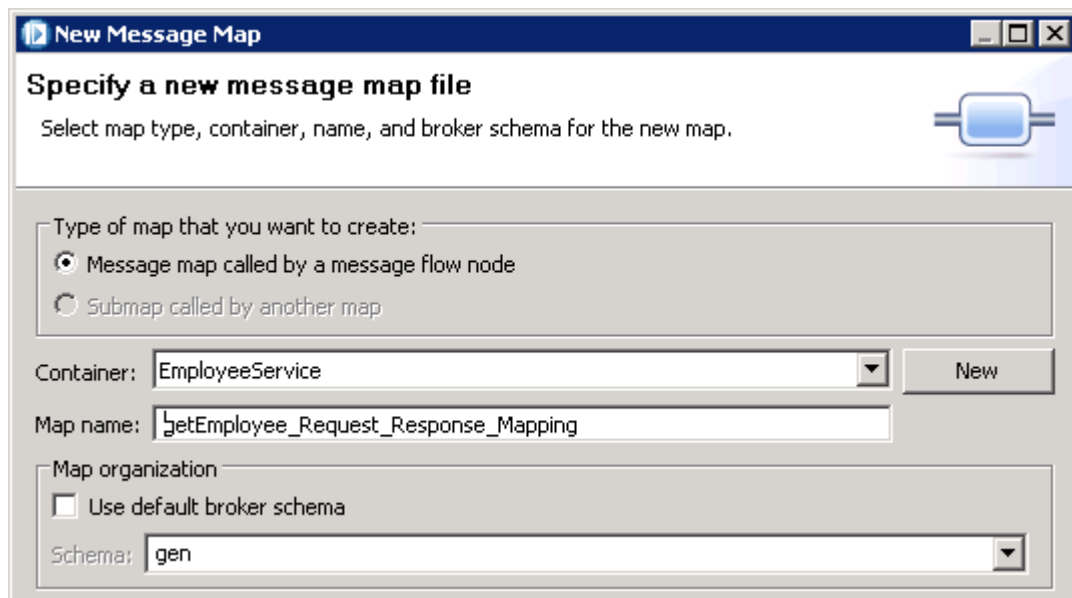
- The integration service defines a subflow for each service operation. In this case, the **getEmployee_Request_Response** subflow represents the **getEmployee** operation.
- The **input** node accepts the web service input message, **getEmployee**.
- The **output** node expects the web service output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

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

- ___ 3. Wire 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.

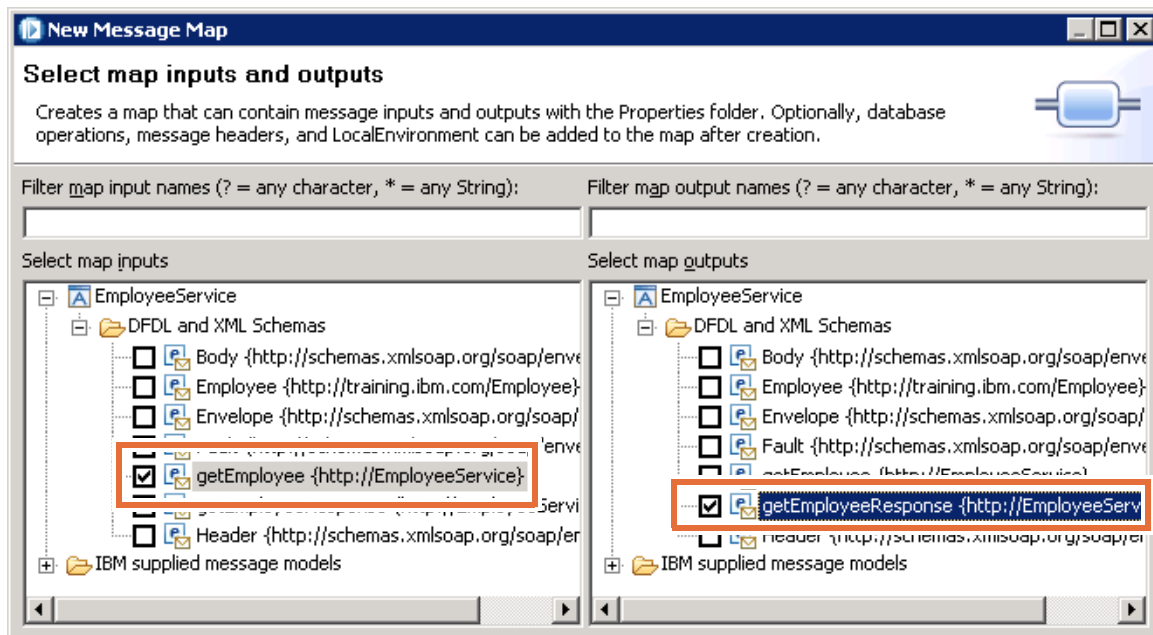


- ___ 4. Create a map between the `getEmployee` WSDL input message and the `getEmployeeResponse` WSDL output message.
 - ___ a. Double-click the Mapping node.
 - ___ b. In the New Message Map wizard, leave the message map name to the default values.



- ___ c. Click **Next**.
- ___ d. In the **Select map inputs and outputs** page, select the web service input message `getEmployee`.

- ___ e. Select **getEmployeeResponse** as the output message of the mapping node.

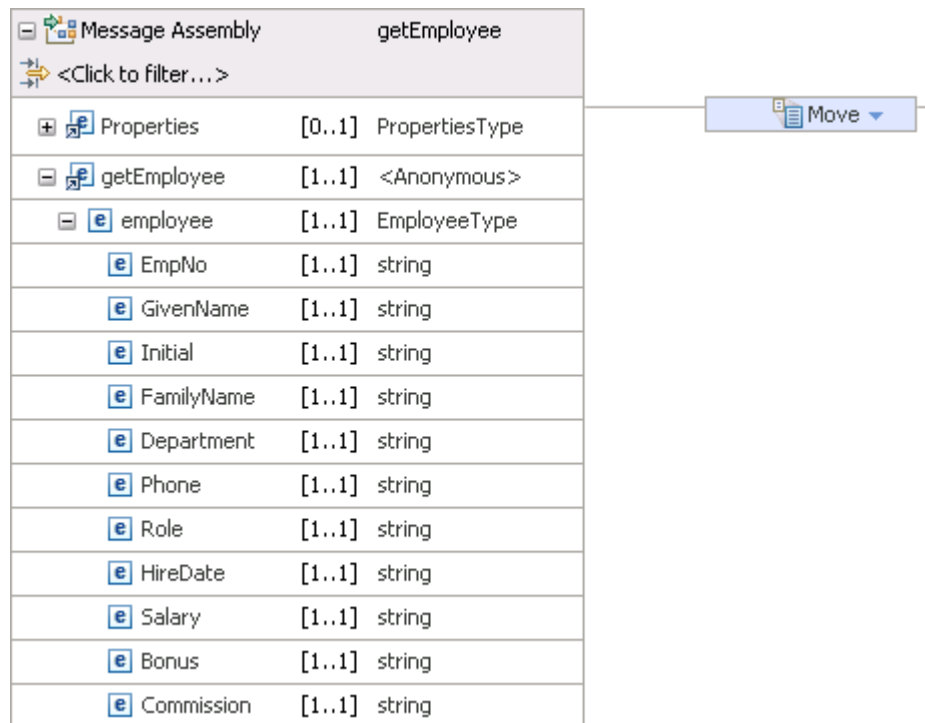


Note

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**.

- ___ 5. 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** WSDL messages.



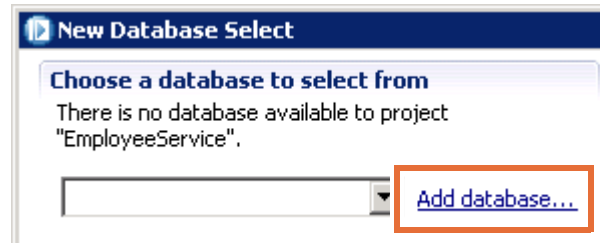
Information

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.

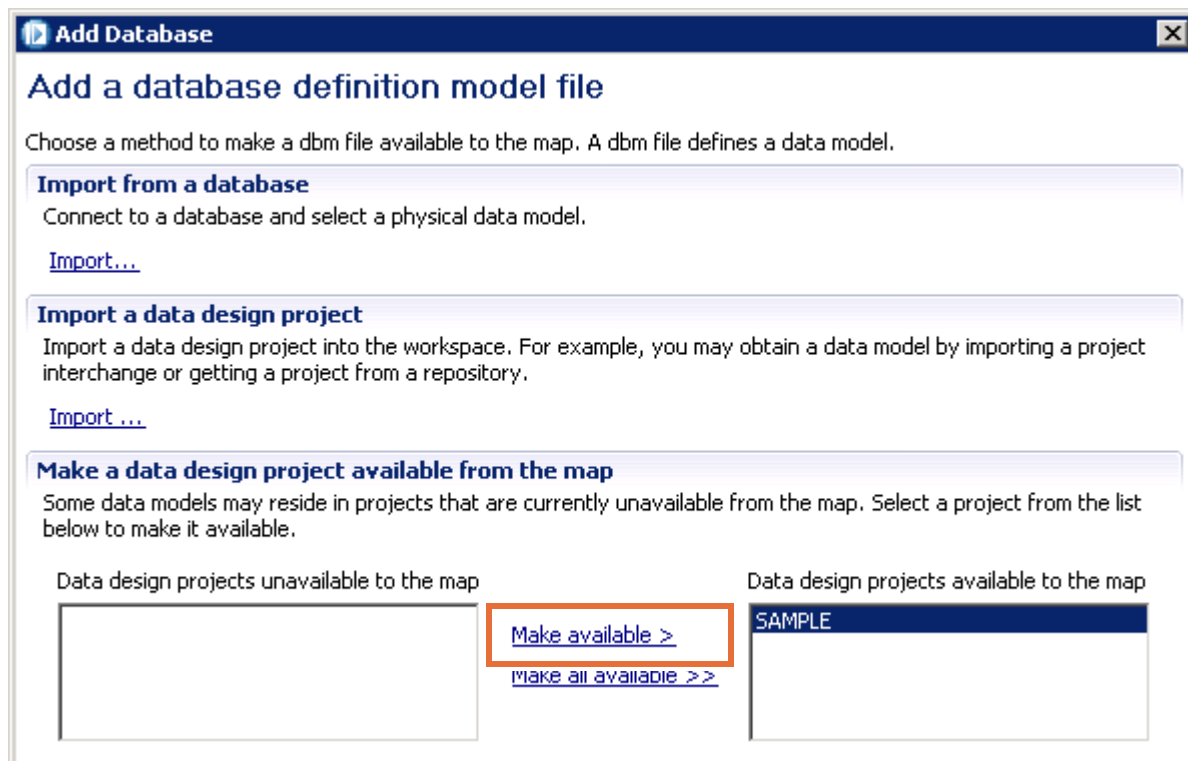
- ___ b. From the editor toolbar, click **Create rows from a database**.



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



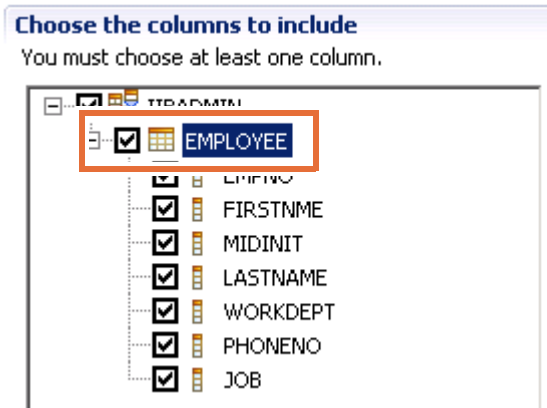
- ___ d. In the **Add a database definition file**, select the **SAMPLE** database definition file from the **Make a data design project available from the map**.
- ___ e. Click **Make available**.



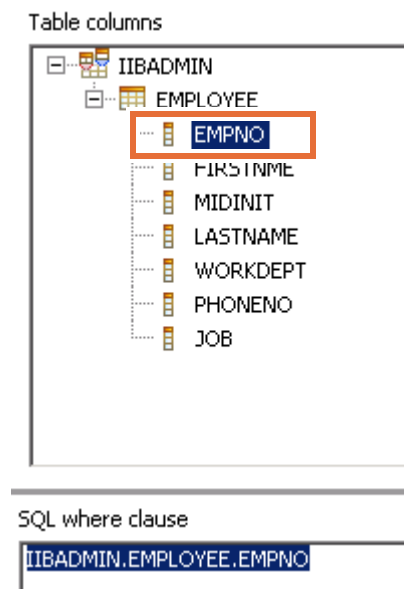
- ___ f. Confirm that the wizard selects the **SAMPLE** data design project.



- ___ 6. 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 **EmpNo** XML element in the **Available inputs for column values** column.

Define a where clause

The where clause is used to extract only those rows that fulfill a specified condition, which is often the value of a key column in the database table. The value can come from other inputs in the map. The expression must evaluate to a boolean.

The screenshot shows three panes in a dialog box:

- Table columns:** A tree view showing the database structure: IIBADMIN > EMPLOYEE > EMPNO (selected), FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, and JOB.
- Operators:** A list of logical and comparison operators: AND, OR, NOT, =, <>, >, <, >=, <=, BETWEEN, LIKE, and IN.
- Available inputs for column values:** A tree view showing message inputs: \$MessageAssembly > Properties > io:getEmployee > employee > io2:EmpNo (selected and highlighted with a red box). Other inputs like io2:Initial, io2:FamilyName, etc., are also visible.

SQL where clause

IIBADMIN.EMPLOYEE.EMPNO = ?

Plac...	XPath expression
?	veeService}:getEmployee/employee/{http://training.ibm.com/Employee}:EmpNo



Information

The New Database Select wizard creates the SQL statement:

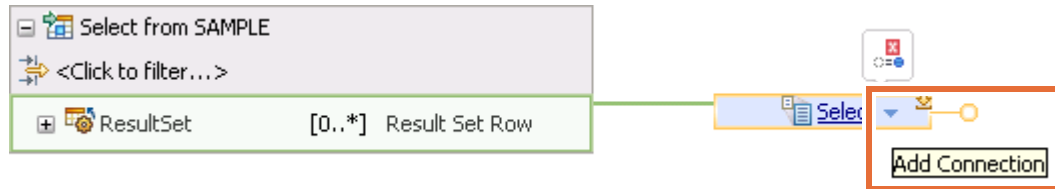
```
SELECT IIBADMIN.EMPLOYEE.FIRSTNAME, IIBADMIN.EMPLOYEE.MIDINIT,
IIBADMIN.EMPLOYEE.LASTNAME, IIBADMIN.EMPLOYEE.WORKDEPT,
IIBADMIN.EMPLOYEE.PHONENO, IIBADMIN.EMPLOYEE.JOB WHERE
IIBADMIN.EMPLOYEE.EMPNO = ?
```

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

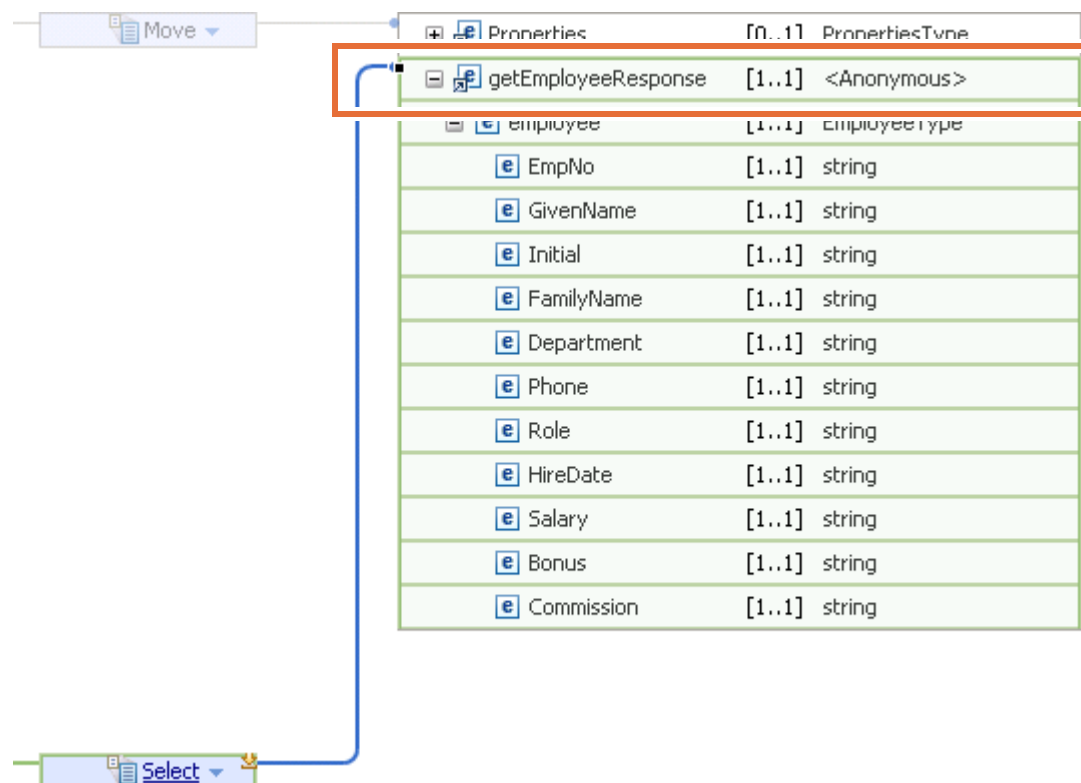
```
$MessageAssembly/{http://EmployeeService}:getEmployee/employee/
{http://training.ibm.com/Employee}:empNo/
```

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

- ___ f. Click **OK**.
- ___ 7. 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.



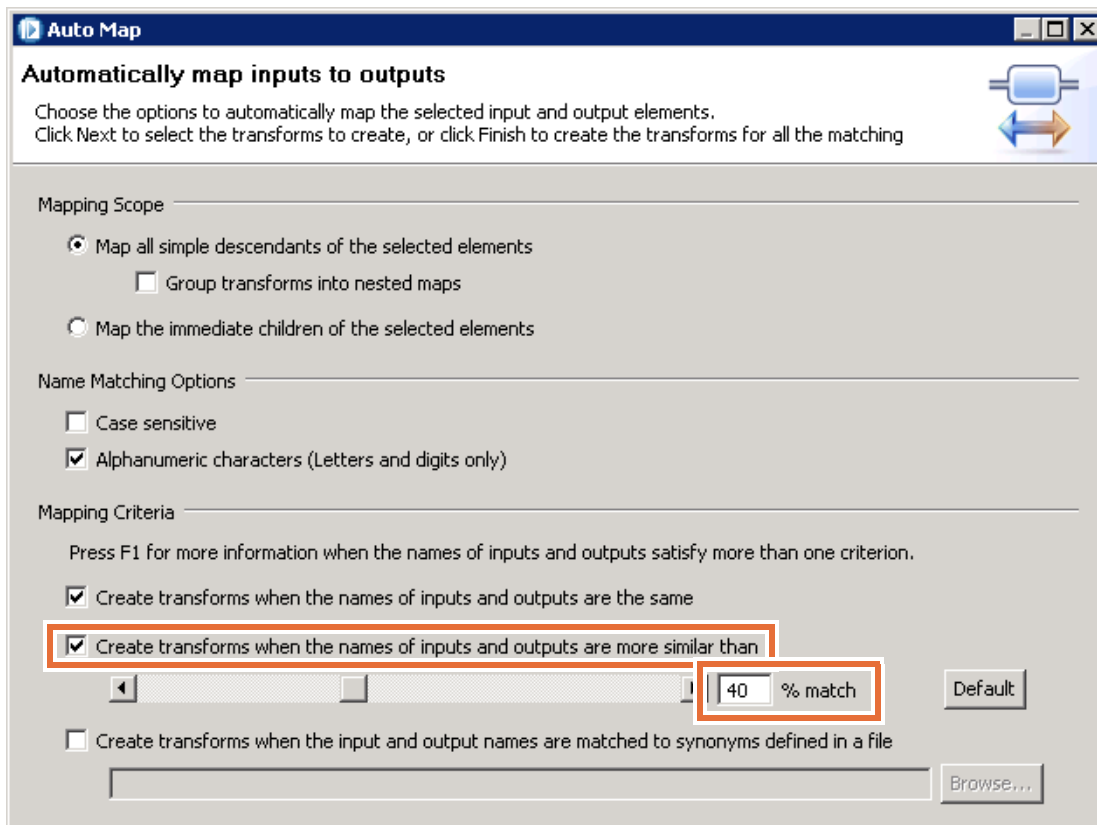
- ___ 8. Map the Connect from SAMPLE result set fields to the elements of Employee in the output message.
- ___ a. Click the **Select** action to open the nested message map.



- ___ b. Click the **Automap** function.

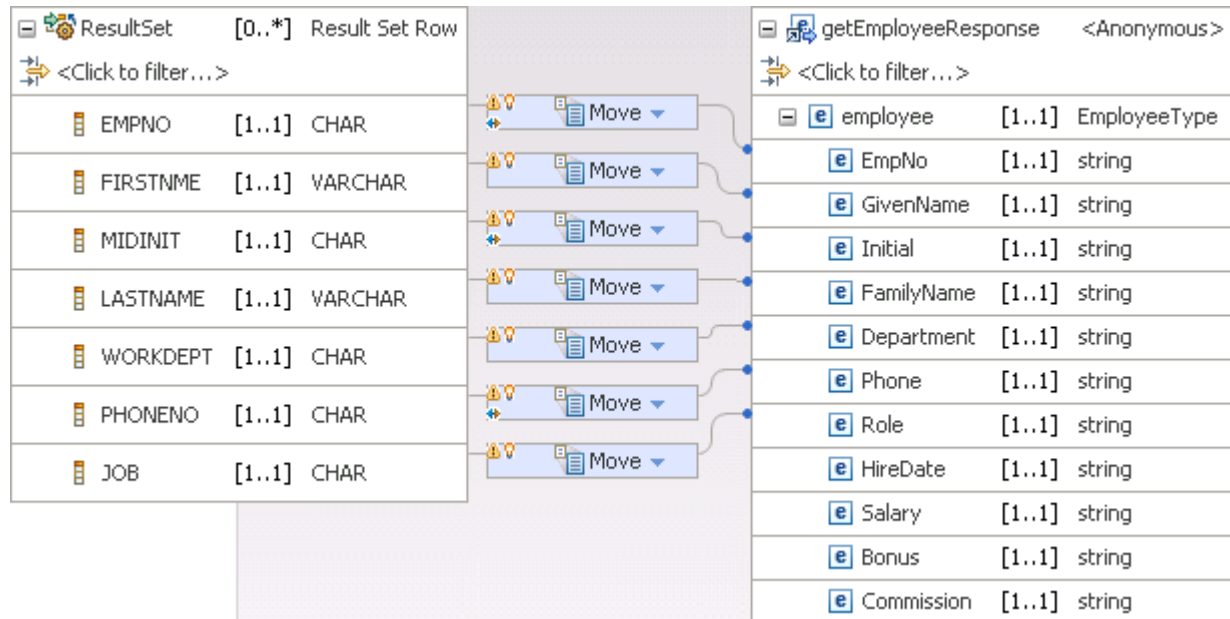


- ___ c. In the AutoMap wizard, select **Create transforms when the names of inputs and outputs are more similar than**.
- ___ d. Enter **40** as the percentage match factor.



- ___ e. Click **Finish**.
- ___ 9. Complete the mapping between the SQL Select result set and the `getEmployeeResponse` message.
- ___ a. In the mapping editor, draw a wire between **FIRSTNAME** and the **GivenName** XML element.
- ___ b. Change the wire from **LASTNAME** to **FamilyName**.
- ___ c. Draw a wire between **WORKDEPT** and **Department**.
- ___ d. Draw a wire between **JOB** and **Role**.

- ___ e. Confirm that all **getEmployeeResponse > Employee** child elements have mappings to the result set.

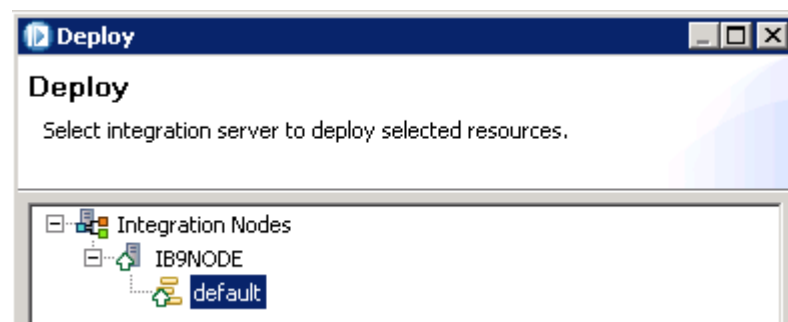


- ___ f. Save and close the mapping editor.
___ g. Save and close the message flow.

Part 5: Deploy the web service

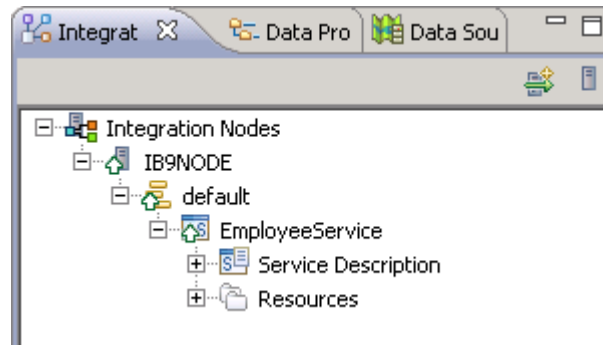
Deploy and review the integration service with the IBM Integration Explorer.

- ___ 1. Deploy the integration service.
- ___ a. In the Application Development view, right-click the **EmployeeService** project.
 - ___ b. Click **Deploy**.
 - ___ c. In the Deploy wizard, click **IB9NODE > default** as the integration server to deploy.



- ___ d. Click **Finish**.

- ___ e. Confirm that the EmployeeService integration service is deployed and running in the **default** integration server.



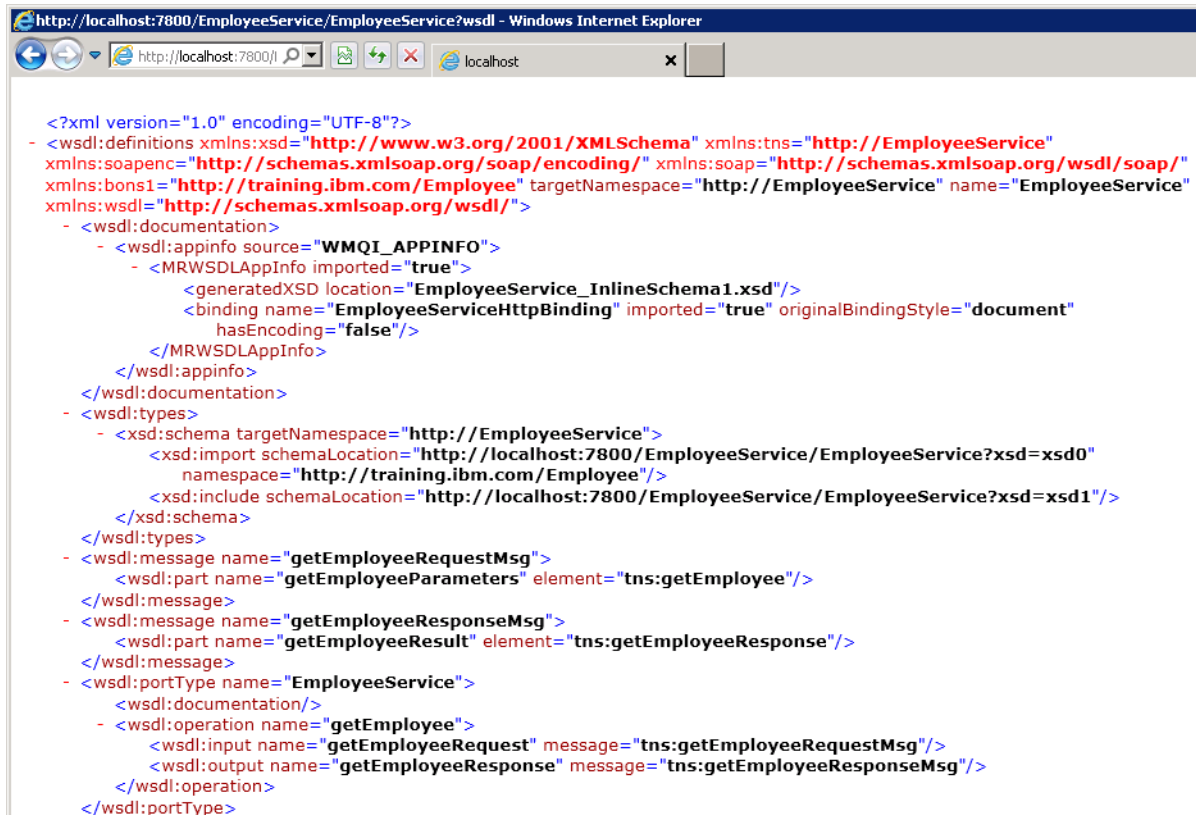
- ___ 2. Retrieve the WSDL document that describes the service endpoint.
- ___ a. Open a web browser.
- ___ b. Enter `http://localhost:7800/EmployeeService/EmployeeService?wsdl` in the address bar.
- ___ c. Press **Enter**.



Questions

How do you change the web service address from IBM Integration Toolkit?

- ___ d. Confirm that the EmployeeService WSDL document appears in the browser.



```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://EmployeeService"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:bons1="http://training.ibm.com/Employee" targetNamespace="http://EmployeeService" name="EmployeeService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation>
    <wsdl:appinfo source="WMQI_APPINFO">
      <MRWSDLAppInfo imported="true">
        <generatedXSD location="EmployeeService_InlineSchema1.xsd">
          <binding name="EmployeeServiceHttpBinding" imported="true" originalBindingStyle="document"
            hasEncoding="false">
          </MRWSDLAppInfo>
        </wsdl:appinfo>
      </wsdl:documentation>
    <wsdl:types>
      <xsd:schema targetNamespace="http://EmployeeService">
        <xsd:import schemaLocation="http://localhost:7800/EmployeeService/EmployeeService?xsd=xsd0"
          namespace="http://training.ibm.com/Employee">
        <xsd:include schemaLocation="http://localhost:7800/EmployeeService/EmployeeService?xsd=xsd1">
        </xsd:schema>
      </wsdl:types>
      <wsdl:message name="getEmployeeRequestMsg">
        <wsdl:part name="getEmployeeParameters" element="tns:getEmployee"/>
      </wsdl:message>
      <wsdl:message name="getEmployeeResponseMsg">
        <wsdl:part name="getEmployeeResult" element="tns:getEmployeeResponse"/>
      </wsdl:message>
      <wsdl:portType name="EmployeeService">
        <wsdl:documentation/>
        <wsdl:operation name="getEmployee">
          <wsdl:input name="getEmployeeRequest" message="tns:getEmployeeRequestMsg"/>
          <wsdl:output name="getEmployeeResponse" message="tns:getEmployeeResponseMsg"/>
        </wsdl:operation>
      </wsdl:portType>
    </wsdl:definitions>
```

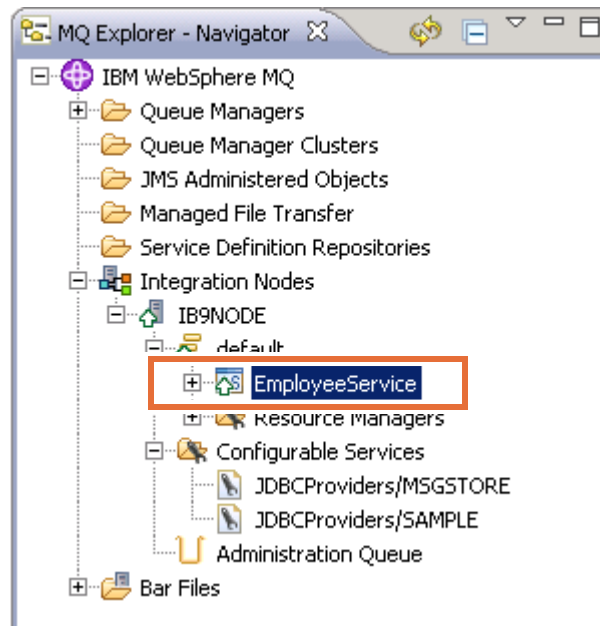


Questions

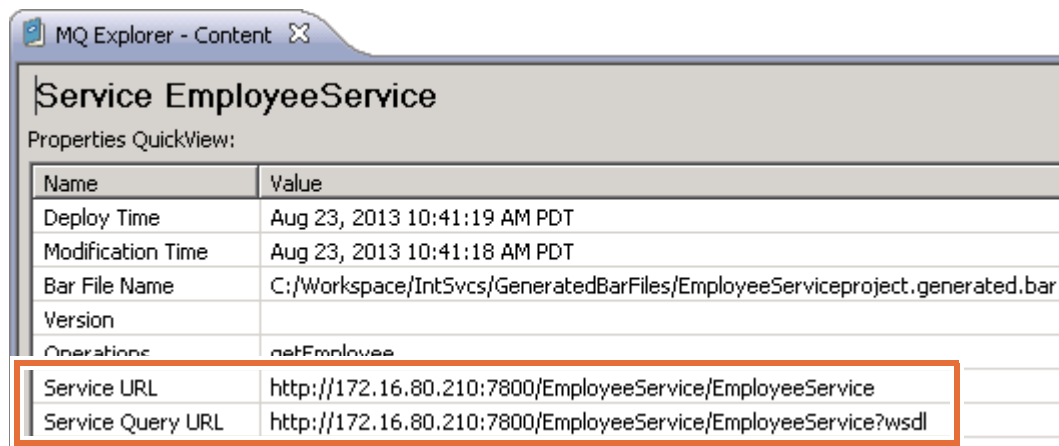
How do you enable the WSDL document retrieval feature in IBM Integration Toolkit?

- ___ 3. Examine the EmployeeService web service from Integration Explorer.
- ___ a. From the desktop, open **IBM Integration Explorer** (Installation 1).
- ___ b. In the **WebSphere MQ Explorer - Navigator** view, expand the **Integration Nodes** folder.

- ___ c. Click **IB9NODE > default > EmployeeService**.



- ___ d. Review the web service endpoint address and the service query URL in the **WebSphere MQ Explorer - Content** view.

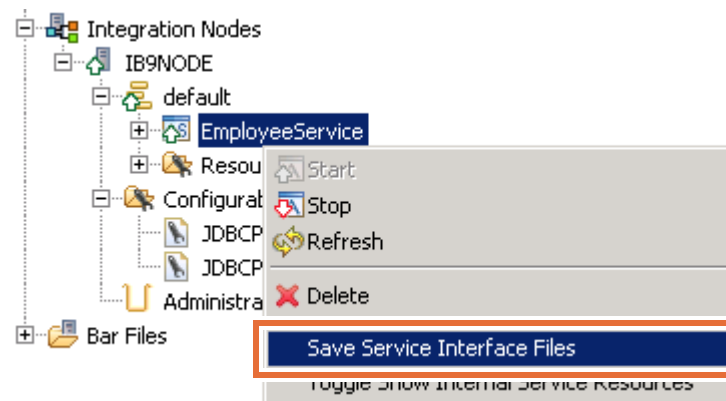


Note

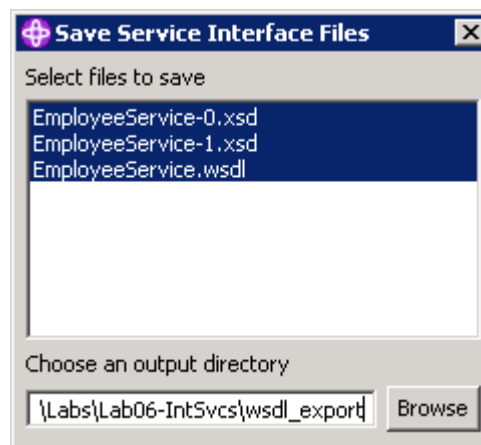
The **Service URL** and **Service Query URL** display the web service endpoint that clients can access over the network. The web service endpoint with localhost is valid for applications that are running on the same system as the Integration server.

- ___ 4. Export the WSDL document that describes the EmployeeService integration service.
- ___ a. From the **WebSphere MQ Explorer - Navigator** view, right-click **EmployeeService**.

- ___ b. Click **Save Service Interface Files**.



- ___ c. In the **Choose an output directory** field, click **Browse** and go to **C:\Labs\Lab06-IntSvcs**.
- ___ d. Click **Make new folder** and then enter: **wsdl_export**.
- ___ e. Click the new folder and then click **OK**.



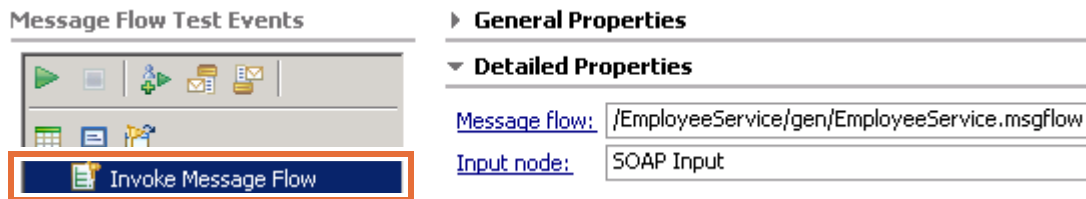
- ___ f. Click **OK**.
- ___ g. Confirm that the application saved the WSDL document and XML schema definition files in the **wsdl_export** directory.

Part 6: Test the web service with the Test Client

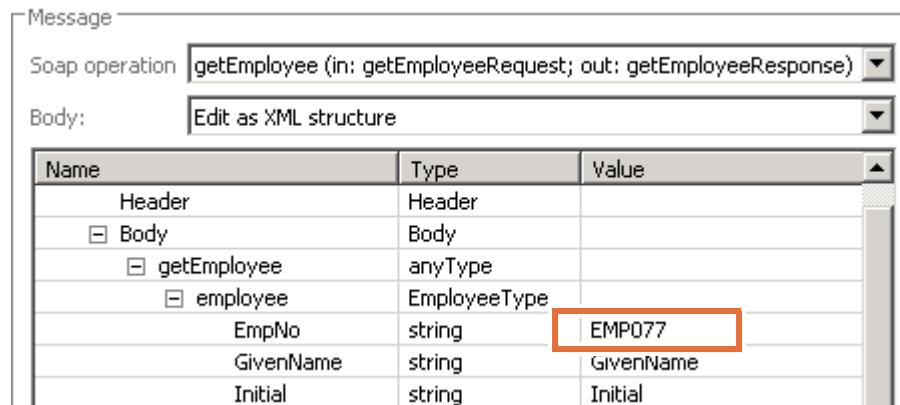
The IBM Integration Toolkit Test Client can test SOAP web services. Use the Test Client to send a SOAP request message to the EmployeeService web service endpoint. View the result of the web service call as an XML message.

- ___ 1. Call the EmployeeService web service with the Test Client.
- ___ a. From the **Application Development** view, right-click the **EmployeeService** integration service project and then click **Test**.

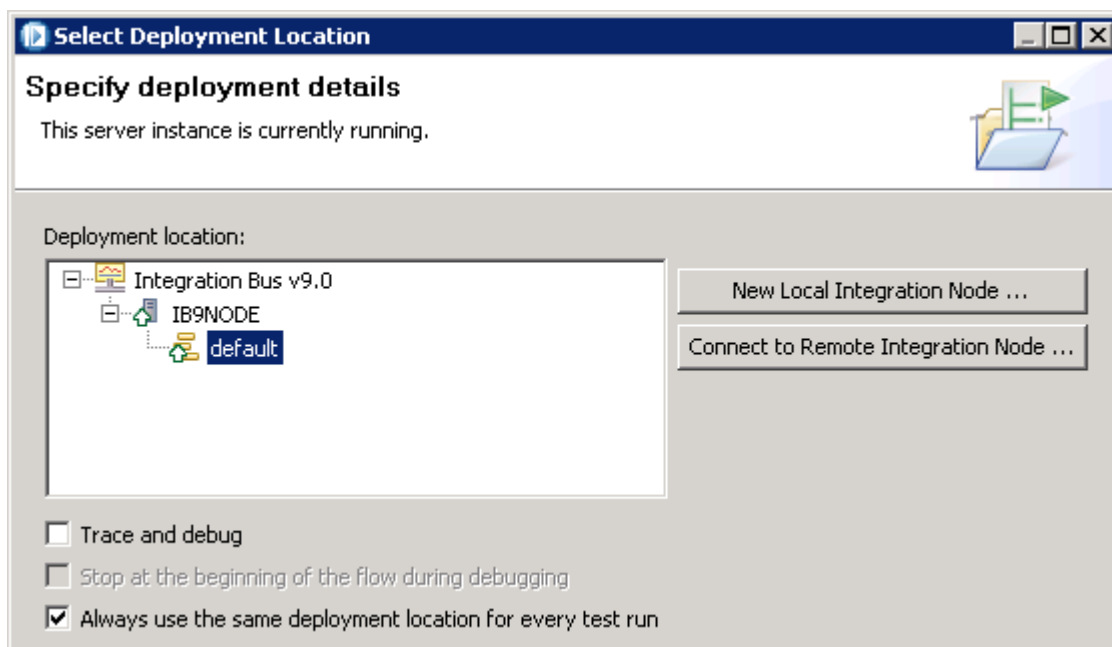
- ___ b. In the Test Client editor, click **Invoke Message Flow**.



- ___ c. Select the **EmpNo** field in the **Value** column.
- ___ d. Enter **EMP077** as the employee number value.



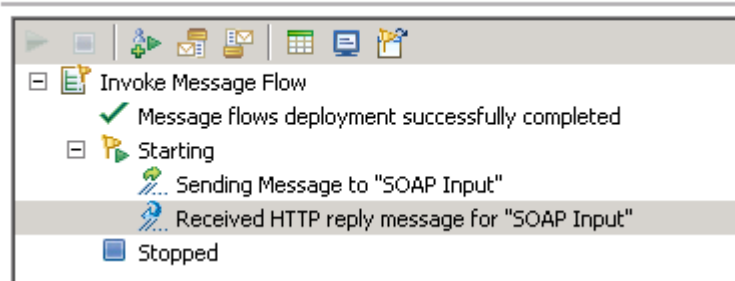
- ___ e. Click **Send Message**.
- ___ f. In the **Select Deployment Location** dialog box, click the **default** integration server.



- ___ g. Click **Finish**.

- ___ 2. Verify the results of the web service operation.
- ___ a. Confirm that the test returns an HTTP reply message.

Message Flow Test Events



- ___ b. Examine the HTTP reply message.

▼ Detailed Properties

Endpoint URL:

Message

Body:

Name	Value
[-] soapenv:Envelope	
xmlns:soapenv	http://schemas.xmlsoap.org/soap/envelope/
[-] soapenv:Body	
[-] io2:getEmployeeResponse	
xmlns:io2	http://EmployeeService
xmlns:io	http://training.ibm.com/Employee
[-] employee	
io:EmpNo	EMP077
io:GivenName	Colin
io:Initial	J
io:FamilyName	Watson
io:Department	C01
io:Phone	4835
io:Role	EMPLOYEE



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 performs an SQL select based on the `EmpNo` field, and completes the employee record details.

- ___ 3. Save and close the Test Client.

Part 7: Optional: Test the web service with soapUI test client

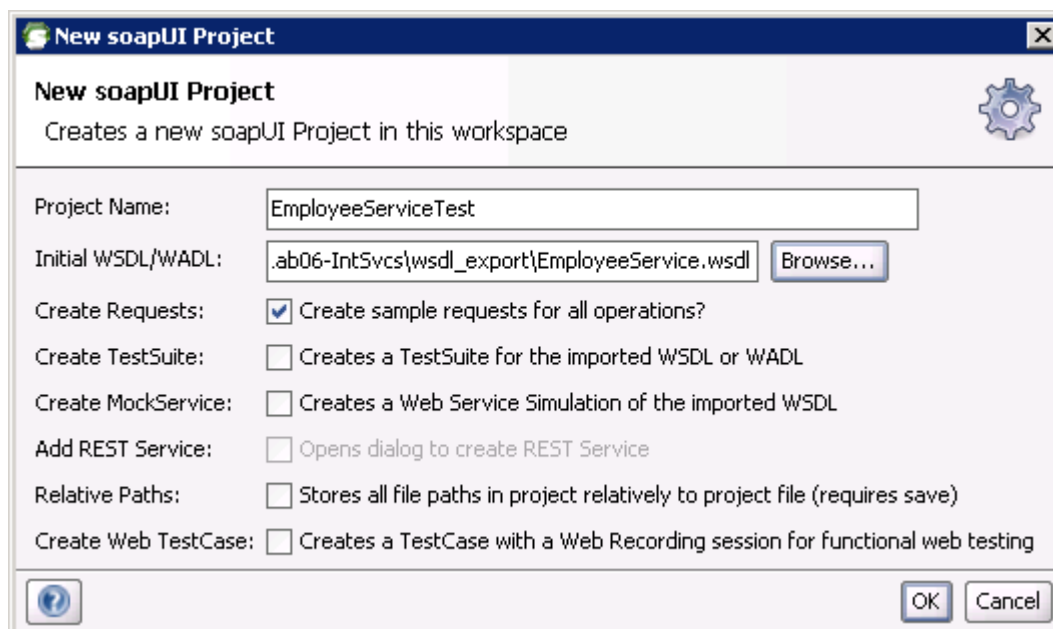
soapUI is an open source functional testing client for SOAP and RESTful web services. It is a third-party product that is widely used for unit testing services.



Optional

This part of the exercise is optional: it is provided for your reference.

- ___ 1. Download and install the soapUI test client.
 - ___ a. Open a web browser and go to **www.soapui.org**.
 - ___ b. Review the licensing terms for the soapUI test client. If you do not agree with the licensing terms, skip to the next part of this exercise.
 - ___ c. Download and install the soapUI test client.
- ___ 2. Start the soapUI application.
- ___ 3. Create a soapUI project.
 - ___ a. From the main menu bar, click **File > New soapUI Project**.
 - ___ b. Enter **EmployeeServiceTest** as the project name.
 - ___ c. In the **Initial WSDL/WADL** field, click **Browse**.
 - ___ d. Go to the **C:\Labs\Lab06-IntSvc\wsdl_export** directory.
 - ___ e. Click **EmployeeService.wsdl**.
 - ___ f. Click **OK** to return to the New soapUI Project wizard.



- ___ g. Click **OK** to create the project.

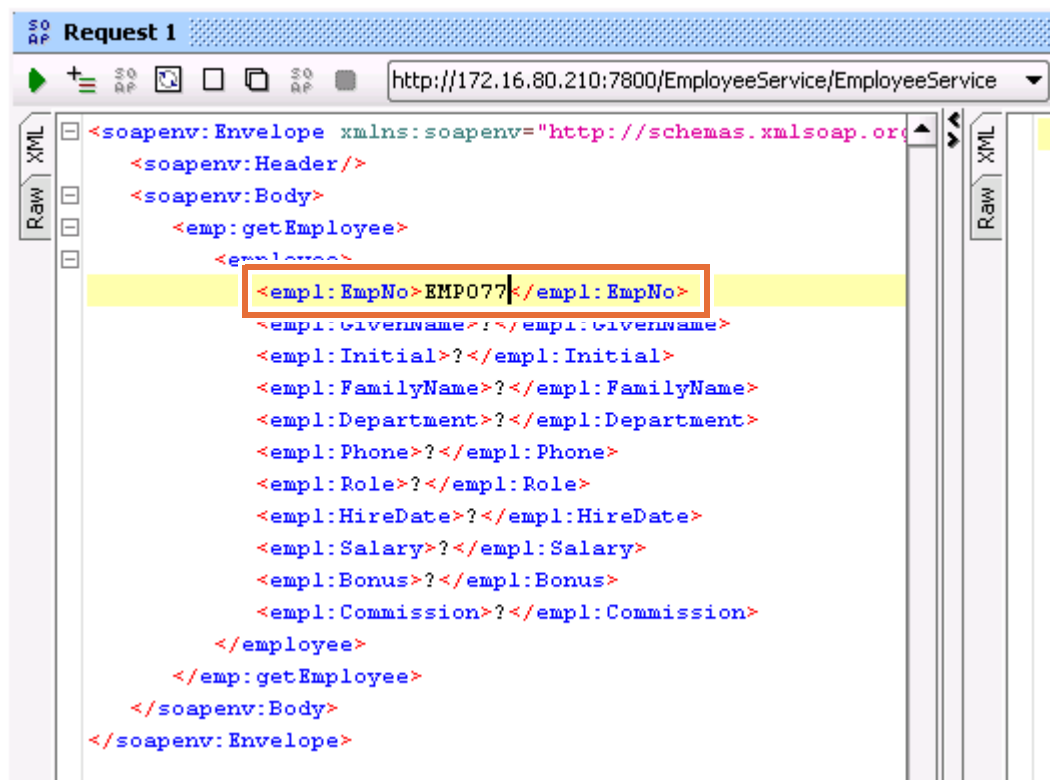


Information

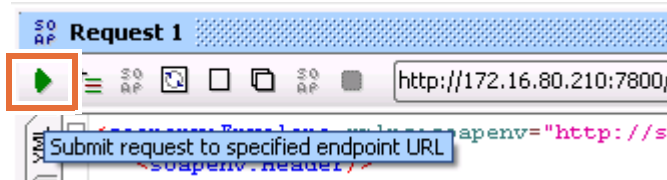
WSDL documents describe the interface, bindings, and endpoint location of a SOAP web service. The Web Application Description Language (WADL) file is an XML document that describes HTTP-based web applications. Specifically, WADL describes the network address and resource elements of a Representational State Transfer (RESTful) service.

The authors of WADL submitted the specification to the World Wide Web Consortium (w3c) in 2009. However, the specification is not standardized and it is not widely used.

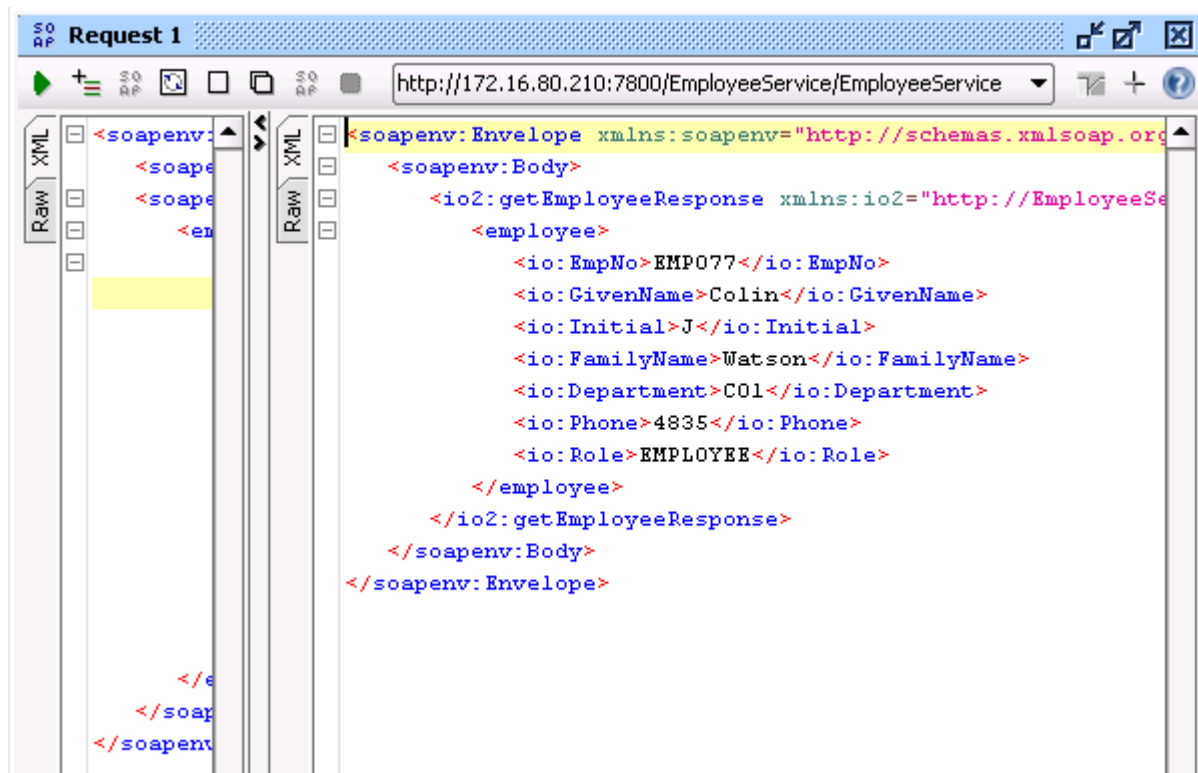
- ___ 4. Create a request message to test the web service.
- ___ a. In the **Navigator** view, select **Projects > EmployeeServiceTest > EmployeeServiceHttpBinding > getEmployee**.
- ___ b. Open **Request 1**.
- ___ c. In the **Request 1** SOAP message window, enter EMP077 as the value of the **emp1:EmpNo** XML element.



- ___ d. From the toolbar, click **Submit request to specified endpoint URL**.



- ___ e. Confirm that the response message contains the employee record for Colin Watson.



- ___ 5. Close the soapUI.

Part 8: Clean up

- ___ 1. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, if you are prompted.
- ___ 2. In the **Integration Nodes** view, right-click the **default** integration server, and then click **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.

End of exercise

Exercise 6. Discovering WebSphere MQ and database services

What this exercise is about

In this exercise, you create a WebSphere MQ request and response service and a database service. You also use the services to define message flow node properties and extend applications.

What you should be able to do

At the end of this exercise, you should be able to:

- Create a WebSphere MQ request and response service definition
- Publish the WebSphere MQ services in an Integration Registry
- Import previously stored WebSphere MQ services into a developers workspace
- Create a database service
- Extend an existing database service by adding more operations
- Use the WebSphere MQ and database services to define message flow node properties
- Use the WebSphere MQ and database services to create and extend applications

Introduction

IBM Integration Bus provides a service discovery feature for WebSphere MQ and database resources. Discovered services are used at development time to create Integration Toolkit WebSphere MQ and database artifacts. These artifacts are then used to create applications. The WebSphere MQ and database artifacts are then deployed to the Integration Bus runtime environment.

WebSphere MQ services can be optionally stored in an Integration Registry component that runs on an integration node. WebSphere MQ services that are stored in this way can be imported by other Integration Bus developers.

In this lab, you create a request/response WebSphere MQ service that is used to develop an application that is called **ManageEmployee**.

The application also uses a discovered database service to manage and retrieve data in a database.

You publish discovered WebSphere MQ services in the Integration Registry so that other developers might develop applications by using the discovered services.

Requirements

- WebSphere MQ and IBM Integration Bus components
- IBM Integration Bus default configuration
- DB2 and the SAMPLE database
- Lab files in the C:\Labs\Lab05-ServiceDisc directory.
- The following WebSphere MQ local queues:
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

Part 1: Preparing the environment

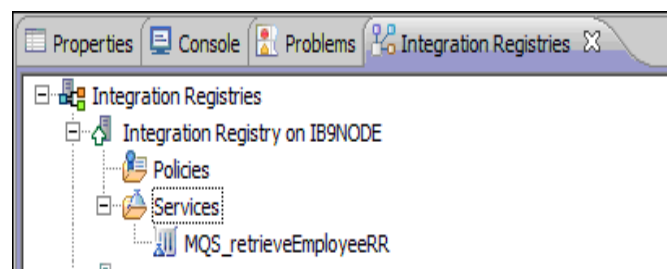
In this part of the exercise, you add an existing WebSphere MQ service to the Integration Registry and import a service definition that is used later in the exercise.

- ___ 1. Open a new workspace in the IBM Integration Toolkit.
- ___ 2. Ensure that the integration node IB9NODE is running. Also, ensure that no flows or resources are deployed to the **default** integration server.
- ___ 3. This lab requires an existing WebSphere MQ service that is stored in the Integration Registry for the integration node that is named IB9NODE. The required WebSphere MQ is available in a Project Interchange file in
C:\Labs\Lab05-ServiceDisc\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. Expand **Other** and then select **Project Interchange**.
- ___ c. Browse to C:\Labs\Lab05-ServiceDisc\install and then click PrimeServiceRegistry.zip.
- ___ d. Click the **PrimeServiceRegistry** project and click **Finish**.
- ___ 4. In the Application Development view, expand the **PrimeRegistryService** library until you see the WebSphere MQ Service definition MQS_retrieveEmployeeRR.service (under the **Services** folder).
- ___ 5. Right-click the Service MQS_retrieveEmployeeRR.service service definition and click **Publish to Registry**.
- ___ 6. Select the Integration Registry for IB9NODE and then click **Finish**.
- ___ 7. After a few moments, the Integration Registry view opens automatically.

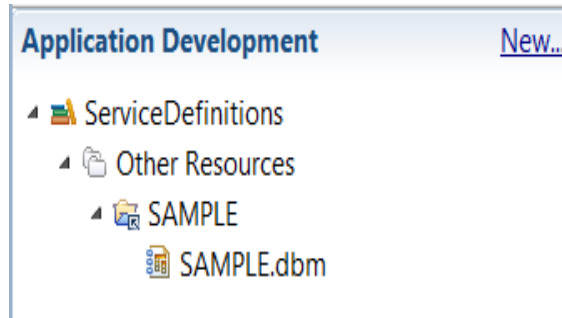
Verify that the **Integration Registries** view contains a service that is named MQS_retrieveEmployeeRR in the **Services** folder.



- ___ 8. After you confirm that the Integration Registry for IB9NODE contains a service, delete the **PrimeServiceRegistry** library from the Application Development view.

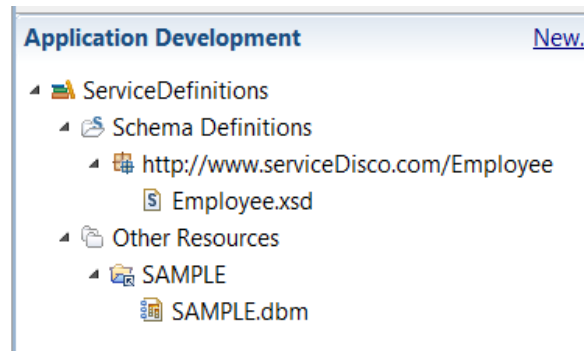
- ___ 9. Create a library that is named `ServiceDefinitions` to hold the WebSphere MQ and DB2 database service definitions that are created in this exercise.
 - ___ a. From the Application Development view, click **New Library**.
 - ___ b. For the library name, enter: `ServiceDefinitions`
 - ___ c. Click **Finish**.
- ___ 10. Include the database definition for the SAMPLE database in the **ServiceDefinitions** library.
 - ___ a. Import the **SAMPLE** project from the
C:\Labs\Lab05-ServiceDisc\resources\SAMPLE.zip Project Interchange file.
 - ___ b. Right-click the **ServiceDefinitions** library in the Application Development view and then click **Manage included projects**.
 - ___ c. Click the **SAMPLE** project and then click **OK**.

The **SAMPLE** project should now be listed within the **Other Resources** folder under the **ServiceDefinitions** library.



- ___ 11. An XML schema that is named `Employee.xsd` is used to define the message types when creating the WebSphere MQ service definitions later in this exercise. Import the `Employee.xsd` file into the **ServiceDefinitions** library.
 - ___ a. In Windows Explorer, browse to the C:\Labs\Lab05-ServiceDisc\resources directory.
 - ___ b. Drag the `Employee.xsd` file from Windows Explorer and drop it onto the **ServiceDefinitions** library in the Application Development view of the IBM Integration Toolkit.

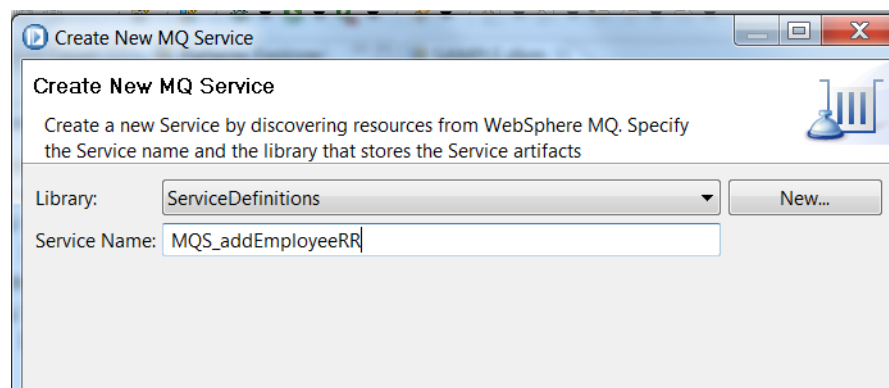
The `Employee.xsd` schema should now be listed under the **Service Definitions** folder of the **Service Definitions** library.



Part 2: Create a WebSphere MQ request/response service definition

In this part of the lab, you create WebSphere MQ Service definitions for queues that are used in a message flow later in this exercise.

- ___ 1. Right-click the **ServiceDefinitions** library in the Application Development view and then click **New > MQ Service**.
- ___ 2. In the **Create New MQ Service** window, enter `MQS_addEmployeeRR` for the **Service Name** and then click **Finish**.



- ___ 3. The WebSphere MQ Service editor opens on the **Description** tab. Click **Next** to advance to the **Connect** page.



Note

Expand the WebSphere 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 IBM Integration Toolkit.
 - ___ a. For the **Queue Manager Name**, enter: `IB9QMGR`

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

The screenshot shows the 'Connect' step in a wizard. The breadcrumb trail at the top is: 1. Description > 2. Connect > 3. Select Resources > 4. MQ Configuration > 5. Service Interface > 6. Summary. Below the breadcrumb, a message box states 'i Connection was tested successfully'. The form contains the following fields:

Type of connection:	Local binding connection
Queue Manager Name:	IB9QMGR
CCDT file URL:	<input type="text"/> Select...
Host or IP:	localhost
Port:	1414
Channel Name:	SYSTEM.BKR.CONFIG

At the bottom of the form, there is a button labeled 'Test Connection'.

- ___ c. Click **Next**.
- ___ 5. The **Select Resources** page lists the queues on queue manager IB9QMGR. On this page, define a request-response operation.
- ___ a. Select the **Select resources for Request-Response operation** option.
- ___ b. 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**.
- ___ c. Select 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.

- ___ d. Select 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.

1. Description ▶ 2. Connect ▶ 3. **Select Resources** ▶ 4. MQ Configuration ▶ 5. Service Interface ▶ 6. Summary

Query and select the resources from the target system

Message Exchange Pattern: ☒ Select resources for Request-Response operation
☐ Select resources for One-Way operation

Filter Text:

Include Resources: ☒ System Queues ☐ Dynamic Queues

To discover MQ Service, select a queue from the following list, then click Select Request to make it your Request queue. Repeat this step, but click Select Response to select a Response queue. (If the list of queues is empty, click Refresh queues to refresh the list)

Queues	Select Request	Select Response
SERVICE.DISCOVERY.DB.ADD.IN		
SERVICE.DISCOVERY.DB.ADD.OUT		<input checked="" type="text" value="SERVICE.DISCOVERY.DB.ADD.OUT"/>
SERVICE.DISCOVERY.DB.DELETE.IN		
SERVICE.DISCOVERY.DB.DELETE.OUT		
SERVICE.DISCOVERY.DB.READ.IN		
SERVICE.DISCOVERY.DB.READ.OUT		
SERVICE.TEST		
SERVICE.TEST.RETRIEVE		

Refresh queues...

- ___ e. Click **Next**.
- ___ 6. The **MQ Configuration** page defines the message headers.
- ___ a. Expand the **Request Message Headers** and **Response Message Headers** sections.
- ___ b. Verify that the **Message type** is set to `Request` for the **Request Message Headers** and set to `Reply` for the **Response Message Headers**.
- ___ c. Click **Next**.
- ___ 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 Part 1 of this exercise.
- ___ a. Under the **Operations** section, click the **Click to select message** link for the **Input**.
- ___ b. Select **EmployeeType** from the list of **Filtered data types** and then click **OK**.
- ___ c. Click the **Click to select message** link for the **Output**.

___ d. Select **EmployeeType** from the list of **Filtered data types** and then click **OK**.

1. Description ▶ 2. Connect ▶ 3. Select Resources ▶ 4. MQ Configuration ▶ 5. **Service Interface** ▶ 6. Summary

Configure your interface and advanced binding parameters

▼ **Interface**

Name	MQS_addEmployeeRR		
Namespace	http://MQS_addEmployeeRR		
Message Configuration	<input checked="" type="radio"/> Use XML schema types	<input type="radio"/> Use XML schema elements	

▼ **Operations**

Operation	Message		
	SERVICEDISCOVERYDBADDIN_SERVICEDISCOVERYDBADDOUT_Operation		
Input	<input checked="" type="checkbox"/> SERVICEDISCOVERYDBADDIN_SERVICEDISCOVERYDBADDOUT_Operation [EmployeeType]	Open file...	
Output	<input checked="" type="checkbox"/> SERVICEDISCOVERYDBADDIN_SERVICEDISCOVERYDBADDOUT_OperationResponse [EmployeeType]	Open file...	

___ e. Click **Next**.

___ 8. On the **Summary** page, review the service structure and then click **Save**.

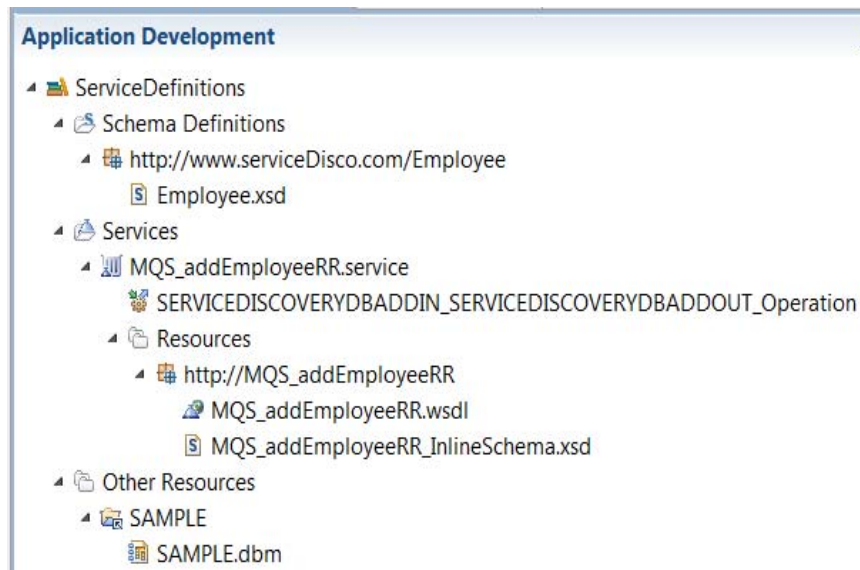
1. Description ▶ 2. Connect ▶ 3. Select Resources ▶ 4. MQ Configuration ▶ 5. Service Interface ▶ 6. **Summary**

Review the Service structure and the list of files that are generated when you save a Service

- Interface
 - MQS_addEmployeeRR
 - MQS_addEmployeeRR
 - SERVICEDISCOVERYDBADDIN_SERVICEDISCOVERYDBADDOUT_Operation
 - EmployeeType
 - EmployeeType
- Data Types
 - Employee

___ 9. The Application Development view now contains the `MQS_addEmployeeRR.service` under the **Services** folder in the **ServiceDefinitions** library.

Close the **MQS_addEmployeeRR.service** tab.



Part 3: Publish the WebSphere MQ service to an Integration Registry

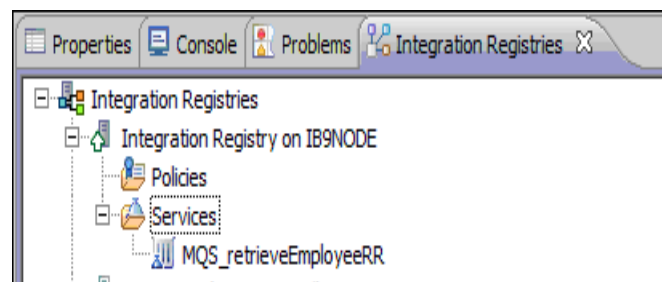
In this part of the exercise, you store the WebSphere MQ service that you created into the Integration Registry for IB9NODE.

- ___ 1. If it is not already open, open the **Integration Registries** view in the IBM Integration Toolkit by clicking **Window > Show view > Other > Integration Development > Integration Registries**.

The view automatically attempts to connect to integration nodes that are running.

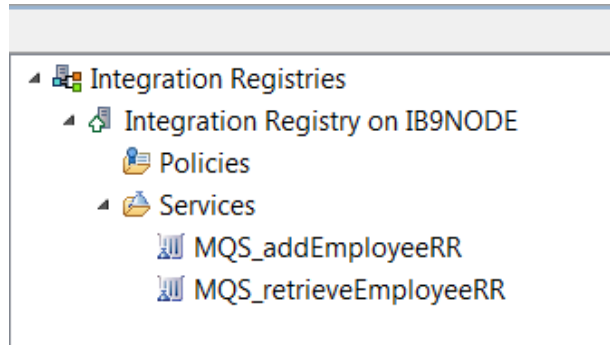
- ___ 2. Expand the IB9NODE registry view to show the contents of the **Policies** and **Services** folders.

The **Services** folder already contains the **MQS_retrieveEmployeeRR** service that you published to the registry in Part 1 of this exercise.



- ___ 3. In the Application Development view, right-click the **MQS_addEmployeeRR.service Services** entry and then click **Publish to Registry**.
- ___ 4. Click **IB9NODE** and then click **Finish**.

After a few moments, the **MQS_addEmployeeRR** service should be listed under the **Services** folder of the **Integration Registries** view.



Information

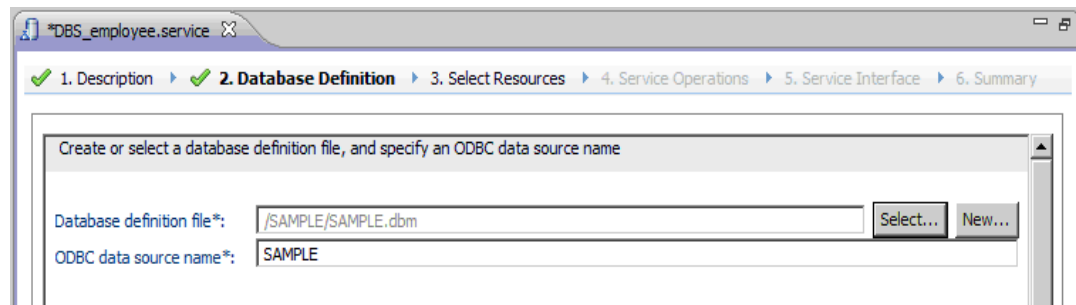
Other developers with access to the IB9NODE Integration Registry can use the service definitions that are stored in the IB9NODE Integration Registry. Right-clicking the service definition name and clicking **Import to workspace** copies the definition into a library in the workspace.

Part 4: Discover the database service

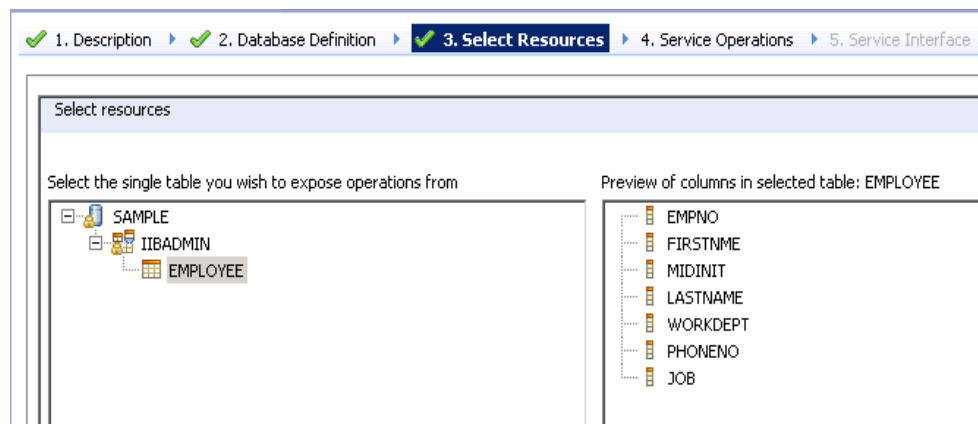
In this part of the exercise, you create a database service that is based on the `SAMPLE.dbm` file that you imported in Part 1 of this exercise. You create a database service that is used to perform add (INSERT) and retrieve (SELECT) operations on an EMPLOYEE table in the SAMPLE database.

- ___ 1. Right-click the **ServiceDefinitions** library in the Application Development view and then click **New > Database Service**.
- ___ 2. Name the new service `DBS_employee` and then click **Finish**.
- ___ 3. The Database Service editor opens. Click **Next** to go to the **Database Definition** page.
- ___ 4. On the **Database Definition** page, click **Select** to select a database definition file.
- ___ 5. Click `SAMPLE.dbm` in the SAMPLE project and then click **OK**.

- ___ 6. The details of the SAMPLE database are copied into the **Database definition file** and **ODBC data source name** fields.

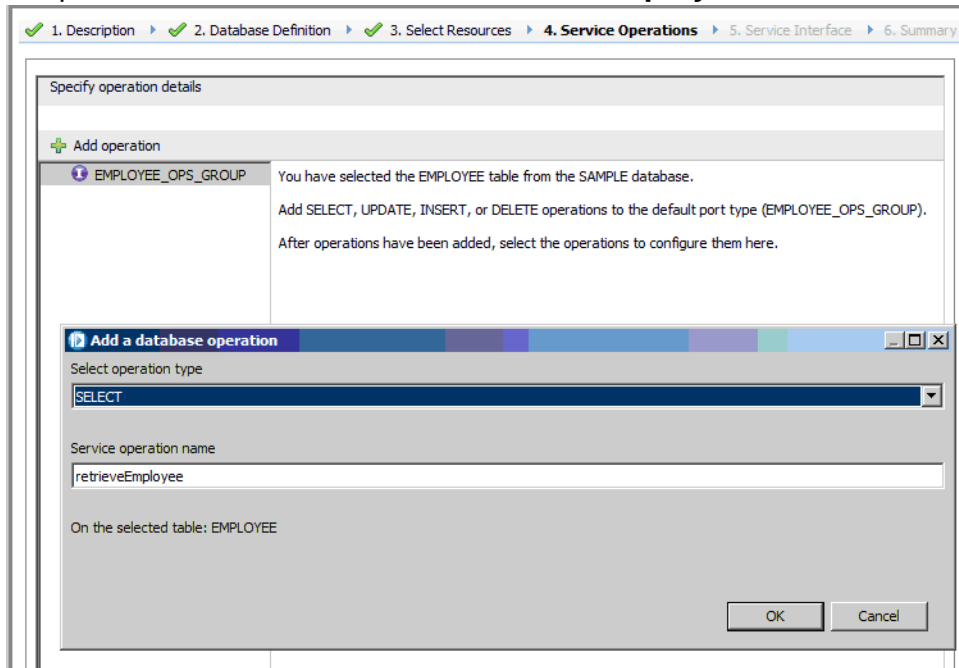


- ___ 7. Click **Next**.
- ___ 8. On the **Select Resources** page, select the EMPLOYEE table. The **Preview of columns** pane is updated with columns of the selected table.

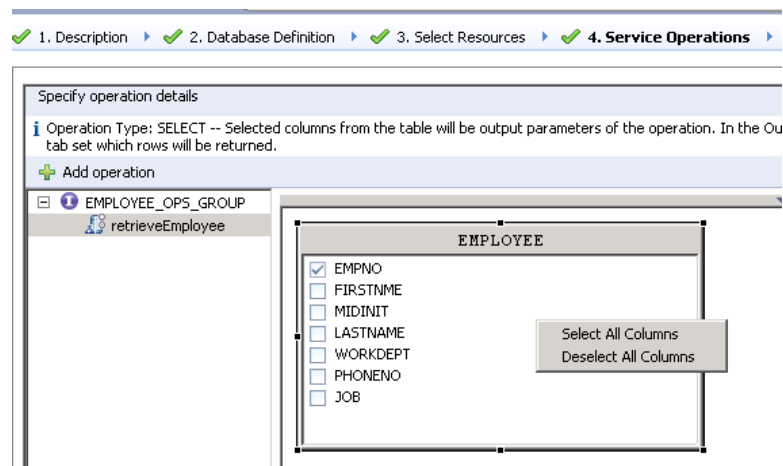


- ___ 9. Click **Next**.
- ___ 10. On **Service Operations** page, click **Add operation**.

- ___ 11. On the **Add a database operation** window, choose **SELECT** for the operation type and keep the default service name of **retrieveEmployee**. Click **OK**.

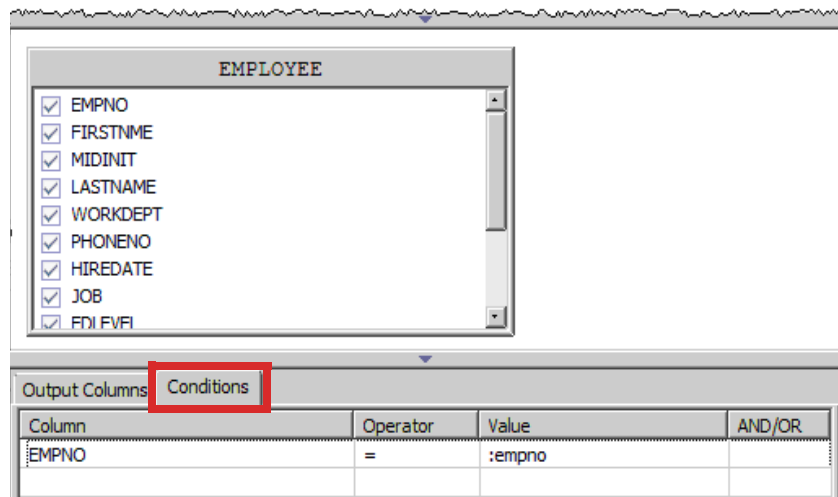


- ___ 12. A list of column names that can be selected as output parameters of **retrieveEmployee** operation are displayed. EMPNO is selected by default. Right-click the EMPLOYEE table and click **Select All Columns**.

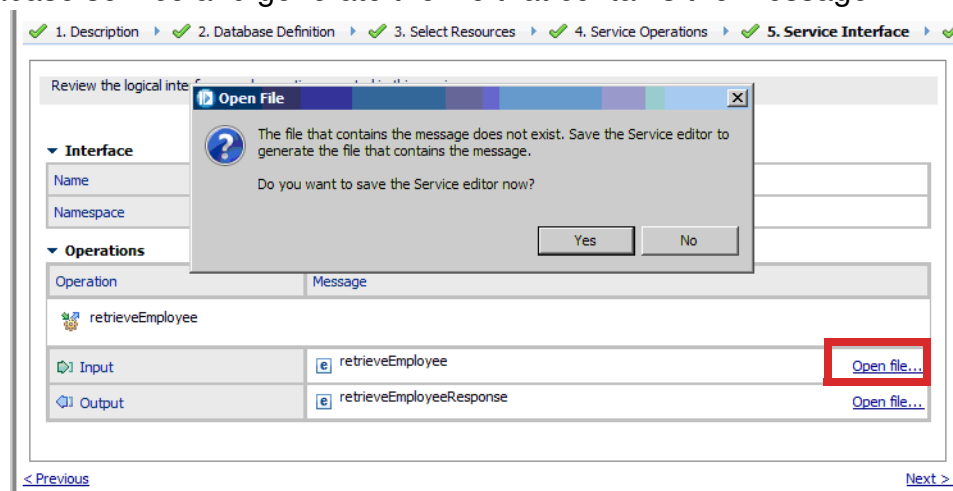


- ___ 13. Click the **Conditions** tab.
- ___ 14. Click an empty cell under **Column** and then select EMPLOYEE.EMPNO from the EMPLOYEE table.

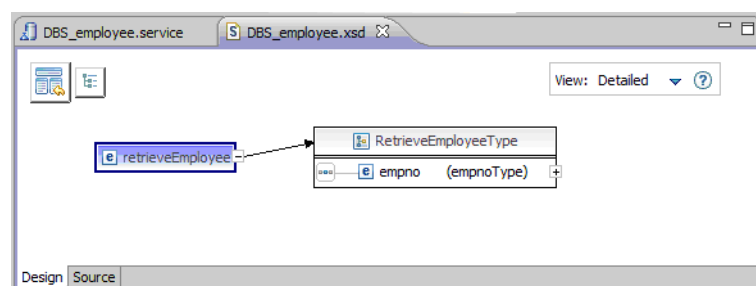
The **Operator** column is set to = and the **Value** column is set to `empno` (the input parameter) automatically.



- ___ 15. Click **Next**.
- ___ 16. On the **Service Interface** page, click the **Open file** link on the **retrieveEmployee** Input row.
- ___ 17. 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.

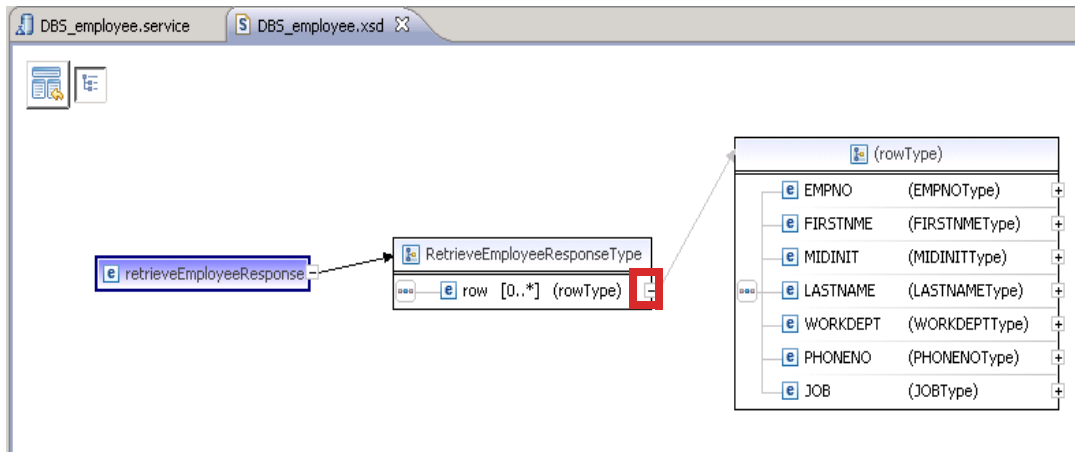


- ___ 18. After it is generated, the **retrieveEmployee** input XML schema file (`DBS_employee.xsd`) is opened in the XML Schema editor.

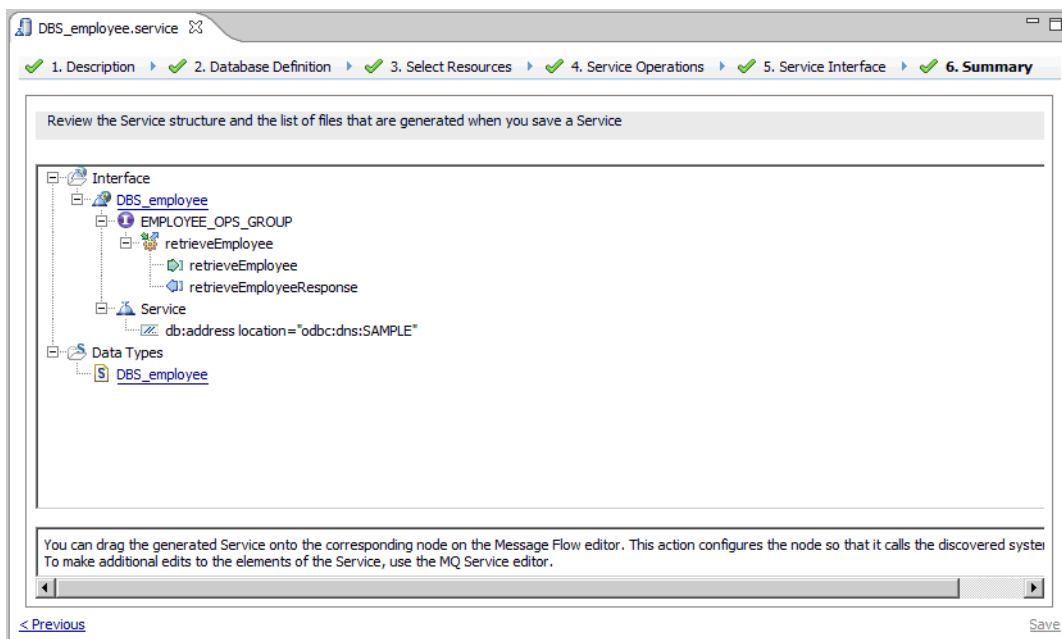


- ___ 19. Close the DBS_employee.xsd file.
- ___ 20. On the **Service Interface** page of the Database Service editor, click the **Open file** link for the **retrieveEmployeeResponse** output.

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.



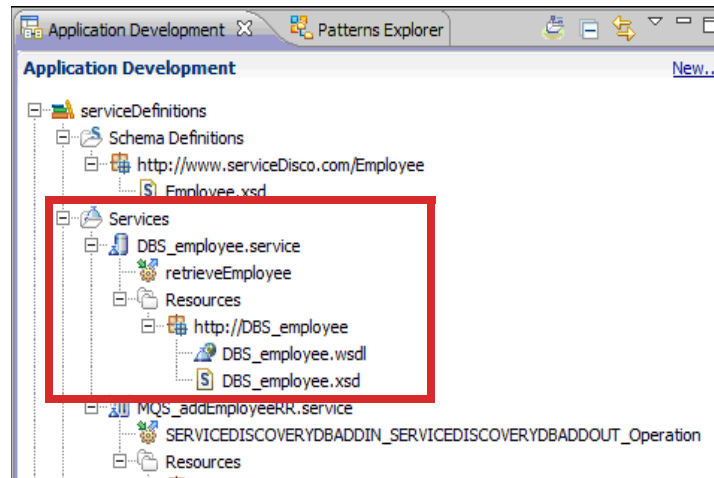
- ___ 21. Close the DBS_employee.xsd file.
- ___ 22. Click **Next** to go to the **Summary** page.



- ___ 23. 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.

- ___ 24. Now that you have created the database service, the Application Development view is updated to include the database service under the **Services** folder in the **ServiceDefinitions** library:

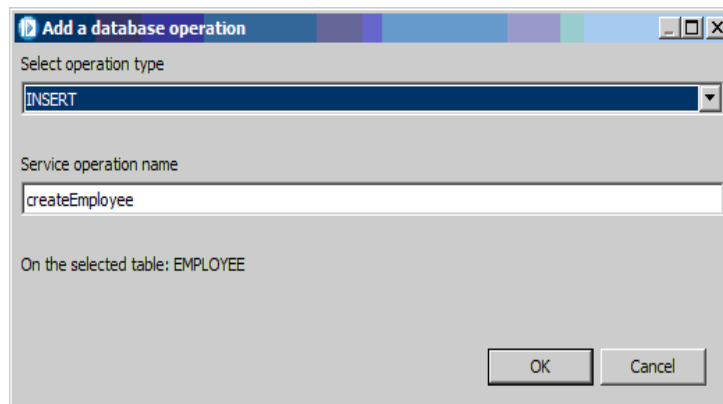


Part 5: Add an operation to the database service

The application that you create later in this lab adds and retrieves data in the database. 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, select **4. Service Operations**.
The operation list contains the **retrieveEmployee** operation that you created in Part 4 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.

Column	Value
EMPNO	:empno
FIRSTNAME	:firstname
MIDINIT	:midinit
LASTNAME	:lastname
WORKDEPT	:workdept
PHONENO	:phoneno
JOB	:job

- ___ 6. Click **Next**.
- ___ 7. The **createEmployee** operation is added to the **Service Interface** page.

Review the logical interfaces and operations created in this service

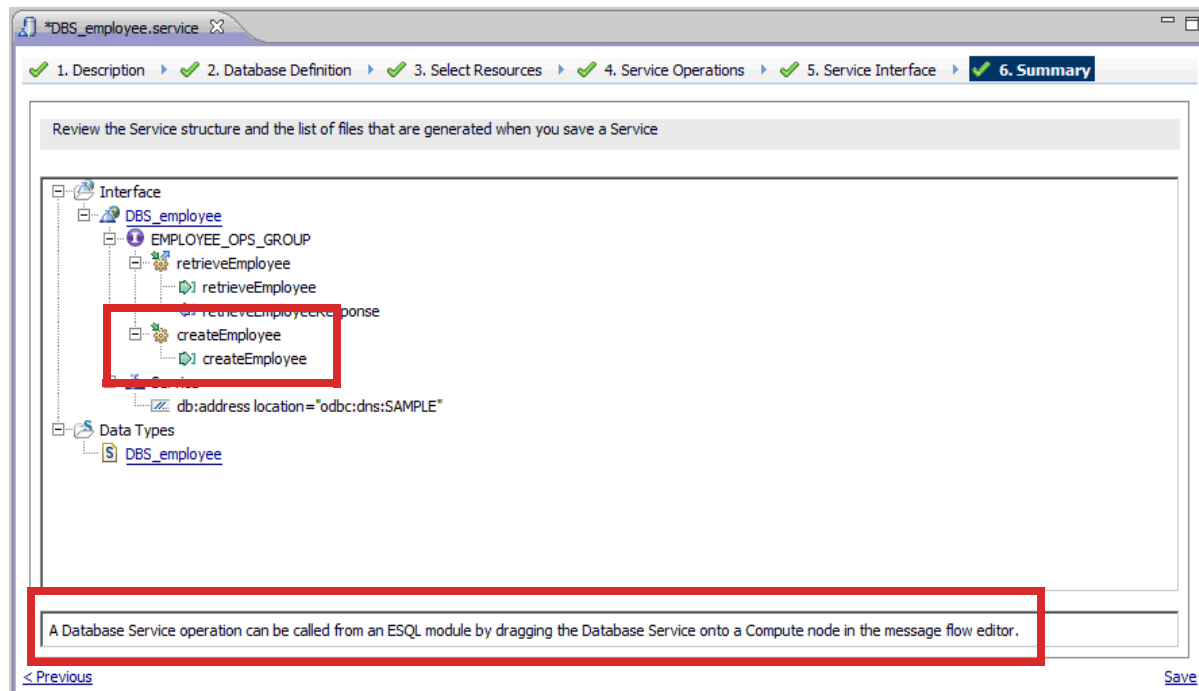
Interface

Name	EMPLOYEE_OPS_GROUP
Namespace	http://DBS_employee

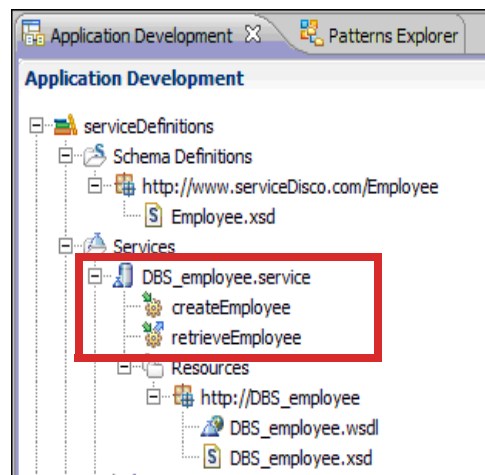
Operations

Operation	Message
retrieveEmployee	
Input	retrieveEmployee Open file...
Output	retrieveEmployeeResponse Open file...
createEmployee	
Input	createEmployee Open file...

- ___ 8. Click **Next**.
- ___ 9. 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.



- ___ 10. Click **Save** to add the **createEmployee** operation to the database service.
- ___ 11. Close the Database Service editor.
- ___ 12. Verify that the **createEmployee** operation is listed under the **DBS_employee.service** in the Application Development view.

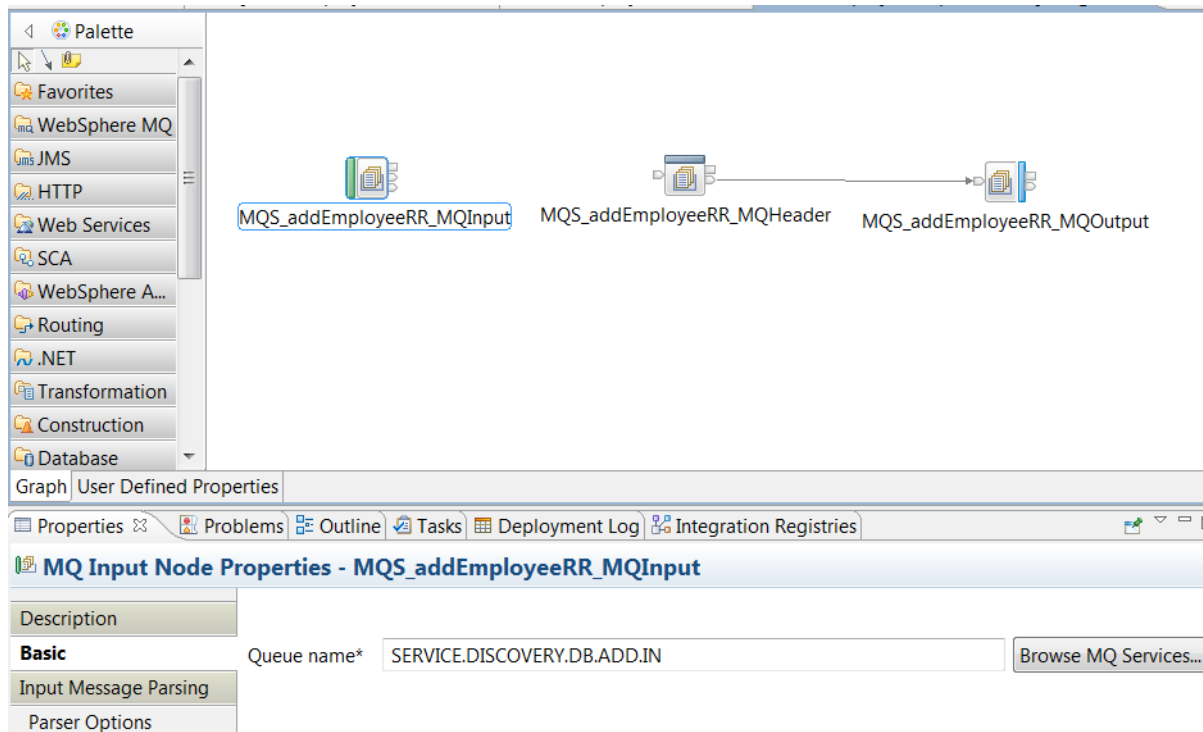


Part 6: Implementing a WebSphere MQ service

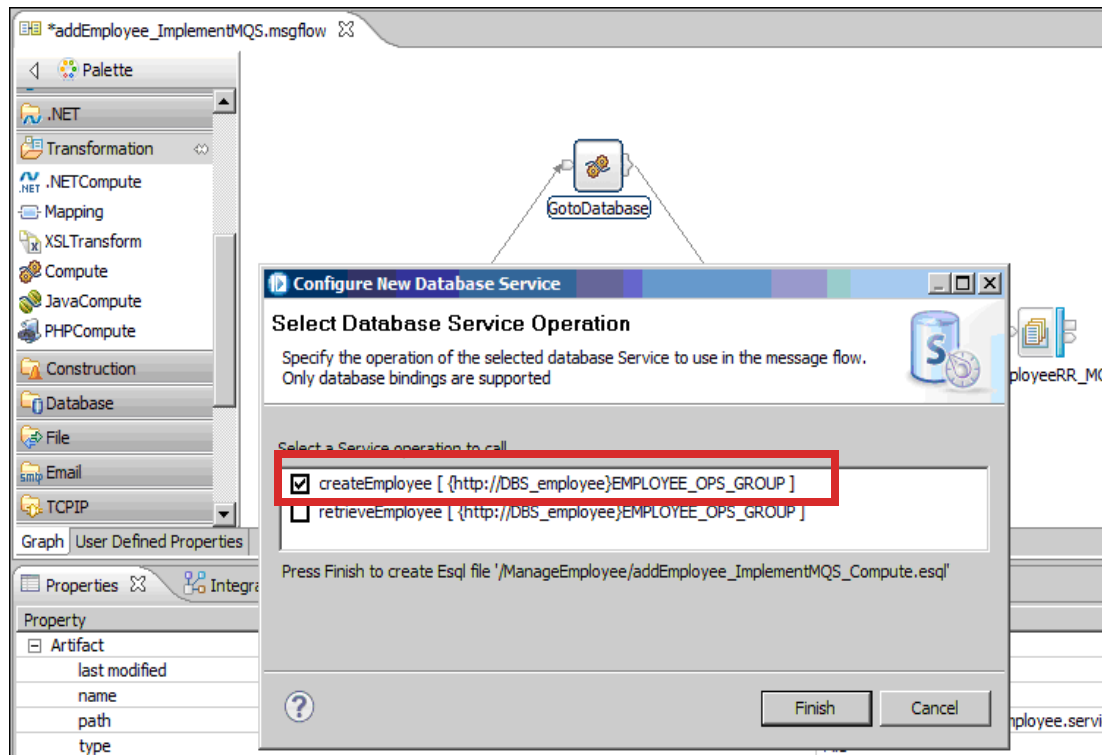
In this part of the exercise, you create an application that uses the WebSphere MQ request-response service and database service that you created in this exercise.

WebSphere MQ services can be called or implemented when using them in an IBM Integration Bus message flow. The application that you create in this part of the exercise shows how each of these options can be used to create an application that is based on the WebSphere MQ service and a database service.

- ___ 1. Create an application that is called `ManageEmployee` with a reference to the library that contains the service definitions.
 - ___ a. In the Application Development view, click **New > Start by creating an application**.
 - ___ b. For the **Application name**, enter: `ManageEmployee`
 - ___ c. Click **Next**.
 - ___ d. Click the **ServiceDefinitions** library to reference it in the new application.
 - ___ e. Click **Finish**.
- ___ 2. Create a message flow that is named `addEmployee_ImplementMQS` that uses the **MQ_addEmployeeRR** service to implement the WebSphere MQ service.
 - ___ a. In the Application Development view, click the **ManageEmployee** application.
 - ___ b. Click **New > Message Flow**.
 - ___ c. For the message flow name, enter: `addEmployee_ImplementMQS`
 - ___ d. Click **Finish**. The Message Flow editor opens.
 - ___ e. Drag the **MQS_addEmployeeRR.service** from the Application Development view (under the **ManageEmployee > References > Service Definitions > Services** folder) onto the Message Flow editor canvas.
 - ___ f. When prompted, choose to option to implement the WebSphere MQ service.
 - ___ g. When implementing the service, the following nodes are automatically created:
 - An MQ Input node that is named `MQS_addEmployeeRR_MQInput` with `SERVICE.DISCOVERY.DB.ADD.IN` configured for the input queue.
 - An MQ Header node that is named `MQS_addEmployeeRR_MQHeader` and is wired to an MQ Output node.
 - An MQ Output node that is named `MQS_addEmployeeRR_MQOutput` with the `SERVICE.DISCOVERY.DB.ADD.OUT` configured for the output queue.



- ___ 3. The message flow implements the WebSphere MQ service that is designed for request and response. This message flow accepts the request on the input queue and responds to the caller on the output queue.
Use the database service to create the request.
 - ___ a. Add a Compute node onto the canvas. Name the node **GotoDatabase**.
 - ___ b. Wire the **Out** terminal on the MQ Input node to the **In** terminal on the **GotoDatabase** Compute node.
 - ___ c. Wire the **Out** terminal of the **GotoDatabase** Compute node to the **In** terminal of the MQ Header node.
 - ___ d. Drag the database service **DBS_employee.service** from the Application Development view onto the **GotoDatabase** Compute node.
 - ___ e. When prompted to select the database service operation, click **createEmployee**.



- ___ f. Click **Finish**.
- ___ g. Dropping the database service definition onto the Compute node automatically:
 - Creates an ESQL procedure that is called **createEmployee**
 - Generates an ESQL procedure that is named `DBS_employee.esql` and saves it in the library
 - Opens the ESQL procedure in the ESQL editor in the IBM Integration Toolkit

Close the ESQL editor; do not change this file.

- ___ 4. Replace the ESQL code for the **GotoDatabase** Compute node.
 - ___ a. Double-click the **GotoDatabase** compute node to open the ESQL template code.
 - ___ b. Replace the contents of the ESQL file with the ESQL in the `addEmployee_ImplementMQS_Compute.esql.txt` file in the `C:\Labs\Lab05-ServiceDisc\resources` directory.

```
PATH DBS_employee.EMPLOYEE_OPS_GROUP;
DECLARE ns NAMESPACE 'http://www.serviceDisco.com/Employee';

CREATE COMPUTE MODULE addEmployee_ImplementMQS_Compute
  CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
    -- CALL CopyMessageHeaders();
    CALL CopyEntireMessage();

    DECLARE insRef REFERENCE TO InputRoot.XMLNSC.ns:Employee;

    DECLARE EMP1 CHAR insRef.EmpNo;

    CALL createEmployee(EMP1,
      insRef.FirstName,
      insRef.Initial,
      insRef.LastName,
      insRef.Dept,
      insRef.Phone,
      insRef.Job);

    RETURN TRUE;
  END;
```

- ___ c. Save and close the addEmployee_ImplementMQS_Compute.esql file.
- ___ 5. Save the **addEmployee_ImplementMQS** message flow.

Part 7: Calling a WebSphere MQ service

In this part of the exercise, you use the **MQ_addEmployeeRR** service to call the WebSphere MQ service.

- ___ 1. Create a message flow that is named addEmployee_CallMQS in the **ManageEmployee** application.
 - ___ a. In the Application Development view, click the **ManageEmployee** application and then click **New > Message Flow**.
 - ___ b. Name the message flow addEmployee_CallMQS and click **Finish**. The Message Flow editor opens.
- ___ 2. Use the **MQS_addEmployeeRR.service** to create the message flow.
 - ___ a. Drag the **MQS_addEmployeeRR.service** from the Application Development view and drop it onto the message flow canvas.
 - ___ b. Select the option to **Call an MQ Service**.

Three nodes are added to the message flow:

- An MQ Header node
- An MQ Output node that uses the Request queue that you defined in the WebSphere MQ service (SERVICE.DISCOVERY.DB.ADD.IN)
- An MQ Get node that uses the Response queue that you defined in the WebSphere MQ service (SERVICE.DISCOVERY.DB.ADD.OUT)

The message flow writes to the SERVICE.DISCOVERY.DB.ADD.IN queue. It waits until a response is received on the SERVICE.DISCOVERY.DB.ADD.OUT queue, which calls the WebSphere MQ service to add a user.

- ___ c. The default setting for the MQ Header node

MQS_addEmployeeRR_MQHeader is to add a WebSphere MQ header.

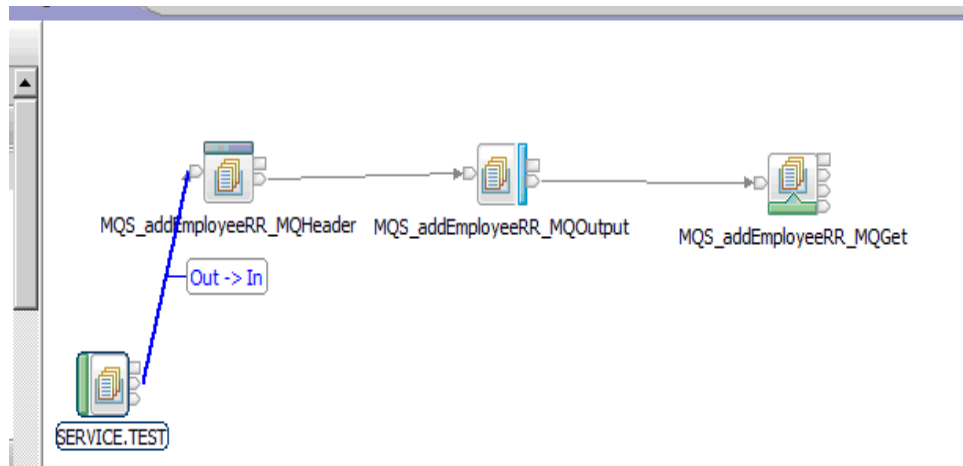
On the **MQMD** properties for the MQ Header node, set the **MQMD header options** to **Carry forward header**.

MQ Header Node Properties - MQS_addEmployeeRR_MQHeader		
Description		
MQMD	MQMD header options	<input checked="" type="radio"/> Carry forward header <input type="radio"/> Add header <input type="radio"/> Modify header
Report		
MQDLH	Coded Character Set Identifier	<Inherit from header>
Monitoring	Format	<Inherit from header>
	Version Number	<Inherit from header>
	Message	<Inherit from header>

- ___ 3. To test this message flow, add an MQ Input node and a File Output node so that the message flow can run in your environment.

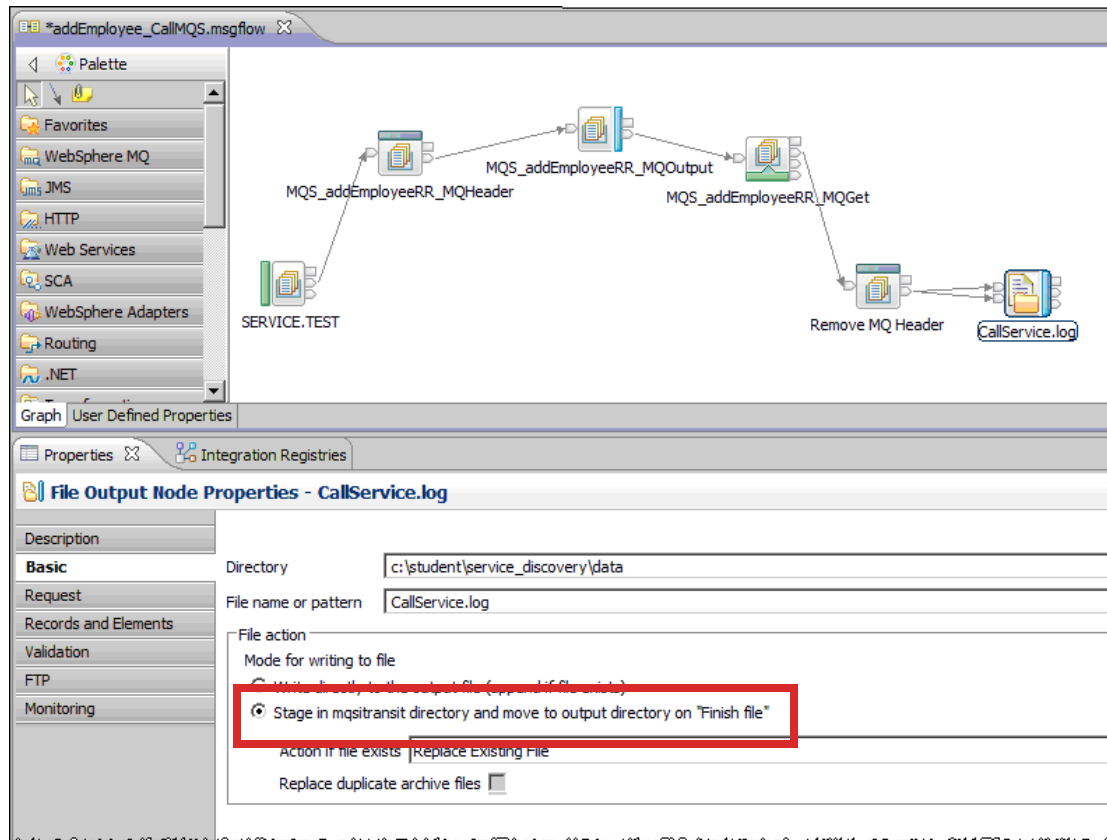
You also add an MQ Header node to remove the WebSphere MQ header before the message is sent to the File Output node.

- ___ a. An MQ Input node to the message flow.
- ___ b. Rename the MQ Input node to `SERVICE.TEST`, set the input queue name to `SERVICE.TEST`, and the **Input Message Parsing** to `XMLNSC`.
- ___ c. Wire the **Out** terminal on the **SERVICE.TEST** MQ Input node to the **In** terminal on the **MQS_addEmployeeRR_MQHeader** node.



- ___ d. Add an MQ Header node to the message flow and rename it to **Remove MQ Header**.
- ___ e. In the properties of the MQ Header node, set **MQMD header options** property to **Delete Header**.
- ___ f. Wire the **Output** terminal of the MQ Get node that is called **MQS_addEmployeeRR_MQGet** to the **In** terminal of the **Remove MQ Header** node.
- ___ g. Add a File Output node to the message flow and change the name of the node to **CallService.log**.
- ___ h. On the **Basic** properties:
 - Set the **Directory** to **C:\Labs\Lab05-ServiceDisc\data**
 - Set the **File name or pattern** to **Callservice.log**
 - Set **File Action** to **Stage in mqsitransit directory**
- ___ i. Wire the **Out** terminal of the **Remove MQ Header** node to the **In** terminal of the **CallService.log** node.

- ___ j. Wire the Out terminal of the **Remove MQ Header** node to the **Finish** terminal of the **CallService.log**.

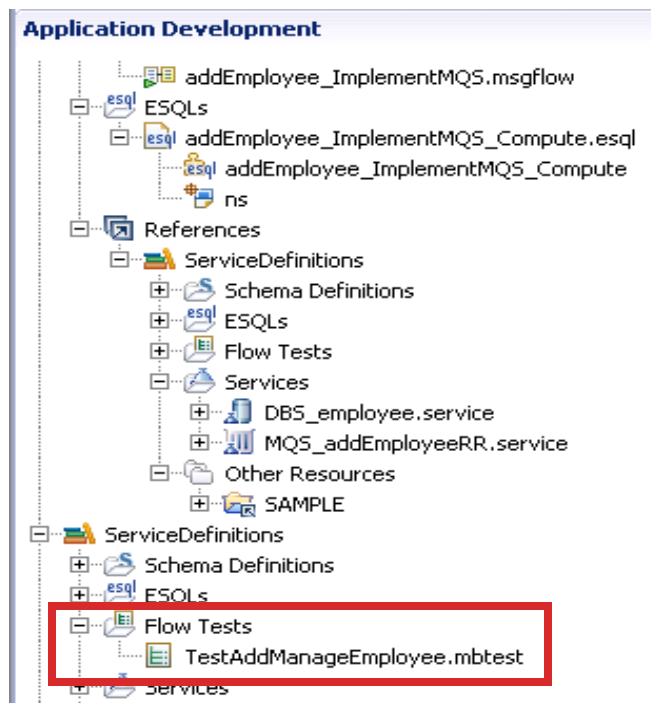


- ___ 4. Save the message flow.
- ___ 5. Check the **Problems** view and verify that there are no problems. If there are no problems, close the Message Flow editor.

Part 8: Deploy and test the application

- ___ 1. Create an integration server on IB9NODE that is named **ServiceDiscovery**
- ___ a. In the **Integration Nodes** view, right-click IB9NODE and then select **New Integration Server**.
- ___ b. For the integration server name, enter: ServiceDiscovery.
- ___ 2. Deploy the **ManageEmployee** application into the **ServiceDiscovery** integration server by dragging the application from the Application Development view to the integration server in the **Integration Nodes** view.
- ___ 3. Start the IBM Integration Toolkit Test Client with a predefined test file.
- ___ a. Open Windows Explorer.
- ___ b. Browse to the C:\Labs\Lab05-ServiceDisc\resources directory.

- ___ c. Drag the file `TestAddManageEmployee.mbttest` from onto the **ServiceDefinitions** library in the IBM Integration Toolkit Application Development view. It should be added to the library under the **Flow Tests** folder.



- ___ d. Double-click the `TestAddManageEmployee.mbttest` file in the Application Development view to start the Test Client.

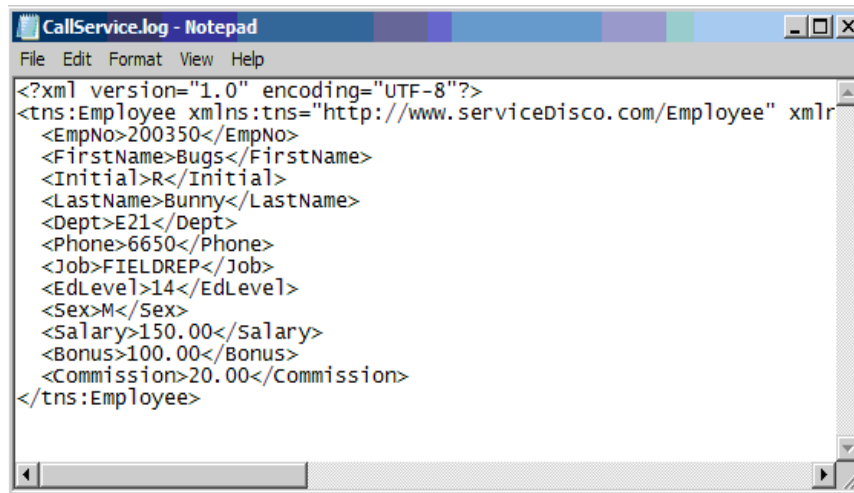
- ___ 4. When you click **Send Message** on the Test Client, the XML message that is displayed is sent to the `SERVICE.TEST` queue. The message is then picked up by the **addEmployee_CallMQS** message flow and written to `SERVICE.DISCOVERY.DB.ADD.IN`.

The message flow **addEmployee_ImplementMQS** then gets this request. It adds the user with `EMPNO=200350` to the database by using the ESQL provided by implementing the database service **DBS_employee** and the **createEmployee** procedure.

The final result of this test is a file that is named `CallService.log` written to the `C:\Labs\Lab05-ServiceDisc\data` directory.

Click **Send Message**.

- ___ 5. After the Test Client stops, verify that the XML passed through the two message flows is written to the `CallService.log` file in the `C:\Labs\Lab05-ServiceDisc\data` directory



```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Employee xmlns:tns="http://www.serviceDisco.com/Employee" xmlns:xs="http://www.w3.org/2001/XMLSchema-
  <EmpNo>200350</EmpNo>
  <FirstName>Bugs</FirstName>
  <Initial>R</Initial>
  <LastName>Bunny</LastName>
  <Dept>E21</Dept>
  <Phone>6650</Phone>
  <Job>FIELDREP</Job>
  <EdLevel>14</EdLevel>
  <Sex>M</Sex>
  <Salary>150.00</Salary>
  <Bonus>100.00</Bonus>
  <Commission>20.00</Commission>
</tns:Employee>
```



Note

The Test Client ends by timing out because the last node in the flow is a File Output output node and not a queue.

- ___ 6. Check the EMPLOYEE table on the SAMPLE database to verify that the row is added.
 - ___ a. Open the DB2 Control Center by clicking **IBM DB2 > DB2COPY1 > General Administration Tools > Control Center** from the Windows **Start > All Programs** menu.
 - ___ b. Click **Basic** and click **OK**.
 - ___ c. Expand the SAMPLE database and click **Tables**.
 - ___ d. Double-click the EMPLOYEE table to show the contents of the table.
 - ___ e. Verify that a row was added for EMPNO=200350

Open Table - EMPLOYEE						
WS2008R2X64 - DB2 - SAMPLE - IIBADMIN.EMPLOYEE						
Edits to these results are performed as positioned UPDATES and DELETES. Use the Tools Settings notebook to change						
EMPNO	FIRSTNAME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	JOB
EMP012	Willie	F	Makeit	E01	4547	MANAGER
EMP025	Justin	Q	Public	E01	4524	EMPLOYEE
EMP106	Betty	M	Bacon	C01	4891	MANAGER
EMP077	Colin	J	Watson	C01	4835	EMPLOYEE
EMP301	Rebecca	L	Sunset	C01	4890	EMPLOYEE
200350	Bugs	R	Bunny	E21	6650	FIELDREP

- ___ f. Close the DB2 Control Center.

- ___ 7. Clean up your lab environment by deleting all flows and resources from the **ServiceDiscovery** integration server.
- ___ 8. Close all open editors.

End of exercise

Exercise review and wrap-up

In Part 1 of the exercise, you prepared the lab environment by importing and publishing a WebSphere MQ service, importing an XML schema that contained the message types, and importing a database definition for the DB2 SAMPLE database.

In Part 2 of the exercise, you created a WebSphere MQ request and response service definition.

In Part 3 of the exercise, you published the WebSphere MQ request and response service definition to the Integration Registry for IB9NODE.

In Part 4 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 5 of the exercise, you added an operation to a database service.

In Part 6 of the exercise, you created a message flow that implemented a WebSphere MQ service.

In Part 7 of the exercise, you created a message flow that called a WebSphere MQ service. You also used a database service to create the request.

In Part 8 of the exercise, you deployed and tested the application.

Exercise 7. Implementing a message flow that uses JMS

What this exercise is about

In this exercise, you implement and test a message flow that uses JMS as the transport.

What you should be able to do

At the end of this exercise, you should be able to:

- Import a message flow that uses JMS message processing nodes
- Review the JMS configuration parameters in the message flow
- Configure the message flow to use the Trace node to log messages
- Test the message flow and interpret the trace results

Introduction

IBM Integration Bus can act as both a producer and a consumer of JMS services. The JMSInput node allows a message flow to use JMS messages. The JMSOutput node allows a message flow to produce JMS messages for an external JMS provider. These messages can use either point-to-point or publish-and-subscribe topologies.

WebSphere MQ is able to act as a JMS provider. This exercise takes advantage of that capability.

In this exercise, you import the JMSNodes sample from the samples gallery. This sample consists of two message flows and a stand-alone Java application.

- The **JMSGateway** message flow receives a message from WebSphere MQ, and then modifies the message assembly by adding a time stamp. It then removes the WebSphere MQ message descriptor (MQMD) and creates a JMS message header by using an MQJMSTransform node. It then passes the message to a JMSOutput node, which writes the message to a JMS queue.
- The **JMSInput_Publication** message flow retrieves the message from the JMS queue by using a JMSInput node. The message is then converted to a WebSphere MQ message by using a JMSMQTransform node. This node removes the JMS header and

creates an MQMD. The message then enters a Compute node, which adds a publication topic and command to the MQRFH2 header. Finally, the message is routed to a Publication node.

- The **stand-alone Java application** acts as the subscriber. It reads the published message that the JMSInput_Publication message flow creates. It then print on the console the message it received.

After importing the sample, you examine the JMS configurations that the JMS message processing nodes use. You add Trace nodes to the message flows to review the message assembly in the trace output. You then test the flows and review the trace output.

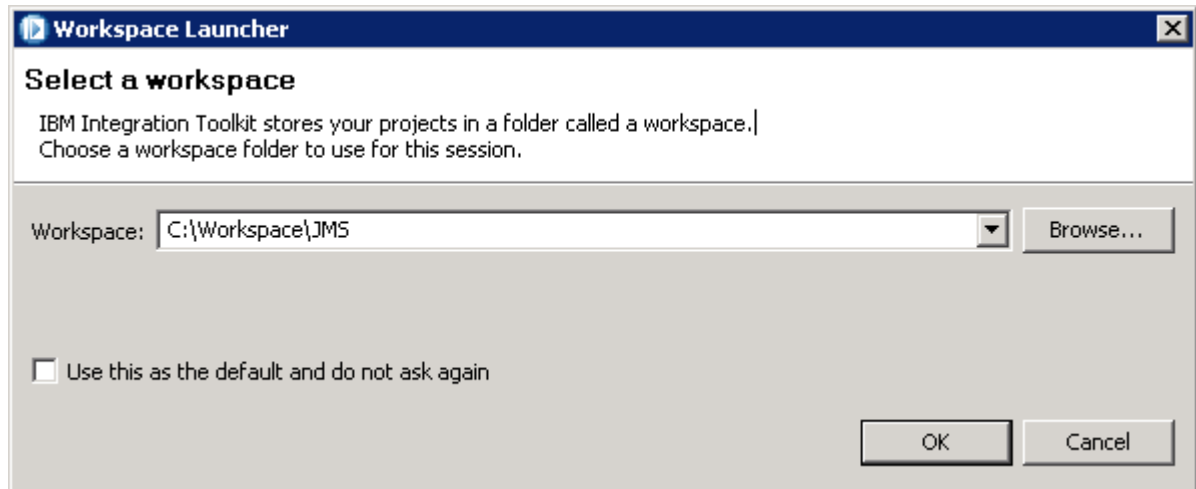
Requirements

- A workstation with the IBM Integration Bus components installed.
- The IBM Integration Bus default configuration
- IBM WebSphere MQ SupportPac IH03, RfhUtil

Exercise instructions

Part 1: Start components and prepare the workspace

- ___ 1. Make sure that all local IBM Integration Bus components are running.
- ___ 2. Start the IBM Integration Toolkit if it is not already running.
- ___ 3. Open a new workspace, such as: C:\Workspace\JMS.



- ___ 4. When the **Welcome** window is displayed, close it. The IBM Integration Toolkit opens, and the **Application Development** view is displayed.

Part 2: Import the JMSNodes sample

In this section, you import the sample from the Samples Gallery.

- ___ 1. Prepare to import the sample.
 - ___ a. In the **Quick Start** pane, click **Start from samples**.

Quick Starts

Start building your application with one of the following tasks.



[Start by creating an application](#)

An **Application** is a container for all the resources that are required to create a solution. [More...](#)



[Start by creating a library](#)

A **Library** is a logical grouping of related code, data, or both. [More...](#)



[Start from WSDL and/or XSD files](#)

Use this task to create an **Application** or **Library** which includes your WSDL and/or XSD files.



[Start from patterns](#)

A **Pattern** is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task. [More...](#)



[Start from samples](#)

Use the **Samples** to learn more about the features in WebSphere Message Broker. [More...](#)

- ___ b. Scroll to the **Transports and Connectivity** section, and then expand it.

- ___ c. Scroll to the **JMS Nodes sample**, and then click the link.

JMS Synchronous Request sample

This sample shows how to call an IBM Information Management System (IMS) transaction synchronously from within a message flow. The sample uses the IMSRequest node to make the synchronous calls by using IMS Connect. This sample uses the IMS sample transaction DSPALLI (Display All Invoices), which is typically available on all IMS systems. The DSPALLI transaction can call a REXX or COBOL program, although REXX is the default that is typically installed on IMS.

JMS Nodes sample

This sample shows how to use the JMS nodes as a JMS Consumer and Producer to an external JMS provider.

MQHeader Node sample


This message flow sample application shows how to use the MQHeader node to add and remove an MQMD header.

SAP Call Out to an Asynchronous System sample


The tutorial for the sample opens in the Information Center.

- ___ d. Click **Read about the sample**.

Click the following links to find out more about the sample and how to get using the wizards.


 **Import and deploy: 5 minutes**

 [Read about the sample](#)

 You can set up the sample in one of the following ways:

Review the description of the sample, including the explanation of the message flows.

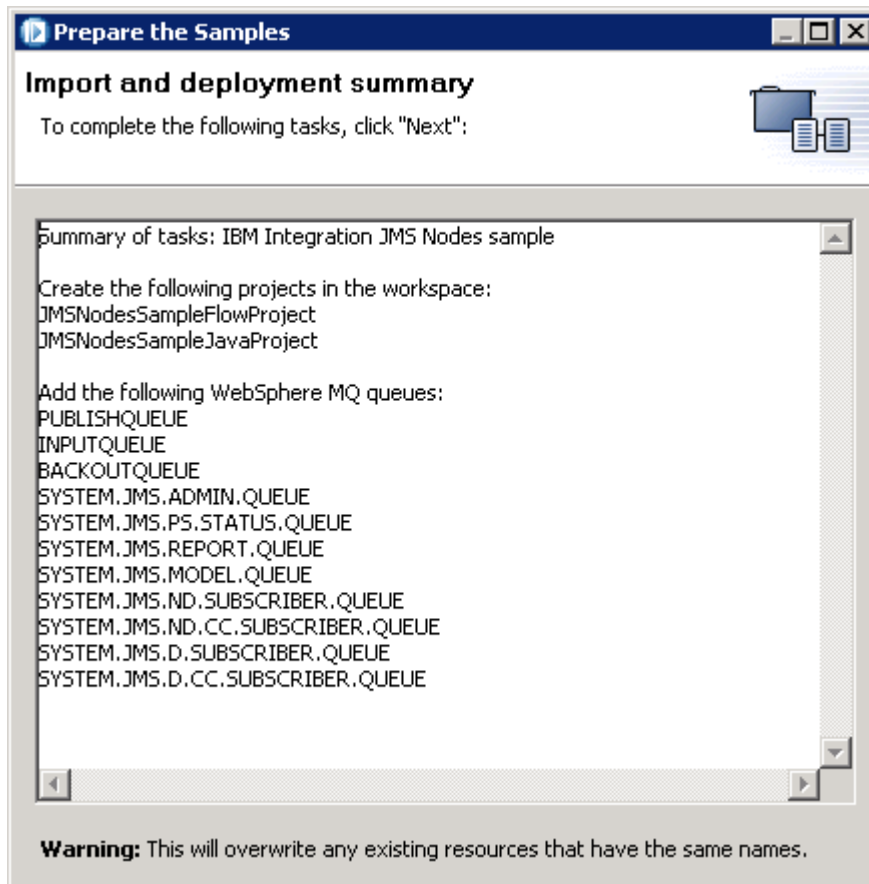
- ___ e. After you finish reviewing the description, scroll to the bottom, and then click **Back to sample home**.

 [Back to sample home](#)

You can also click the left arrow in the Information Center toolbar to return to the sample home page.



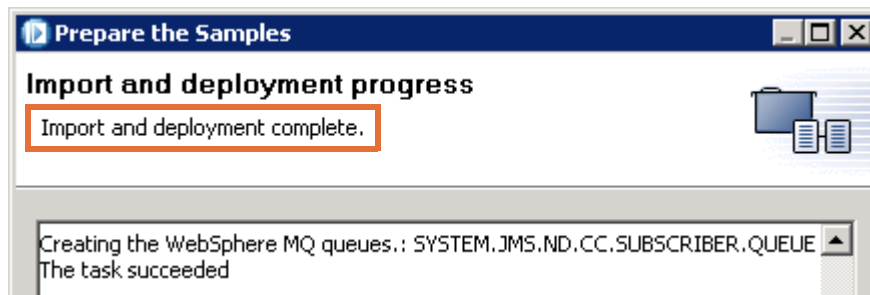
- ___ 2. Import the sample and prepare the configuration.
- ___ a. Click **Import the sample and create the queues**. The **Import and deployment summary** dialog box is displayed.



As the dialog box indicates, the following steps are performed (in the next step):

- The **JMS Nodes Sample Flow project** is imported. The project includes both message flows described previously.
 - The **JMS Nodes Sample Java Project** is imported. This project contains the stand-alone Java subscriber application.
 - The **PUBLISHQUEUE**, **INPUTQUEUE**, and **BACKOUTQUEUE** queues are created in WebSphere MQ. The message flows and stand-alone Java application use these queues.
 - A number of **SYSTEM.JMS.*** queues are created. IBM Integration Bus and WebSphere MQ use this system queues to manage the JMS activities in the message flows.
- ___ b. Click **Next**. The files are imported and the queues are created.

- ___ c. Wait until **Import and deployment complete** is displayed in the progress area.



- ___ d. Scroll through the messages and ensure that the tasks completed successfully. You see the All tasks were completed successfully message at the bottom of the window if the import and queue creation tasks completed.

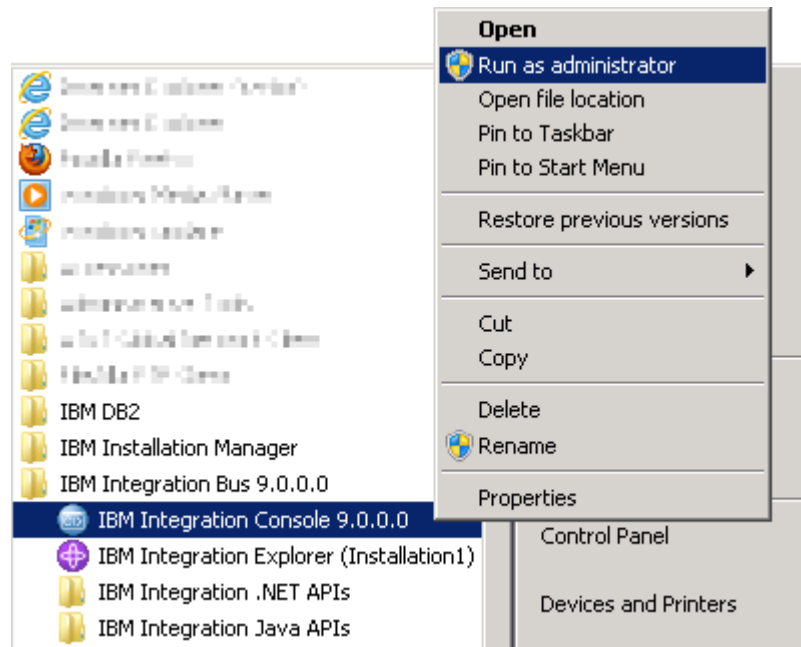
```

+++++++
All tasks were completed successfully.
+++++++

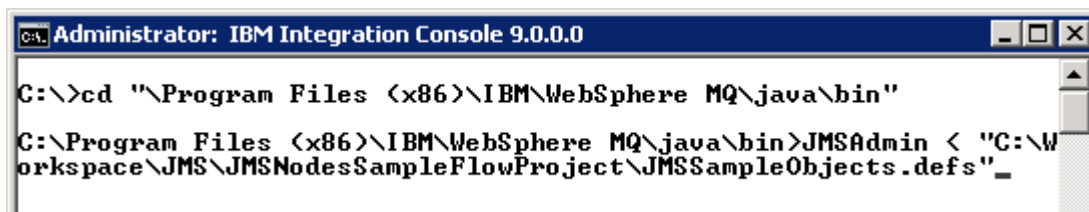
```

- ___ e. If you want to review the actual commands that were entered and the resulting messages, click **Open Log File**. Click **OK** after you are done reviewing the log.
- ___ f. Click **Next**. A message is displayed that indicates the sample import was successfully completed.
- ___ g. Click **Finish**. The sample description in the Information Center is displayed again.
- ___ 3. Review the setup instructions.
- ___ a. In the sample description, click **Setup instructions**.
The instructions describe a number of steps that you must perform if you are not running the sample in a Windows environment, or if you are not using WebSphere MQ as the JMS provider.
- ___ 4. Create the queue connection factories with the JMSAdmin command.
- ___ a. From the Windows Start menu, click **All Programs > IBM Integration Bus 9.0.0.0 > IBM Integration Console 9.0.0.0**.

- ___ b. Right-click the selection and click **Run as administrator**.



- ___ c. In the User Account Control dialog box, click **Yes** to run the application as an administrator.
- ___ d. In the command prompt, go to the Websphere MQ Java binaries directory:
`cd "%Program Files (x86)%\IBM\WebSphere MQ\java\bin"`
- ___ e. Execute the JMSAdmin command:
`JMSAdmin < "C:\Workspace\JMS\JMSNodesSampleFlowProject\JMSSampleObjects.defs"`



- ___ f. Confirm that the JMSAdmin script completed without any errors.
- ___ g. Close the integration console.



Information

The JMSSampleObjects.defs file defines three JMS administered objects:

```
# Define a QueueConnectionFactory
# Only parameters being overridden from their default values are specified.
# This sets up a MQ client binding.
DEF QCF(qcf1) TRANSPORT(CLIENT) QMANAGER(IB9QMGR) HOSTNAME(127.0.0.1) +
```

```

PORT(2414)
DEF Q(publishqueue) QUEUE(PUBLISHQUEUE) QMANAGER(IB9QMGR)
DEF Q(backoutqueue) QUEUE(BACKOUTQUEUE) QMANAGER(IB9QMGR)
END

```

The **qcf1** queue connection factory object creates connections between the JMS application client and the JMS provider: IBM Integration Bus.

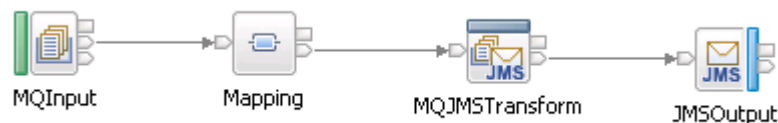
The two JMS queue objects, **publishqueue** and **backoutqueue**, provides JMS access to the two Integration Bus queues **PUBLISHQUEUE** and **BACKOUTQUEUE**.

Without these objects, the JMS application client cannot access the queues hosted by the **IB9QMGR** queue manager.

Part 3: Review the imported components

In this section, you examine the components that were imported. You review the message flows and their components, and the configuration settings on the JMS nodes.

- ___ 1. Review the **JMSGateway** message flow.



This message flow:

- Retrieves a message from a WebSphere MQ queue
 - Transforms the message in a Mapping node by adding a time stamp
 - Removes the WebSphere MQ message descriptor (MQMD) and adds a JMS header
 - Sends the message to a JMSOutput node
- ___ a. In the Application Development view, expand **Independent Resources > JMS Nodes Sample Flow Project > Flows**, and then double-click **JMSGateway.msgflow**. The message flow opens in the Message Flow editor.
 - ___ b. Right-click the **MQInput** node, and then click **Properties**.
 - ___ c. Review the **Basic** tab and the **Input Message Parsing** tab.

The node reads from which queue? _____

What parser does the node use? _____

- ___ d. Double-click the **Mapping** node. Where does the JMSGateway_Mapping message map insert the current date and time?

- ___ e. Review the **MQJMSTransform** node. Does it have any configurable properties?

- ___ f. Examine the **JMSOutput** node.
Does the node send its output to a JMS publication topic, or a JMS queue?

- ___ g. On the **JMS Connection** tab, identify the location of the **JMS bindings** file. This location is where you find the JMS binding information.



Note

The location of the JMS binding information depends on the operating system where the IBM Integration Bus runtime component is installed.

- ___ 2. Open the JMS bindings file:
 - ___ a. Start a Windows Explorer session. You can do so by right-clicking **Start** in the Windows taskbar, and then clicking **Explore**.
 - ___ b. Browse to the directory, and double-click the file to open it in Notepad.
- ___ 3. Review the details of the **publishqueue** JMS queue.
 - ___ a. Enter Ctrl + F (or click **Edit > Find** from the toolbar) to open a search dialog box in Notepad.
 - ___ b. In **Find what**, enter: `publishqueue`
 - ___ c. Click **Find Next** several times until you see a search result that shows the **FactoryName**.

**Information**

The factory name is:

```
publishqueue/FactoryName=com.ibm.mq.jms.MQQueueFactory
```

This line indicates that the WebSphere MQ JMS queue connection factory is used to connect the application to WebSphere MQ. If you continue to search for the `publishqueue` string, you see references to the queue manager where the JMS queue connection is made, and other details about the environment.

```
publishqueue/RefAddr/17/Content=IB9QMGR
```

- ___ d. Search the JMS bindings file for the **Connection factory name** value that is displayed in the **JMS Connection** tab.

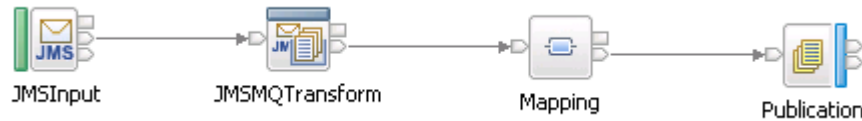
**Information**

You see more information about the WebSphere MQ environment that is related to the connection factory, such as the system control queues and the WebSphere MQ attributes that are used to establish the JMS connection. These attributes include the port number, the connection options, the message type, and other connection information.

```
qcf1/RefAddr/4/Type=PORT  
qcf1/RefAddr/73/Type=XMSC_RTT_PROXY_HOSTNAME  
qcf1/RefAddr/4/Content=2414  
qcf1/RefAddr/84/Encoding=String
```

- ___ e. After you browse the file, click **Cancel** to close the search dialog box, and then close the file. Do not save any changes.

___ 4. Review the **JMSInput_Publication** message flow.



This message flow:

- Retrieves a message from a JMSInput node
- Removes the JMS header and adds a WebSphere MQ message descriptor (MQMD)
- Adds a JMS topic and a `publish` command to the MQRFH2 header in a Mapping node
- Sends the message assembly to a Publication node.

- ___ a. In the Application Development view, expand **Independent Resources > JMS Nodes Sample Flow Project > Flows**, and then double-click **JMSInput_Publication.msgflow**. The message flow opens in the Message Flow editor.
- ___ b. Right-click the **JMSInput** node, and then click **Properties**.

The node reads from which queue? _____

What queue do you see specified in the JMSInput node that you do not see specified in the MQInput node in the JMSGateway message flow? (Hint: this queue is specified in the **JMS Connection** tab.)

- ___ c. Open the **Mapping** node. How does the JMSInput_Publication_Mapping message map change the publication topic to **update/stock**?
- _____

___ 5. Review the stand-alone Java subscriber application.

The sample includes a stand-alone Java application that acts as a subscriber. It connects to the queue and subscribes to the topic. When the application receives a message, it writes its contents on the console.

- ___ a. In the Application Development view, expand **Independent Resources > JMS Nodes Sample Java Project > src > jmsnode**, and then double-click **JMSSubscriber.java**. The application opens in the Java editor.

- ___ b. Scroll to the `JMSSubscriber.main` method and review it.

```
public static void main(String[] args) {
    JMSSubscriber sub = new JMSSubscriber("update/stock");
    sub.start();
}
```

The main creates a subscriber on the topic **update/stock**.

- ___ c. Scroll to the **JMSSubscriber.start** method and review it.

```
public void start() {
    try {
        ippsFactory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
        ippsFactory.setQueueManager("IB9QMGR"); // TODO Adjust if no
        ippsFactory.setHostName("localhost");
        ippsFactory.setPort(2414); // TODO Adjust if no
        ippsFactory.setBrokerVersion(JMSC.MQJMS_BROKER_V2);
        ippsFactory.setSubscriptionStore(JMSC.MQJMS_SUBSTORE_BROKER);

        ipConn = ippsFactory.createTopicConnection();
        ipConn.start();

        subscribe();

    } catch (JMSException e1) {
        System.out.println("JMS Subscriber:\tError creating IP Topic C
        e1.printStackTrace();
    }
}
```

In particular, note the connection factory attributes that the method sets, including the queue manager name, the host name, and the connection port.



Note

The `com.ibm.mq.jms.JMSC` interface is deprecated. Use the constants that are defined in the `com.ibm.msg.client.wmq.WMQConstants` interface instead.

For more information, see www.ibm.com/support/docview.wss?uid=swg21423244.

- ___ 6. Replace all references to the `JMSC` interface with the `WMQConstants` interface.

- ___ a. Add an import statement for the **WMQConstants** interface:

```
import com.ibm.msg.client.wmq.WMQConstants;
```

- ___ b. Delete the import statement for the `com.ibm.mq.jms.JMSC` interface.

```
import com.ibm.msg.client.wmq.WMQConstants;
import com.ibm.mq.jms.MQTopicConnectionFactory;
```

- ___ c. In the **Start** method, replace the constant `JMSC.MQJMS_TP_CLIENT_MQ_TCPIP` with the equivalent value from `WMQConstants`:

```
ippsFactory.setTransportType( WMQConstants.WMQ_CM_CLIENT );
```

- ___ d. Replace the constant `JMSC.MQJMS_BROKER_V2` to the equivalent value from `WMQConstants`:

```
ippsFactory.setBrokerVersion( WMQConstants.WMQ_BROKER_V2 );
```

- ___ e. Replace the constant `JMSC.MQJMS_SUBSTORE_BROKER` with the equivalent value from `WMQConstants`:

```
ippsFactory.setSubscriptionStore(
    WMQConstants.WMQ_SUBSTORE_BROKER );
```

- ___ f. Delete the `@SuppressWarnings("Deprecation")` annotation from the **Start** Java method.

```
public void start() {
    try {
        ippsFactory.setTransportType (WMQConstants.WMQ_CM_CLIENT);
        ippsFactory.setQueueManager ("IB9QMGR"); // TODO Adjust if not using
        ippsFactory.setHostName ("localhost");
        ippsFactory.setPort (2414); // TODO Adjust if not using
        ippsFactory.setBrokerVersion (WMQConstants.WMQ_BROKER_V2);
        ippsFactory.setSubscriptionStore (WMQConstants.WMQ_SUBSTORE_BROKER);

        ipConn = ippsFactory.createTopicConnection();
        ipConn.start();

        subscribe();
    }
}
```

- ___ g. Save the changes in the editor.
- ___ h. Confirm that there are no warnings or errors in the **Problems** view.



Note

You can also copy the changes from a corrected version of the `JMSSubscriber.java` class:
C:\Labs\Lab07-JMS\Solutions\JMSSubscriber.java

- ___ 7. Review the **subscribe** method in the **JMSSubscriber** Java class.
- ___ a. Scroll to the **JMSSubscriber.subscribe** method and review it. It performs two main functions:

```
try {
    TopicSession topicsession = ipConn.createTopicSession(false, Session.AUT
    Topic topic = topicsession.createTopic(ourtopic);
    System.out.println("JMS Subscriber:\tSubscribing on topic: " + ourtopic);
    TopicSubscriber topicsubscriber = topicsession.createSubscriber(topic);
    Message message = null;

    try {
        message = (Message)topicsubscriber.receive();
    } catch (MessageEOFException eof) {}

    if (message instanceof TextMessage)
    {
        System.out.println("JMS Subscriber:\tReceived Text Message");
        TextMessage textmessage = (TextMessage)message;
        System.out.println("JMS Subscriber:\tMessage Text = " + textmessage.get
        System.out.println("JMS Subscriber:\tReceived from Message Destination
    } else
    if (message == null)
        System.out.println("JMS Subscriber:\tERROR: No Message received");
```

- First, the application attempts to subscribe on the topic. If successful, it writes a message to the console.
- Next, the application attempts to accept a message from the publisher. If it is successful, the application determines the format of the message: byte format or text format. In the example that you test, a text format message is received.
- The message is echoed to the console.

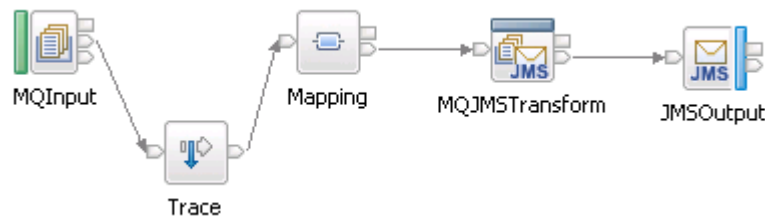
- ___ 8. Close the Java editor.

Part 4: Testing the sample

Instead of using the sample instructions to run the sample, you modify the message flows by adding trace nodes. This process allows you to view the changed message assembly as it appears after each node runs.

- ___ 1. Modify the JMSGateway message flow.
- ___ a. In the **Application Development** view, double-click the **JMSGateway.msgflow** message flow to open it in the editor, if it is not already open.
- ___ b. Delete the wire between the **MQInput** node and the **Mapping** node.
- ___ c. From the **Construction** drawer, add a **Trace** node to the canvas between the **MQInput** and **Mapping** nodes.

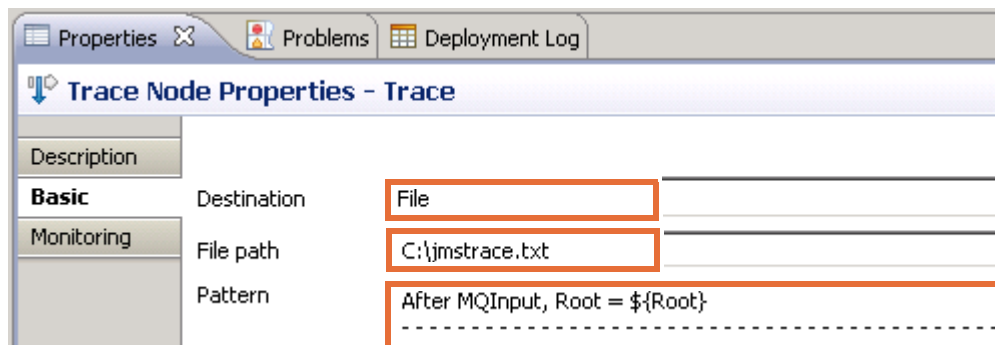
- ___ d. Wire the **Out** terminal of the **MQInput** node to the **In** terminal of the **Trace** node.
- ___ e. Wire the **Out** terminal of the **Trace** node to the **In** terminal of the **Mapping** node.



- ___ f. Configure the **Trace** node properties. On the **Basic** tab:

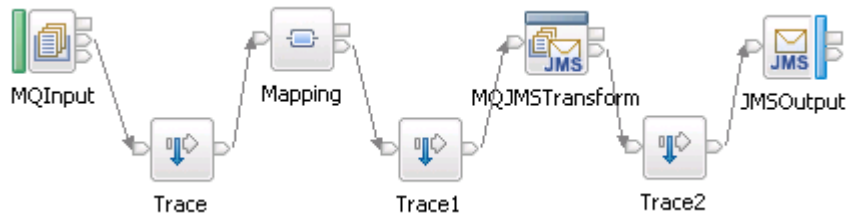
- Set the **Destination** to **File**.
- For **File path**, enter: C:\jmstrace.txt
- For **Pattern**, enter:
After MQInput, Root = \${Root}

The line of dashes is to help separate the output in the text file. It does not matter how many you use. You can use a different separator character if you want.

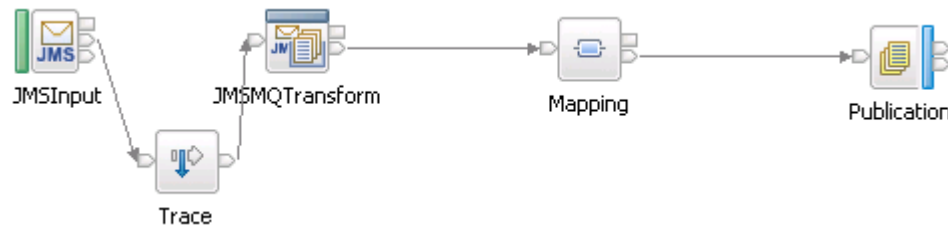


- ___ g. Repeat steps b-f for the **Mapping** and **MQJMSTransform** nodes.
- Be sure to use the **Out** terminal from the **Mapping** node.
 - Remember to change the **Pattern** value to "After Mapping".
- ___ h. Repeat steps b-f for the **MQJMSTransform** and **JMSOutput** nodes. Remember to change the **Pattern** value to "After MQJMSTransform".

- ___ i. Save the workspace (Ctrl + S). The updated message flow should be similar to the one shown here.

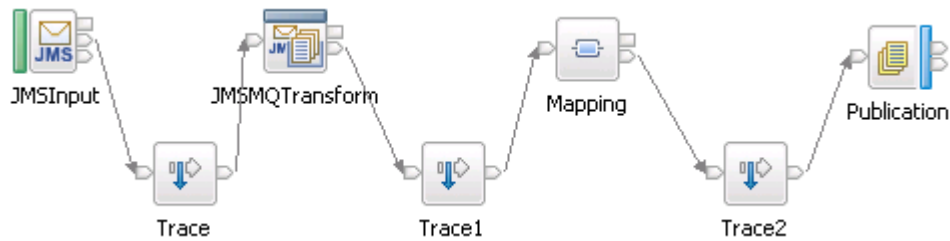


- ___ 2. Modify the JMSInput publication message flow.
- In the Application Development view, double-click the **JMSInput_Publication.msgflow** message flow to open it in the editor, if it is not already open.
 - Delete the wire between the **JMSInput** node and the **JMSMQTransform** node.
 - From the **Construction** drawer, add a **Trace** node to the canvas between the **JMSInput** and **JMSMQTransform** nodes.
 - Wire the **Out** terminal of the **JMSInput** node to the **In** terminal of the **Trace** node.
 - Wire the **Out** terminal of the **Trace** node to the **In** terminal of the **JMSMQTransform** node.

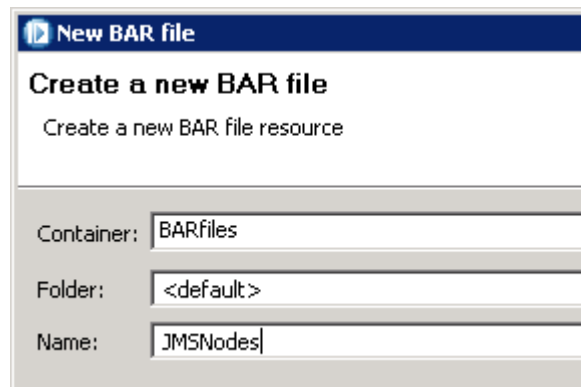


- Configure the **Trace** node properties. On the **Basic** tab:
 - Set the **Destination** to **File**.
 - For **File path**, enter: C:\JMStrace.txt
 - For **Pattern**, enter:
After JMSInput, Root = \${Root}
- Repeat steps b-f for the **JMSMQTransform** and **Mapping** nodes. Remember to change the **Pattern** value to "After JMSMQTransform".
- Repeat steps b-f for the **Mapping** and **Publication** nodes.
 - Be sure to use the **Out** terminal from the **Mapping** node.
 - Remember to change the **Pattern** value to "After Mapping".

- ___ i. Save the workspace (Ctrl + S). The updated message flow should be similar to the one shown here.

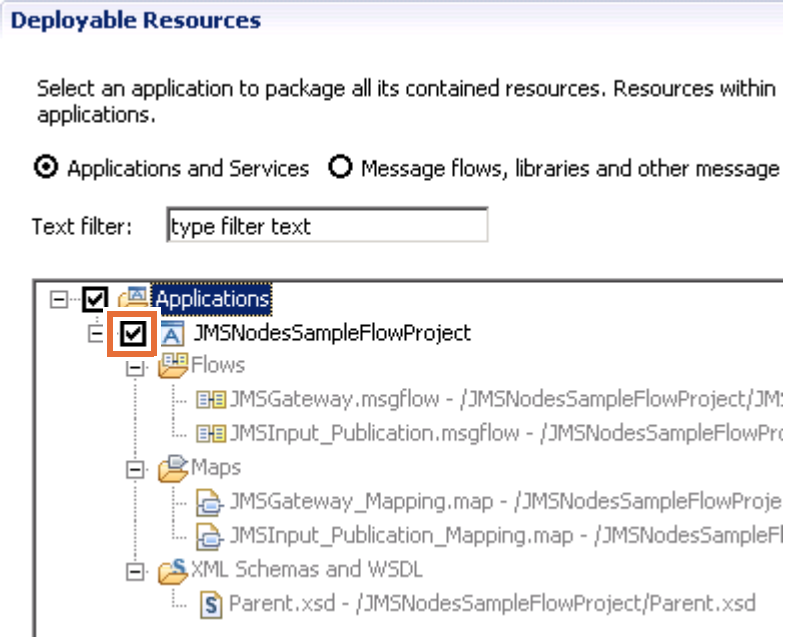


- ___ 3. Create the Broker archive (BAR) file.
- ___ a. In the Application Development view, right-click **JMS Nodes Sample Flow Project**, and then click **New > BAR file**.
- ___ b. For **Name**, enter: JMSNodes

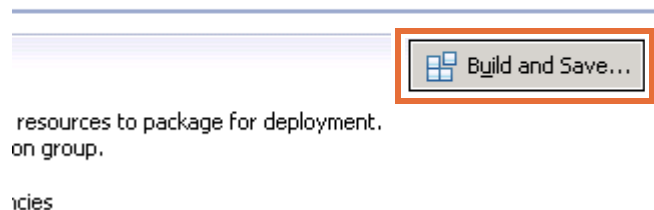


- ___ c. Click **Finish**. The BAR file editor opens.

- ___ d. Select the **JMSNodesSampleFlowProject** check box.



- ___ e. In the upper-right side of the BAR file editor, click **Build and Save**.



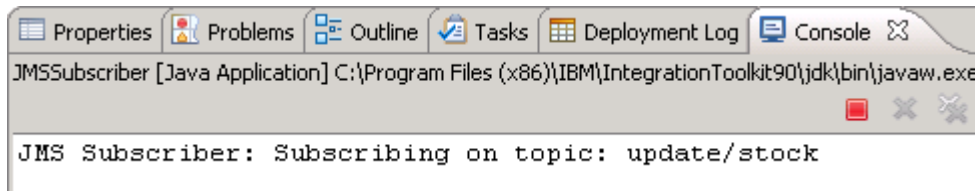
- ___ f. The BAR file is built. After the **Operation completed successfully** message is displayed, click **OK**.
- ___ 4. Deploy the components to the runtime environment.
- ___ a. In the Application Development view, under **BARs**, right-click **JMSNodes.bar** > **BARfiles/JMSNodes.bar**, and then click **Deploy**. The **Deploy** dialog box is displayed.
- ___ b. Select the **default** integration server, and then click **Finish**. The BAR file is deployed.



Note

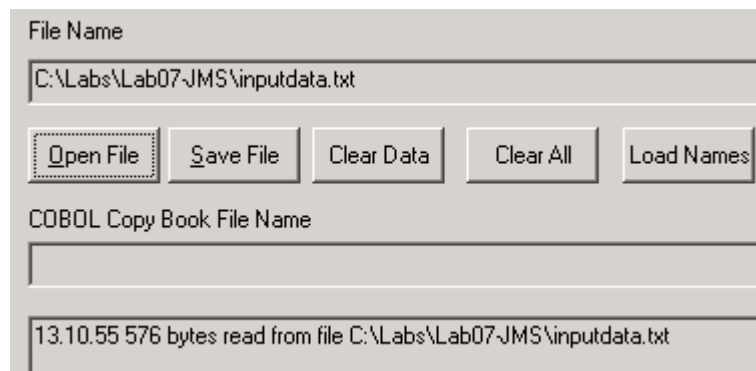
You can also deploy the BAR file by dragging it from the Application Development view to the **default** integration server in the **Integration Nodes** view.

- ___ 5. Start the stand-alone Java subscriber application.
 - ___ a. In the Application Development view, expand **Independent Resources > JMS Nodes Sample Java Project > src > jmsnode**, if it is not already expanded.
 - ___ b. Right-click `JMSSubscriber.java`, and then click **run as > Java Application**.
 - ___ c. Wait until the **Console** view is displayed. The application writes the initial subscription message to the console, and then waits for a message to appear on the subscription queue.



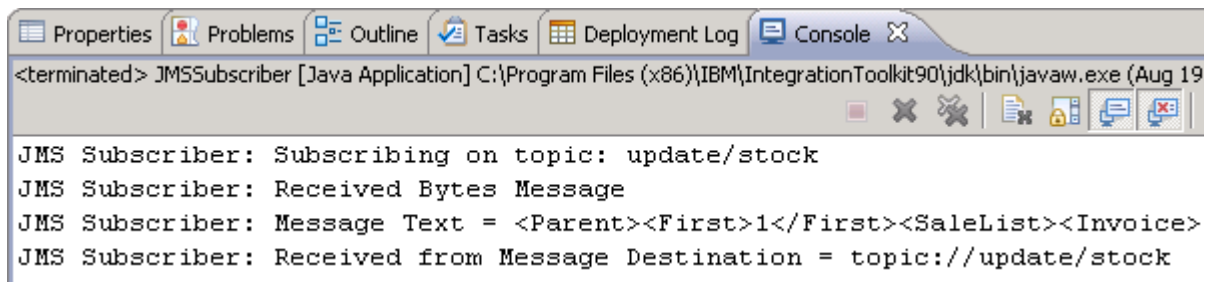
- ___ 6. Run the message flows.
 - ___ a. Start RFHUtil by double-clicking the **RFHUtil** icon on the Windows desktop.
 - ___ b. For **Queue Manager Name**, select **IB9QMGR**.
 - ___ c. For **Queue Name**, select **INPUTQUEUE**.
 - ___ d. Click **Open File**.
 - ___ e. Go to `C:\Labs\Lab07-JMS\inputdata.txt`, and then click **Open**.

A message is displayed, indicating that data was read from the file.



- ___ f. Click **Write Q**. The message data is written to queue **INPUTQUEUE**.
The JMSGateway message flow runs, and then the JMSInput_Publication message flow runs. Finally, the stand-alone Java application runs and uses the published message.

- ___ 7. Review the results on the runtime console.
- ___ a. Switch to the IBM Integration Toolkit, and review the **Console** window.



```

<terminated> JMSSubscriber [Java Application] C:\Program Files (x86)\IBM\IntegrationToolkit90\jdk\bin\javaw.exe (Aug 19
JMS Subscriber: Subscribing on topic: update/stock
JMS Subscriber: Received Bytes Message
JMS Subscriber: Message Text = <Parent><First>1</First><SaleList><Invoice>
JMS Subscriber: Received from Message Destination = topic://update/stock

```

The stand-alone Java subscriber wrote several console messages. You should see that it:

- Subscribed on the **update/invoices** topic
 - Received a message
 - Displayed the message. You see that the message includes the time stamp that the **Compute** node in the JMSGateway message flow added. (Scroll to the right to see the time stamp value.)
 - Received the message from the **PUBLISHQUEUE** queue.
- ___ 8. Review the trace results from the Trace nodes that you embedded in the message flows.
- ___ a. Start a Windows Explorer session.
- ___ b. Go to C:\jmstrace.txt, and then double-click the file. The file opens in Notepad.
- ___ c. The first message assembly that is displayed in the file is the message as it was received from the **MQInput** node in the JMSGateway message flow. Note the following items:
- In the MQMD, the application that put the message to the queue is shown, along with the date and time. The user ID that is associated with the application that put the message to the queue is also displayed.

```

:UserIdentifier      = 'IIBAdmin      ' (CHARACTER)
:AccountingToken     = X'16010515000000921ece9ba5532441065163b6550400
:AppIdentityData     = '              ' (CHARACTER)
:PutApplType         = 11 (INTEGER)
:PutAppName          = 'C:\Labs\Tools\rfhutil.exe' (CHARACTER)
:PutDate             = DATE '2013-08-19' (DATE)

```

- The Timestamp is not populated in the message assembly yet.

```
(0x01000000:Folder):XMLNSC      = ( ['xmlns' : 0x1730c570]
(0x01000000:Folder):Parent = (
  (0x03000000:PCDataField):First      = '1' (CHARACTER)
  (0x01000000:Folder):SaleList = (
    (0x01000000:Folder):Invoice = (
      (0x01000000:Folder):Timestamp =
      (0x03000000:PCDataField):Initial = 'I' (CHARACTER)
      (0x03000000:PCDataField):Initial = 'D' (CHARACTER)
      (0x03000000:PCDataField):Surname = 'Montana' (CHARACTER)
    (0x01000000:Folder):Item = (
```

- ___ d. The next message assembly that is displayed is the result of the **Mapping** node in the JMSGateway message flow. You see that the Timestamp element was inserted into the message assembly.

```
:Invoice = (
aField):Timestamp = '2013-08-19T13:30:44-07:00' (CHARACTER)
aField):Initial = 'I' (CHARACTER)
aField):Initial = 'D' (CHARACTER)
aField):Surname = 'Montana' (CHARACTER)
```

- ___ e. The last trace node in the JMSGateway message flow shows the resulting message assembly after the MQJMSTransform node was run. What changed in the message because this node ran?
-

- ___ f. The first trace node in the JMSInput_Publication message flow follows the JMSInput node. Did the values of the **JMSXUserID** and **JMSXAppID** (the identities of the user ID and application that created the message) change between the JMSGateway flow and the JMSInput_Publication flow?

Review the trace output from the **JMSMQTransform** node. The MQMD.Format field is set to **MQHRF2**. What does this value indicate?

Review the trace output from the final **Mapping** node. The node set the **Topic** field in preparation for sending the message to the **Publication** node.

```

-----
After Mapping, Root = ( ['MQROOT' : 0x2c326bd0]
  (0x01000000:Name ):Properties = ( ['MQPROPERTYPARSER' : 0x2c358310]
    (0x03000000:NameValue):MessageSet           = '' (CHARACTER)
    (0x03000000:NameValue):MessageType          = '' (CHARACTER)
    (0x03000000:NameValue):MessageFormat        = '' (CHARACTER)
    (0x03000000:NameValue):Encoding             = 0 (INTEGER)
    (0x03000000:NameValue):CodedCharsetId      = 1208 (INTEGER)
    (0x03000000:NameValue):Transactional        = FALSE (BOOLEAN)
    (0x03000000:NameValue):Persistence          = FALSE (BOOLEAN)
    (0x03000000:NameValue):CreationTime         = GMTTIMESTAMP '2013-08-19 2
    (0x03000000:NameValue):ExpirationTime       = NULL
    (0x03000000:NameValue):Priority              = 0 (INTEGER)
    (0x03000000:NameValue):ReplyIdentifier      = \'\'' (CHARACTER)
    (0x03000000:NameValue):ReplyProtocol        = 'JMS' (CHARACTER)
    (0x03000000:NameValue):Topic                = 'update/stock' (CHARACTER)
    (0x03000000:NameValue):ContentType         = '' (CHARACTER)
  )
)

```

Part 5: Clean up

- ___ 1. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, if you are prompted.
- ___ 2. In the **Integration Nodes** view, right-click the **default** integration server, and then click **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.
- ___ 3. Close RFHUtil.
- ___ 4. Close the Information Center window.

End of exercise

Exercise 8. 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

At the end of 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 facilities, you can authenticate and authorize each individual message when it is presented for processing in a message flow. Additionally, you can 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. In this exercise, because you do not have an

external security provider to work with, you work with the message transport nodes and propagating the message through the message flow. You also simulate the credential mapping operation of an external security provider by using a Compute node.

Requirements

- A workstation with WebSphere MQ and IBM Integration Bus components installed
- The IBM Integration Bus default configuration
- The WebSphere MQ local queues: SECURITYIDFROMMQIN, SECURITYIDFROMMQOUT, SECURITYIDFROMMSGIN, SECURITYIDFROMMSGOUT

Exercise instructions

Part 1: Start components and prepare the workspace

- ___ 1. Make sure that all local IBM Integration Bus components are running.
- ___ 2. Start the IBM Integration Toolkit and open a new workspace that is named C:\Workspace\security.
- ___ 3. When the **Welcome** window is displayed, close it. The Application Development perspective is displayed.
- ___ 4. Import the starting projects from the project interchange file.
 - ___ a. From the menu bar, click **File > Import**.
 - ___ b. Expand **Other**, click **Project Interchange**, and then click **Next**.
 - ___ c. To the right of **From zip file**, click **Browse**.
 - ___ d. Browse to C:\Labs\Lab08-Security\SecurityImportPI.zip, and then click **Open**. The **Import Projects** dialog box is displayed.
 - ___ e. Ensure that all projects are selected, and then click **Finish**.

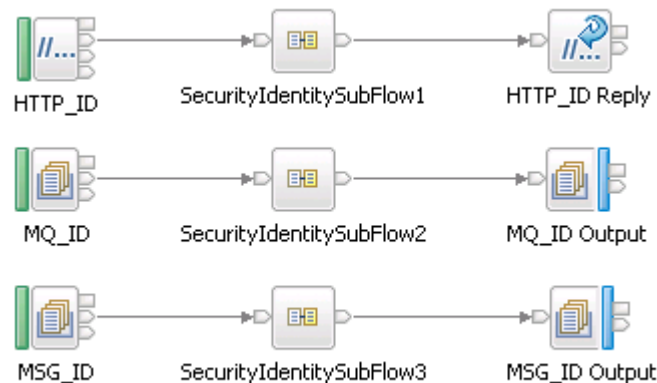
The projects are imported. Wait for the workspace build to complete.

Part 2: Reviewing the message flows

There are three message flows in this exercise. You review each of them in this part of the exercise.

- ___ 1. In the Application Development view, expand **Independent Resources > SecurityIdentitySampleFlowProject > Flows**.
- ___ 2. Double-click **SecurityIdentitySampleFlow.msgflow** to open it in the Message Flow editor.

There are three flows in this message flow file. Each flow does similar processing; the only difference is in the transport protocols that are used.



- The first 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 second flow receives a message from MQInput node **MQ_ID**, calls the SecurityIdentitySubflow, and then writes the resulting message assembly to an MQOutput node.
- The third flow receives a message from MQInput node **MSG_ID**, calls the SecurityIdentitySubflow, and then writes the resulting message assembly to an MQOutput node. This flow is similar to the second flow, except for the security parameters that are specified on the MQInput 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?
-

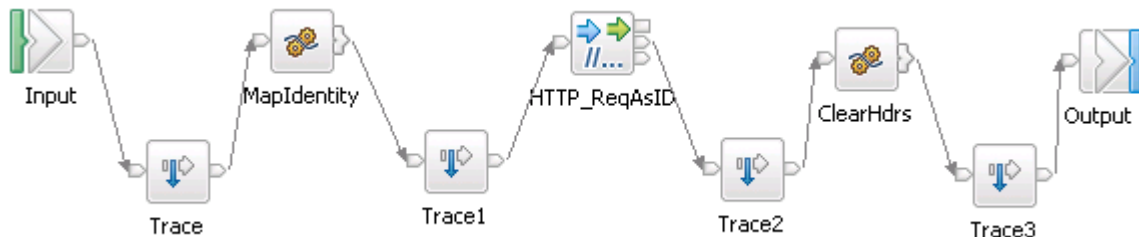
___ e. **Question 7:** On the **Security** tab, what is the **Identity token type** value?

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.



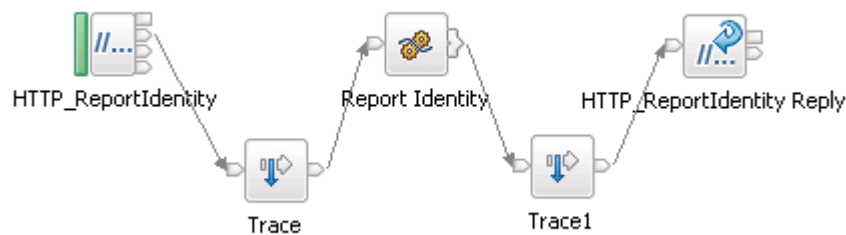
The subflow does most of the work in this exercise. It contains a **MapIdentity** Compute node, an **HTTP_ReqAsID** HTTP Request node, and a **ClearHdrs** Compute node. Between each of these nodes, and following the **ClearHdrs** node, are **Trace** nodes.

___ a. Double-click the **MapIdentity** node. This action opens the ESQL editor.

- **Question 10:** Explain briefly what this 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.

- **Question 12:** Review the properties of the **HTTP_ReportIdentity** node. On the **Security** tab, what is the **Identity token type** value?

- ___ a. Double-click the **ReportIdentity** node. This action opens the ESQL editor.

Question 13: Explain briefly what this node does.

Part 3: Testing the message flows

In this part of the exercise, you test the message flows and analyze the results.

- ___ 1. Deploy the components to the integration server.
- ___ 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 **default** integration server 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.

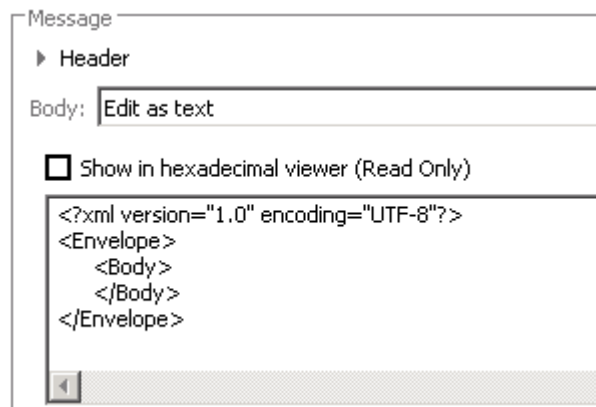
In this exercise, IBM Integration Bus Test Client test files are already created for you. You can find them in the Application Development view, under **Independent Resources > SecurityIdentitySampleFlowProject > Flow Tests**.

Test 1: Testing with a WebSphere MQ message that contains security information in the MQMD header

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 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 it contains an envelope and an empty message body.



- ___ c. Click the **Configuration** tab at the bottom of the 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). 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.

▼ MQ Message Header "Send Identity"

Accounting token:		Message type:	8
Application origin data:		Message sequence number:	1
Application id data:		Offset:	0
Backout count:	0	Original length:	-1
Character set:	0	Persistence:	2
Correlation id:		Priority:	-1
Encoding:	0	Put application name:	TestClient
Expiry:	-1	Put application type:	0
Feedback:	0	Put date/time:	
Format:		Report:	0
Group id:		Reply to queue manager name:	
Message id:		Reply to queue name:	
Message flags:	0	User id:	mqmdUID

☐ Include RFH V2 header

- ___ 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 WebSphere 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

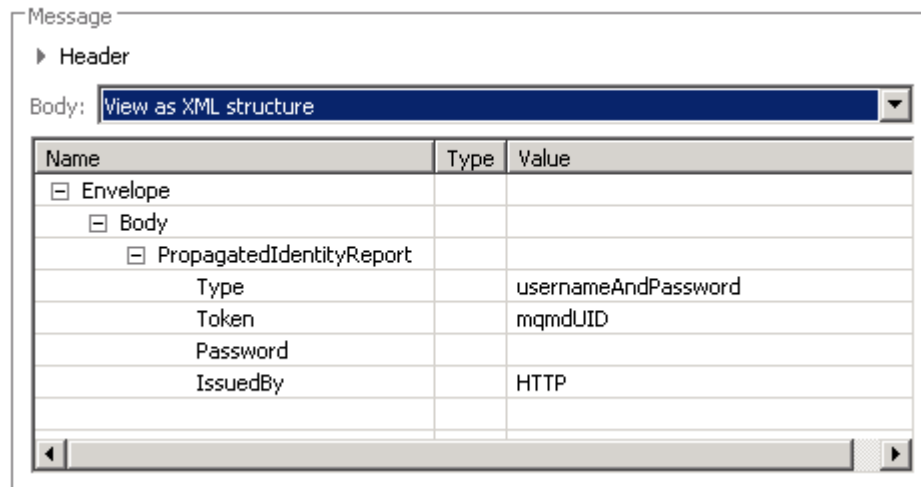
Use the output from the **Trace** nodes to verify the contents of the message assembly as the message flow was running. The Trace nodes write their output to C:\securitytrace.txt. You can verify the output destination by reviewing the properties for any of the Trace nodes.

- **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 WebSphere MQ by the **MQ_ID Output** node in the main flow. Review the node properties to determine the queue that was used.

You can use RFHUtil to browse the queue, or you can return to the Test Client and click **Dequeue** in the **Message Flow Test Events** section, and then click **Get Message** under the **Detailed Properties**.



Be sure to change the **Queue manager** to IB9QMGR.

- 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 Test Client window. Do not save changes if 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 **default** integration server, and then click **Stop**. Wait for the integration server to end. The green up-arrow turns into a red down-arrow when the integration server is stopped.

- ___ 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 **default** integration server by right-clicking it and then clicking **Start** in the **Integration Nodes** view. Wait for the integration server to start. The red down-arrow turns into a green up-arrow when the integration server is restarted.

Test 2: Testing with a WebSphere MQ message that contains security information in the message body

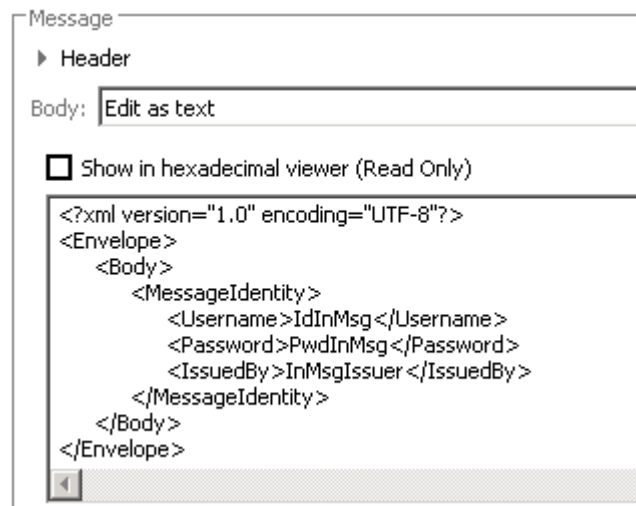
In this test, you submit a message that contains security credentials within the message itself, instead of in the transport header. WebSphere MQ does not provide a location to include a password within the MQMD. Passing the user ID and password credentials as part of the message body, rather than just 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 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.

▼ **MQ Message Header "Default Header"**

Accounting token:		Message type:	8
Application origin data:		Message sequence number:	1
Application id data:		Offset:	0
Backout count:	0	Original length:	-1
Character set:	0	Persistence:	2
Correlation id:		Priority:	-1
Encoding:	0	Put application name:	
Expiry:	-1	Put application type:	0
Feedback:	0	Put date/time:	
Format:		Report:	0
Group id:		Reply to queue manager name:	
Message id:		Reply to queue name:	
Message flags:	0	User id:	

☐ Include RFH V2 header

- ___ 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. Again, you can verify that the message was enqueued on the **SECURITYIDFROMMSGOUT** queue by using WebSphere 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 WebSphere MQ by the **MSG_ID Output** node in the main flow.

You can use RFHUtil to browse the queue, or you can return to the Test Client and click **Dequeue** in the **Message Flow Test Events** section, and then click **Get Message** under the **Detailed Properties**.

Message		
▶ Header		
Body: View as XML structure		
Name	Type	Value
Envelope		
Body		
PropagatedIdentityReport		
Type		usernameAndPassword
Token		IdInMsg
Password		PwdInMsg
IssuedBy		HTTP

- Did you expect this result? _____

___ b. Again, 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 if the HTTP Input node also changed it (although it did).

- ___ 4. Close the Test Client window. Do not save changes if 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: Testing with a WebSphere MQ message where mapped credentials are required

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 a WebSphere 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.

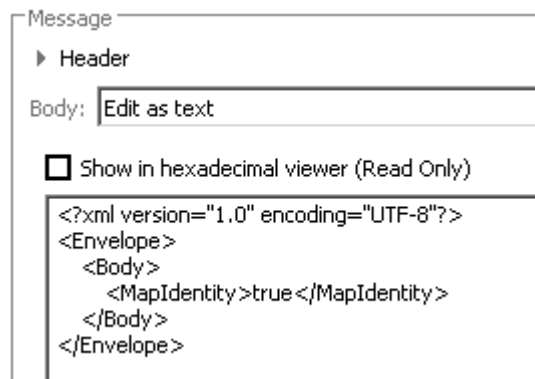
As you saw in the previous two tests, the MQMD provides a location for a user ID only, not a password. Therefore, mapping is often used in message flows that use WebSphere MQ as the message transport, by using the user ID and “Issued by” values. In this test, the message flow simulates credential mapping by using the **MapIdentity** Compute node in the `SecurityIdentitySubflow`, instead of using an external security provider.

In this test, a WebSphere 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 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 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)**. Observe that the **Put application name** and **User ID** elements of the MQMD are again populated.

▼ **MQ Message Header "Send Identity"**

Accounting token:		Message type:	8
Application origin data:		Message sequence number:	1
Application id data:		Offset:	0
Backout count:	0	Original length:	-1
Character set:	0	Persistence:	2
Correlation id:		Priority:	-1
Encoding:	0	Put application name:	BRKTSTCLNT
Expiry:	-1	Put application type:	0
Feedback:	0	Put date/time:	
Format:		Report:	0
Group id:		Reply to queue manager name:	
Message id:		Reply to queue name:	
Message flags:	0	User id:	TESTUSER

☐ Include RFH V2 header

- ___ 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. Again, you can verify that the message was enqueued on the **SECURITYIDFROMQOUT** queue by using WebSphere 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 WebSphere MQ by the **MQ_ID Output** node in the main flow.

Hint: The message was written to the **SECURITYIDFROMMQOUT** queue.

In the Test Client, click **Dequeue** in the **Message Flow Test Events** section, and then click **Get Message** under the **Detailed Properties**.

Message

► Header

Body: View as XML structure

Name	Type	Value
Envelope		
Body		
MapIdentity		true
PropagatedIdentityReport		
Type		usernameAndPassword
Token		testuser@company.com
Password		p_testuser2012
IssuedBy		HTTP

Did you expect this result? _____

- ___ 4. Close the Test Client window. Do not save changes if you are prompted to do so.
- ___ 5. Delete or rename the trace output file.

Part 4: Clean up

- ___ 1. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, if you are prompted.
- ___ 2. In the **Integration Nodes** view, right-click the **default** integration server, and then click **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.

Answers to exercise questions

- 1. /Security/IdentityFromHttp
- 1. .
- 2. Transport Default
- 3. .
- 4. SECURITYIDFROMMQIN
- 5. .
- 6. Transport Default
- 7. .
- 8. SECURITYIDFROMMQOUT.
- 9. .
- 10. SECURITYIDFROMMSGIN
- 11. .
- 12. Username + Password.
- 13. .
- 14. 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.
- 15. .
- 16. SECURITYIDFROMMSGOUT
- 17. .
- 18. 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.
- 19. .

20. 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.
- 21.
22. Transport Default
- 23..
24. 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`.
25. 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`.
26. When the message reached the **HTTP_ReqAsID** HTTP request node in the **SecurityIdentitySubflow**, it started the **SecurityIdentityReportFlow** (based on the **Web service URL** property).

Web service URL*

`http://localhost:7080/Security/Identity/ReportIdentity`

Examine the properties of the **HTTP_ReportIdentity** HTTP Input node in the **SecurityIdentityReportFlow**, specifically the **Security** tab.

Identity token type	Transport Default
Identity token location	
Identity password location	
Identity issuedBy location	
Treat security exceptions as normal exceptions	<input type="checkbox"/>

- 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.

27. As in the first test, it did not generate values because no `InputRoot.XMLNSC.Envelope.Body.MapIdentity` element was found. The only action that the `MapIdentity` node did, was to copy the `InputRoot` root to the `OutputRoot`.

28. 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.

29. 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.

End of exercise

Exercise 9. 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 capture and report failed events.

What you should be able to do

At the end of this exercise, you should be able to:

- Define monitoring events
- Activate flow monitoring
- View event messages in the IBM Integration web console
- Replay messages
- Capture and report failed events

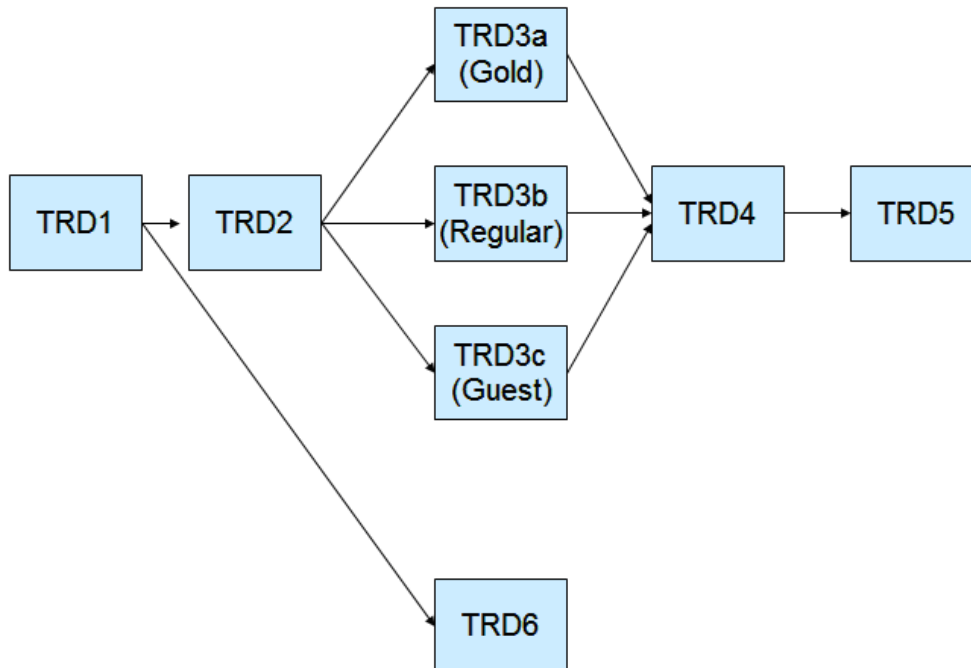
Introduction

This lab uses an application that contains several message flows.

The application processes stock trade requests, and each trade runs five message flows: TRD1, TRD2, TRD3*, TRD4, and TRD5. The TRD3 message flow is run by customer type (“Gold”, “Regular”, or “Guest”).

Several of the nodes of the message flows have monitoring points that are defined on them, using the **Monitoring** tab on the node properties. These monitoring points publish certain items of the message payload data. The **Data viewer** in the IBM Integration web console uses this data to access messages that were processed, and to resubmit (replay) the message for reprocessing.

The application contains the following message flow structure. The TRD3 flows are selected based on the customer type. The TRD6 flow is run if a validation failure occurs in TRD1.

**Note**

In this lab exercise, monitoring points are configured on the message flow by using the **Monitoring** function on the flow and node properties. Flow monitoring can also be achieved noninvasively by using monitoring templates, which are not covered in this lab.

Requirements

- WebSphere MQ and IBM Integration Bus components installed
- The IBM Integration Bus default configuration
- Lab files in the C:\Labs\Lab09-RandR
- The following WebSphere MQ 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

Exercise instructions

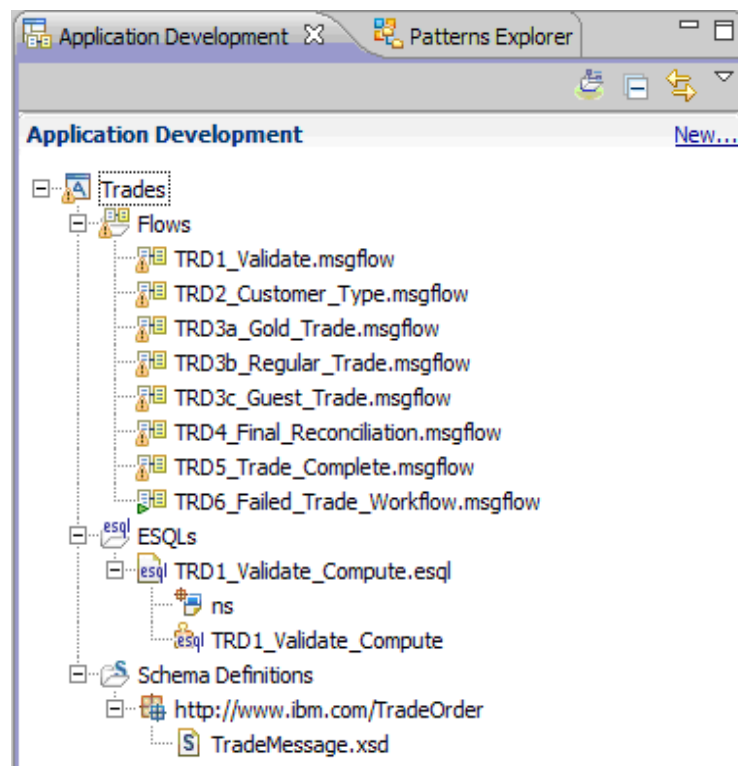
Part 1: Set up the application for record and replay

- ___ 1. If not open, start the IBM Integration Toolkit and create a new workspace such as C:\Workspaces\Record.
- ___ 2. Verify that the integration node IB9NODE is running.
- ___ 3. Import a Project Interchange file that is named Trades.zip from the C:\Labs\Lab09-RandR\application directory.

Ensure that the **Trades** application is selected, and then click **Finish**.

The **Trades** application is imported.

- ___ 4. Expand the **Trades** application. It contains eight message flows. The message flows are run in sequence, with just one of the TRD3* messages flows being used, depending on the type of customer.



The imported message flows contained in the **Trades** application already have several monitoring event points defined. In the next steps, you review the flow monitoring event points in the message flows.

- ___ 5. Open the **TRD1_Validate** message flow, and click the Message Flow editor canvas (not on a node) to display the message flow properties.

- ___ 6. On the message flow **Properties** tab, click **Monitoring** to show the flow monitoring points in the flow, and their initial status.

The screenshot shows the IBM Integration Designer interface. The top pane displays the message flow 'TRD1_Validate.msgflow' with nodes: 'Receive Trade', 'Compute', 'Decide Customer Type', and 'Validation Failure'. The bottom pane shows the 'Properties' view for 'TRD1_Validate', with the 'Monitoring' tab selected. The 'Monitoring' tab displays a table of events defined for the message flow.

Enabled	Node	Event Source	Event Source Address	Event Name	Event Filter
<input checked="" type="checkbox"/>	Receive Trade	Transaction start	Receive Trade.transaction.Start	Trace received	true()
<input checked="" type="checkbox"/>	Validation Failure	In terminal	Validation Failure.terminal.in	Data validation failure	true()

- ___ 7. Click the MQInput node that is named **Receive Trade**.

The **Monitoring** properties now show just the events that are defined for the **Receive Trade** input node. Just a single event is defined, for the “Transaction start” monitoring point.

The screenshot shows the 'MQ Input Node Properties - Receive Trade' dialog box. The 'Monitoring' tab is selected, displaying a table of events defined for the 'Receive Trade' node. The 'Edit...' button is highlighted with a red box.

Enabled	Event Source	Event Source Address	Event Name	Event Filter
<input checked="" type="checkbox"/>	Transaction start	Receive Trade.transaction.Start	Trace received	true()

- ___ c. Click the **Transaction start** event, and then click **Edit**.
- ___ d. On the **Basic** tab for the “Transaction start” event:

- **Event Name** is set to a **Literal** string of Trade instruction received.
- **Event Filter** is used to dynamically determine whether to emit a monitoring event, which is based on the value of a message element. For this event, the **Event Filter** is set to `true()`, which emits all messages that it processes.

The screenshot shows the configuration console for an event source in the Transaction tab. The 'Event Source' dropdown is set to 'Transaction start'. The 'Event Source Address' field contains 'Receive Trade.transaction.Start'. The 'Event Name' section has the 'Literal' radio button selected, and the text field next to it contains 'Trade instruction received'. The 'Event Filter' section has a text field containing 'true()'. Red boxes highlight the 'Literal' radio button, the 'Trade instruction received' text, and the 'true()' text.

- ___ e. On the **Correlation** tab, the **Local transaction correlator** is set to `$Root/XMLNSC/tra:tradeOrder/tra:customerID`.

The **Parent transaction correlator** is set to `$Root/XMLNSC/tra:tradeOrder/tra:tradeOrderID`.

The **Global transaction correlator** is set to **Automatic** and left for values in later message flows.

If you want to change these values, click **Edit** to start the XPath editor, or, you can edit the correlation fields directly.

Event Correlation

A monitoring application uses event correlators to match events emitted by the same, or related, business transactions. A local transaction correlator links the events emitted by a single invocation of a message flow. A parent transaction correlator links the events from a message flow to a parent message flow or an external application. A global transaction correlator links events from a message flow to one or more related message flows or external applications. An event must contain a local transaction correlator, but need not contain a parent transaction correlator or global transaction correlator.

Local transaction correlator:

☐ Automatic ☒ Specify location of correlator

Description
The local correlator will be read from the specified location in the message tree. Ensure the specified location contains a correlator value unique to this message flow invocation.

Parent transaction correlator:

☐ Automatic ☒ Specify location of correlator

Description
The parent correlator will be read from the specified location in the message tree. Ensure the specified location contains a suitable parent correlator value.

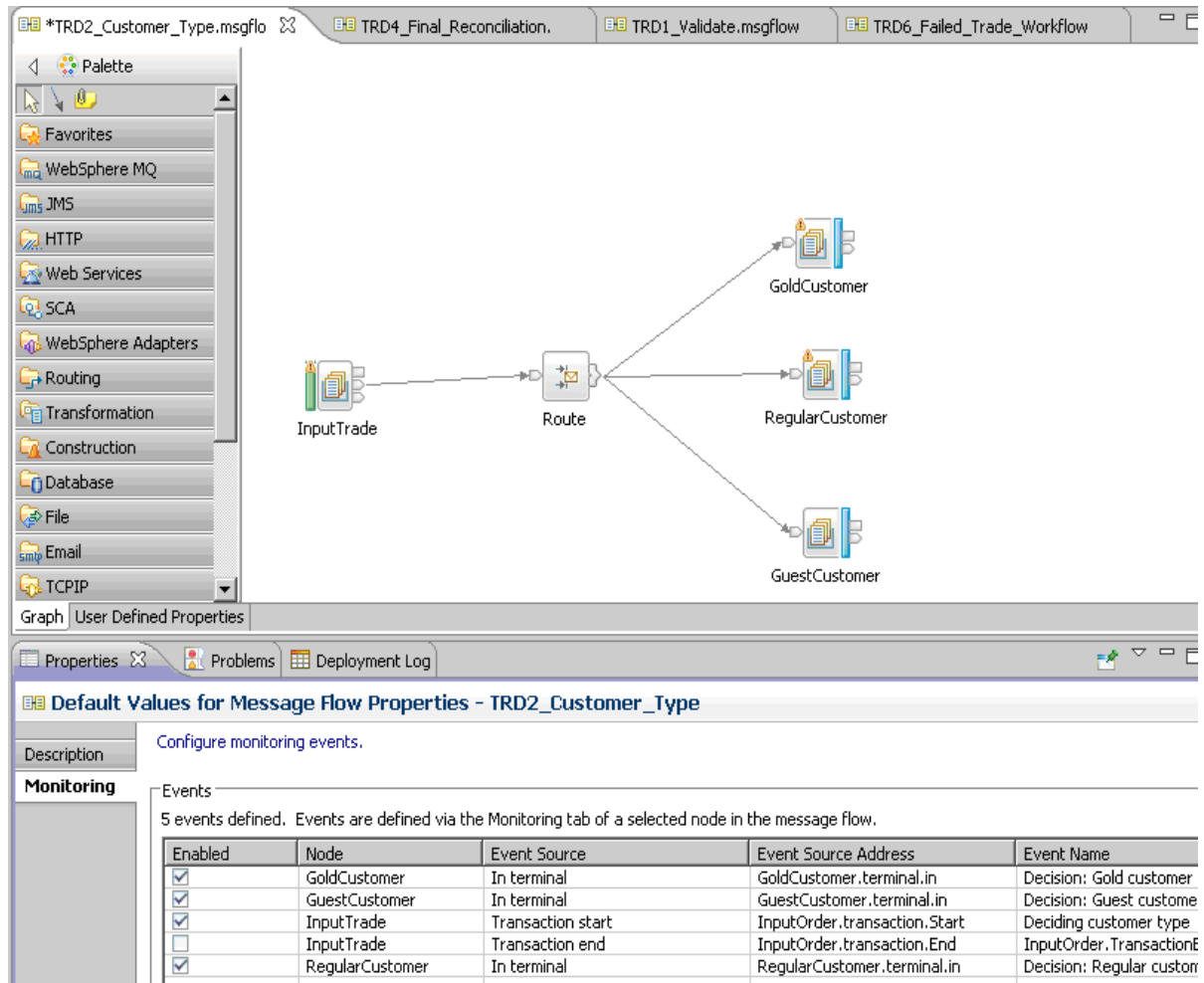
Global transaction correlator:

☒ Automatic ☐ Specify location of correlator

Description
The global correlator used by the most recent event for this message flow invocation will be used. If no correlator exists yet, no global correlator will be used.

- ___ f. Cancel the **Monitor** window for the “Transaction start” event.

- ___ 8. Open the **TRD2_Customer_Type** message flow, and review the monitoring configuration.



- ___ a. Open the “Transaction start” monitoring event for the **InputTrade** node.
- ___ b. Click the **Correlation** tab.

The **Global transaction correlator** is set to a specific location of `$Root/XMLNSC/tra.tradeOrder/StockAmount`, which is a new message tree element that the **TRD1_Validate** message flow creates. It is a combination of the trade and amount elements.

Event Correlation

A monitoring application uses event correlators to match events emitted by the same, or related, business transactions. A local transaction correlator links the events emitted by a single invocation of a message flow. A parent transaction correlator links the events from a message flow to a parent message flow or an external application. A global transaction correlator links events from a message flow to one or more related message flows or external applications. An event must contain a local transaction correlator, but need not contain a parent transaction correlator or global transaction correlator.

Local transaction correlator:

☐ Automatic ☒ Specify location of correlator

Description
The local correlator will be read from the specified location in the message tree. Ensure the specified location contains a correlator value unique to this message flow invocation.

Parent transaction correlator:

☐ Automatic ☒ Specify location of correlator

Description
The parent correlator will be read from the specified location in the message tree. Ensure the specified location contains a suitable parent correlator value.

Global transaction correlator:

☐ Automatic ☒ Specify location of correlator

Description
The global correlator will be read from the specified location in the message tree. Ensure the specified location contains a suitable global correlator value.

- ___ c. Click **Cancel** to close the “Transaction start” monitoring event window.
- ___ 9. Verify that the monitoring event is configured to emit the message payload.
 The Replay function uses the raw message payload (bitstream) that an event monitoring point can emit. If you do not plan to use the “replay” function, do not configure the monitoring event to emit the bitstream.
 In this step, you verify that the monitoring event is configured to emit the bitstream on the node that processes the Validation failures.
- ___ a. If it is not already open, open the **TRD1_Validate** message flow.
- ___ b. Select the **Validation Failure** node.
- ___ c. In the **Validation Failure** node properties, select the **Monitoring** option.
- ___ d. Select the “In terminal” event source, and then click **Edit** to open the Monitoring Event editor for this node.
- ___ e. Under the **Event Payload** section on the **Basic** tab, verify that the option to **Include bitstream data in payload** is selected.

Edit event

Basic Correlation Transaction

Event Source
Select the source of the event.
In terminal

Event Source Address
The broker identifies an event source using an event source address. Use this value when you enable and disable event sources using runtime commands.
Validation Failure.terminal.in

Event Name
Provide the name by which events emitted from this source are to be known. Specify either a literal name, or the location of a character field in the message tree or elsewhere in the message assembly.
☒ Literal Data validation failure
☐ Data location Edit...

Event Filter
Provide an expression to control whether the event is emitted. The expression must evaluate to true or false, and can reference fields in the message tree or elsewhere in the message assembly. If you do not specify a value, the value true() is used.
true() Edit...

Event Payload
Most events need to contain data taken from fields in the message tree or from elsewhere in the message assembly. Data taken from simple fields or complex fields appears in the event in XML character format. An event can also contain bitstream data, which appears in the event as hexadecimal bytes.

Data location	
\$ExceptionList	

Add... Edit... Delete

☒ Include bitstream data in payload

Content All Encoding base64Binary

- ___ f. Click **Cancel** to close the **Edit event** window.
- ___ 10. Deploy the **Trades** application to the **default** integration server.
 - ___ a. Right-click the **Trades** application in the Application Development view and then click **Deploy**.
 - ___ b. Select the **default** integration server.
- ___ 11. Validate that the **Trades** application is deployed.

Part 2: Set up the environment for Record and Replay

- ___ 1. Create the database tables for storing the messages in the TRADES database.
 - ___ a. Run an IBM Integration Console as the administrator (click the IBM Integration Console shortcut on the desktop and then click **Run as administrator**).
 - ___ b. Go the database setup directory for the lab:
C:\Labs\Lab09-RandR\install\DBSetup
 - ___ c. Run the command: `CreateTRADES_Tables.bat`

This command opens a new command window and creates the Record and Replay tables in the TRADES database. The tables are re-created so that this lab starts with a clean display of monitor events, and so that new events are easily viewable in the IBM Integration web console.
 - ___ d. Run the command: `configureJDBC`

This command defines the connection to the TRADES database.
- ___ 2. Enable flow monitoring on the message flows in the **Trades** application.
 - ___ a. In the IBM Integration Console, change directories to:
C:\Labs\Lab09-RandR\monitoring
 - ___ b. Run the command: `enableMonitoringTrades`

This command issues the integration node command:

```
mqsichangeflowmonitoring IB9NODE
-e default
-k Trades
-f TRD1_Validate
-c active
```

It also issues equivalent commands for the other message flows in the **Trades** application.



Important

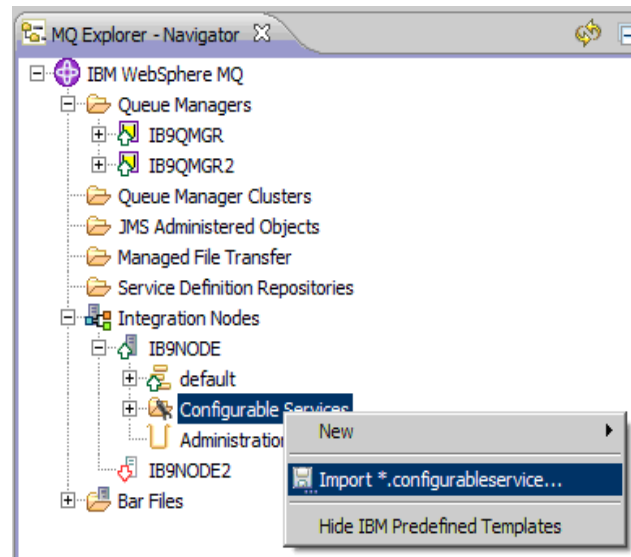
If you redeploy the **Trades** application, the flow monitoring status is reset. You must reissue the command `enableMonitoringTrades` command to reactivate flow monitoring.

- ___ 3. 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.

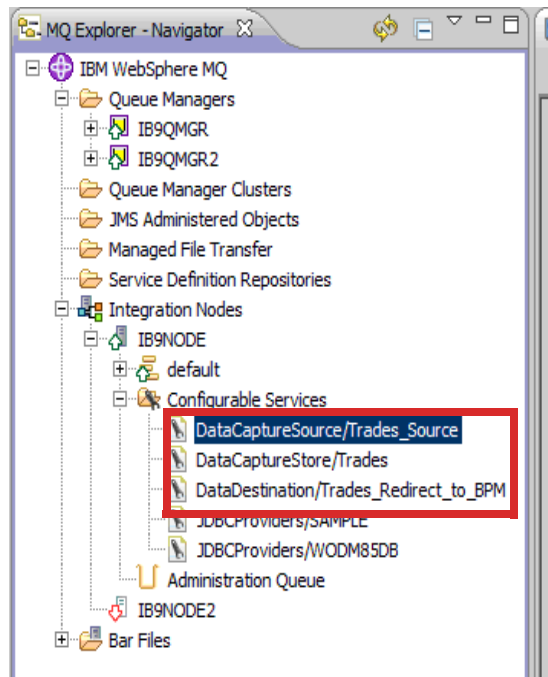
You can define the configurable services manually. For this lab, the configurable services are already created but they must be imported into the IB9NODE integration node.

- ___ a. Open IBM Integration Explorer.
- ___ b. Expand **IB9NODE** under the **Integration Nodes** folder.
- ___ c. Right-click **Configurable Services** and then click **Import *configurable service**.



- ___ d. Browse to the C:\Labs\Lab09-RandR\configurable_services directory and click Trades_data_capture_store.configurableservice.
- ___ e. The **DataCaptureStore/Trades** configurable service should now be listed under the **Configurable Services** folder.
- ___ 4. Following the same process as in the previous step, import Trades_source.configurableservice and Trades_BPM_Data_Destination.configurableservice from the C:\Labs\Lab09-RandR\configurable_services directory.

You should now have three new configurable services for the Trades application.

**Note**

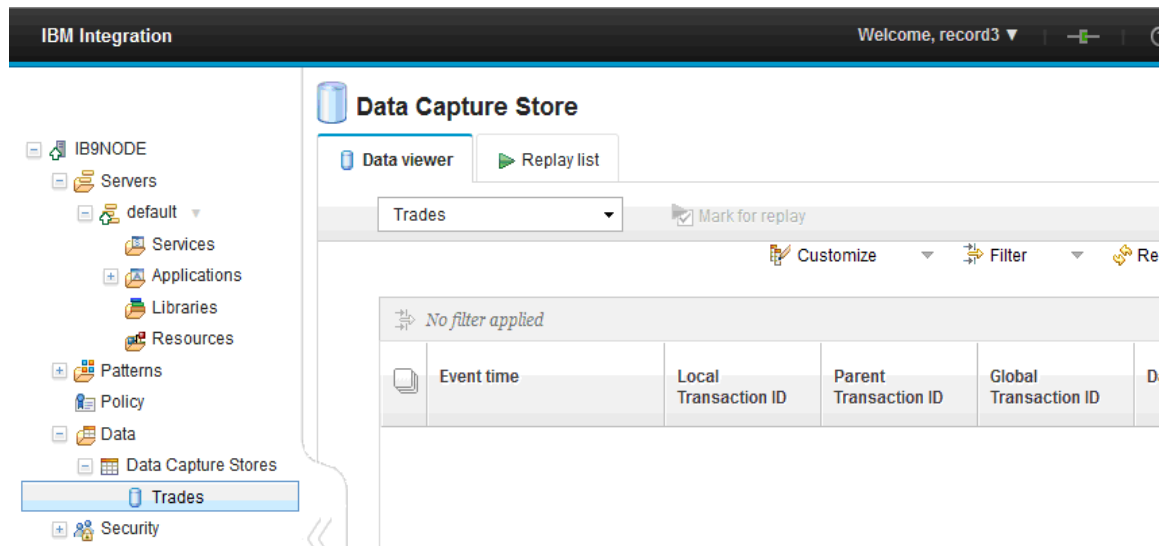
The **Trades_source** service subscribes to the topic `$SYS/Broker/IB9NODE/Monitoring/default/#`. So this data source collects all monitoring events that the applications in the **default** integration server generate. This data source does not collect events that are emitted in other nodes or integration servers.

Part 3: View messages in the IBM Integration web console

- ___ 1. Open a web browser.
- ___ 2. Enter the following URL to start the IBM Integration web console:
http://localhost:4414
- ___ 3. In the IBM Integration web console navigator, you should see full details of the integration node, and the deployed applications. You can also complete various operations, such as starting and stopping the applications.

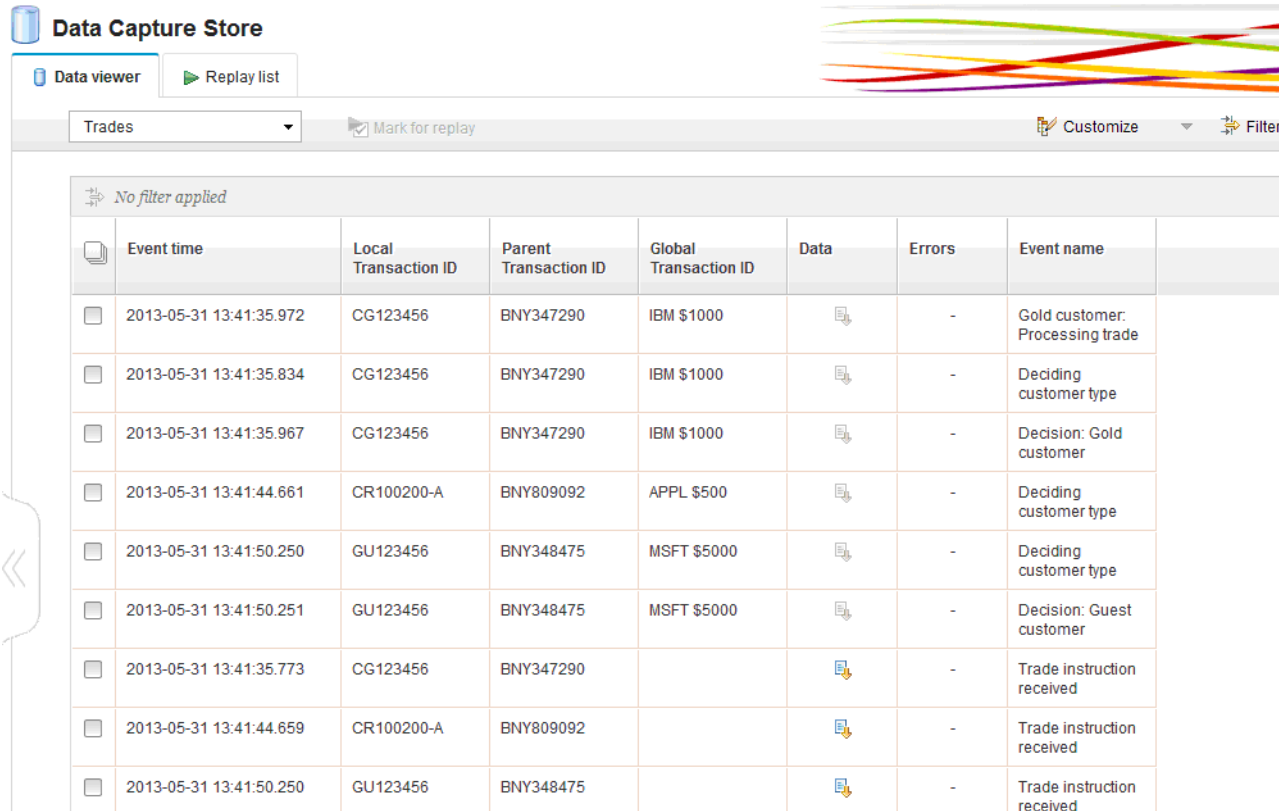
Expand **Data > Data Capture Stores** in the navigator and then click the **Trades** data capture store.

The events list for **Trades** is empty.



- ___ 4. Use RFHUtil to send some new events to the **Trades** application.
 - ___ a. In RFHUtil, open the file for a 'gold' customer:
C:\Labs\Lab09-RandR\data\TradeMessageGold_BNY347290.xml.
 - ___ b. Send one instance of the data to the queue TRADE.VALIDATE.IN.
 - ___ c. Send a message to the TRADE.VALIDATE.IN queue for a 'regular' customer:
C:\Labs\Lab09-RandR\data\TradeMessageRegular_BNY809092.xml.
 - ___ d. Send a message to the TRADE.VALIDATE.IN queue for a 'guest' customer:
C:\Labs\Lab09-RandR\data\TradeMessageGuest_BNY348475.xml.

- ___ 5. Click **Refresh** on the **Data viewer** in the IBM Integration web console to show the new events.



Event time	Local Transaction ID	Parent Transaction ID	Global Transaction ID	Data	Errors	Event name
2013-05-31 13:41:35.972	CG123456	BNY347290	IBM \$1000		-	Gold customer: Processing trade
2013-05-31 13:41:35.834	CG123456	BNY347290	IBM \$1000		-	Deciding customer type
2013-05-31 13:41:35.967	CG123456	BNY347290	IBM \$1000		-	Decision: Gold customer
2013-05-31 13:41:44.661	CR100200-A	BNY809092	APPL \$500		-	Deciding customer type
2013-05-31 13:41:50.250	GU123456	BNY348475	MSFT \$5000		-	Deciding customer type
2013-05-31 13:41:50.251	GU123456	BNY348475	MSFT \$5000		-	Decision: Guest customer
2013-05-31 13:41:35.773	CG123456	BNY347290			-	Trade instruction received
2013-05-31 13:41:44.659	CR100200-A	BNY809092			-	Trade instruction received
2013-05-31 13:41:50.250	GU123456	BNY348475			-	Trade instruction received

- ___ 6. The **Data viewer** uses the standard column heading names. You can customize the headings so that they are more descriptive.

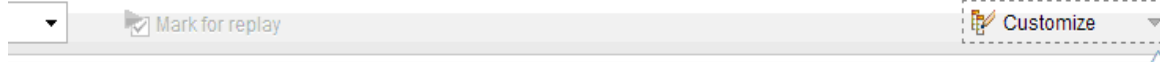
There are several options for customizing the column headings:

- 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 are made by this user. 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 column heading Display name 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.

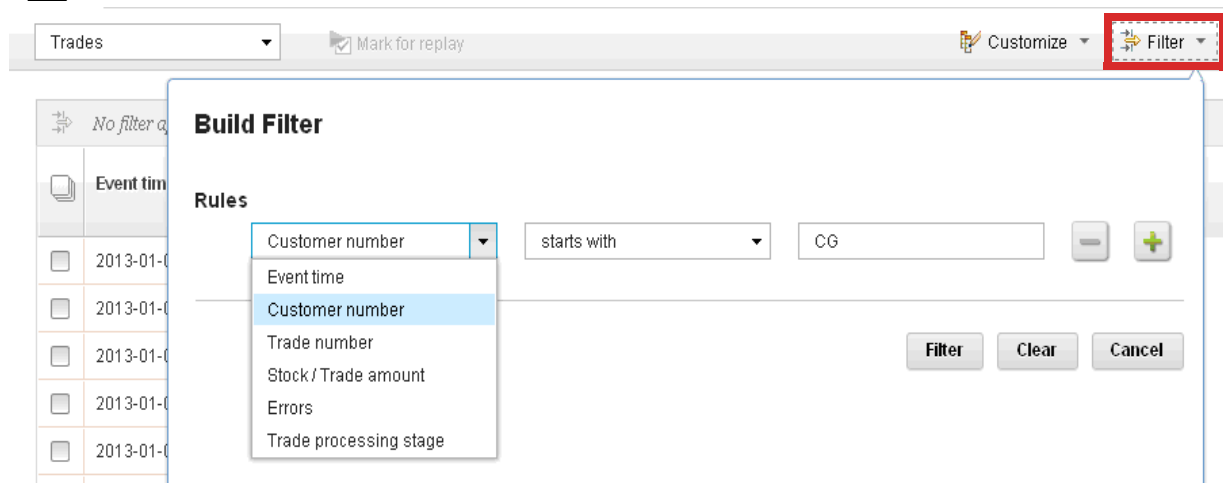


Customize Columns

Select the columns to display in the Data viewer. Double-click a name or width that you want to edit. You can sort the order by clicking the header. You can also reorder the columns and change their widths by using the header in the main Data viewer and saving your changes here. The saved changes apply only to the current data capture store; other data capture stores retain their current settings.

Field ID		Display Name	Width (px)
eventTimestamp	<input checked="" type="checkbox"/>	Event time	160
localTransactionId	<input checked="" type="checkbox"/>	Customer number	100
parentTransactionId	<input checked="" type="checkbox"/>	Trade number	100
globalTransactionId	<input checked="" type="checkbox"/>	Stock / Trade amount	100
hasBitstream	<input type="checkbox"/>	Data	70

- ___ f. Click **Apply**. The column names are updated with the new values.
- ___ 7. 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.
- ___ 8. You can limit the data that is displayed in the **Data viewer** by using the **Filter** function.
- ___ a. Click **Filter**.
- ___ b. Specify the filter criteria of: Customer number starts with CG.
- ___ c. Click **Filter** to activate the defined filters.



- ___ 9. 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 criteria of: Trade processing stage contains Complete.
 - ___ c. Click **Filter** to activate the new filter.

**Note**

The values for each filter are case-sensitive.

- ___ 10. Clear the filters by selecting **Filter** from the **Data viewer** tab and then clicking **Clear**.

Part 4: Replay messages

In this part of the exercise, you configure the IB9NODE to allow messages to be replayed. Replay allows messages that are displayed on the IBM Integration web console to be selected and sent to the same, or a different, message flow for further processing.

In this lab, you replay the message by sending it to a separate WebSphere MQ queue; another application does not process it.

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 separate queue and a separate (and different) message flow.

- ___ 1. Review the configurable service that was defined to enable the replay.
 - ___ a. In the IBM Integration Explorer, expand the **Configurable Services** folder under the IB9NODE.
 - ___ 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 IB9QMGR.

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 console **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.

Data viewer | **Replay list**

Trades | Mark for replay

No filter applied

	Event timeX	Customer number	Trade number	Stock / Trade amount	Data	Error
<input type="checkbox"/>	2013-01-02 11:16:52.598	CG123456	BNY347290			
<input checked="" type="checkbox"/>	2013-01-02 11:16:52.598	CG123456	BNY347290			
<input type="checkbox"/>	2013-01-02 10:53:36.920	GU123456	BNY348475	MSFT \$5000		

- ___ 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**.

Data viewer | **Replay list**

Data Destination | Select destination | Replay All

Replay Status | Event timeX | Customer number | Trade number

<input checked="" type="checkbox"/>	-	2013-01-02 11:16:52.598	CG123456	BNY347290
-------------------------------------	---	-------------------------	----------	-----------

- ___ 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.

Items that did not contain a data bitstream cannot be replayed, as described by the error message that is seen on the **Replay list** view.

Data viewer **Replay list**

Data Destination: Trades_Redirect_to_BPM Replay All

	Replay Status	Event timeX	Customer number	Trade number
	<u>Success</u>	2013-01-02 11:16:52.598	CG123456	BNY347290

- ___ 6. Confirm that the messages were sent to the Replay queue.
 - ___ a. Open IBM Integration Explorer.
 - ___ b. Select **Queues** under **IB9QMGR**.
 - ___ c. The **Current queue depth** of TRADE.FIX.OUT should increase by 2 (or with the number of messages you sent for replay).

Part 5: Handle failed messages

If a message flow encounters an error during processing, it can be captured and reported by using the IBM Integration web console.

To enable this feature, configure the monitoring point on the message flow node to include the `$ExceptionList` in the monitoring event message.

- ___ 1. In this exercise, the **TRD1_Validate** message flow is configured with `$ExceptionList` in the monitoring event message.
 - ___ a. In the IBM Integration Toolkit, open the **TRD1_Validate** message flow if it is not already open.
 - ___ b. Click the **Validation Failure** node.
 - ___ c. In the **Monitoring** properties for the node, click **Edit**.

In the **Event Payload** section, you should see that the **Data Location** is configured to include the `$ExceptionList`.


- ___ 2. Using RFHUtil, send another message to the **Trades** application.
 - ___ a. Open the file `C:\Labs\Lab09-RandR\data\TradeMessage_BadMessage.xml`.
 - ___ b. Send the message to `TRADE.VALIDATE.IN` queue.

Although this message is a valid XML message, it is missing a required XML element. It fails validation on the **ReceiveTrade** node because validation set to **Content and Value**.


- ___ 3. In the IBM Integration web console, you should see two new entries.


The **Trade processing stage** for the first event is *Trade instruction received*.



The **Trade processing stage** for the second event is *Data validation failure*.

 **Data Capture Store**

Data viewer | **Replay list**

Trades | ☒ Mark for replay |  Customize |  Filter

 No filter applied

	Event time	Customer number	Trade number	Stock / Trade amount	Data	Trade processing stage
<input type="checkbox"/>	2013-01-14 15:38:37.394	CG123456	BNY590012			Trade instruction received
<input type="checkbox"/>	2013-01-14 15:38:37.394	CG123456	BNY590012			Data validation failure

- ___ 4. Failed events can be highlighted in the IBM Integration web console **Data viewer** by customizing the displayed columns. Click **Customize**.
 - ___ c. If it is not already selected, enable the **hasException (Errors)** column.
 - ___ d. Click **Apply**.

The **Errors (hasException)** column shows a red cross for all monitoring events that contain the `$ExceptionList` data.

Trades

Mark for replay

Customize

Filter

Refresh

No filter applied

	Event time	Customer number	Trade number	Stock / Trade amount	Data	Errors	Trade processing stage
	2013-01-14 15:38:37.394	CG123456	BNY590012			-	Trade instruction received
	2013-01-14 15:38:37.394	CG123456	BNY590012				Data validation failure

Part 6: Clean up

- ___ 1. Close all open editor windows by typing Ctrl + Shift + W. You do not need to save any changes, if you are prompted.
- ___ 2. In the **Integration nodes** view, right-click the **default** integration server, and then click **Delete > All Flows and Resources**. When the delete confirmation dialog box is displayed, click **OK**.
- ___ 3. Close RFHUtil.
- ___ 4. Close the IBM Integration web console.

End of exercise

Exercise review and wrap-up

In Part 1 of this exercise, you imported an application into the IBM Integration Toolkit and verified event configuration on the message flow.

In Part 2 of this exercise, you set up the environment for recording and replaying messages. Set up included building the database tables and importing the configurable services.

In Part 3 of this exercise, you viewed event messages in the IBM Integration web console, customized the Data viewer, and defined filters to limit the data that is displayed.

In Part 4 of this exercise, you replayed messages by sending them to an alternative queue.

In Part 5 of this exercise, you configured the message flow to generate the `$ExceptionList` and identified failed messages in the IBM Integration web console.

Exercise 10. Creating and implementing a user-defined pattern

What this exercise is about

In this exercise, you create and customize a user-defined pattern. You also package the pattern for deployment.

What you should be able to do

At the end of this exercise, you should be able to:

- Create an exemplar message flow
- Create a pattern authoring project
- Configure the pattern authoring project to specify which elements of the project are customizable by the user when the pattern is used
- Test a user-defined pattern
- Package pattern plug-ins for distribution to users

Introduction

A *pattern* captures a tested solution to a commonly recurring problem, addressing the objectives that you want to achieve. Patterns typically emerge from common usage and the application of a particular product or technology. An IBM Integration Bus pattern can be used to generate customized solutions to a recurring problem in an efficient way. IBM Integration Bus patterns are provided to encourage the adoption of preferred techniques in message flow design to produce efficient and reliable flows.

A *user-defined pattern* extends the function of IBM Integration Bus so that you are able to create patterns that you can reuse within your organization. The *pattern author* creates a user-defined pattern to meet a business or technical requirement, and the *pattern user* configures a user-defined pattern that the pattern author created.

To create user-defined patterns, you must provide the implementation of the solution that reflects good practice in your organization. This implementation is known as an *exemplar*. An exemplar is a project that holds content for a pattern. An exemplar contains message flows and other resources, such as source code, Java classes for JavaCompute

nodes, ESQL modules, message maps, test client, XML files, and style sheet files. Exemplars are used to create pattern plug-ins by configuring a pattern authoring project.

In this exercise, you use the Pattern Authoring editor to configure a pattern and create the pattern plug-ins that implement the pattern. You also learn how to package the pattern plug-ins for distribution so that other people can use your pattern. Your pattern is displayed in the Patterns Explorer view and can be used in the same way as the built-in patterns.

Requirements

A workstation with the WebSphere MQ and IBM Integration Bus components installed.

Exercise instructions

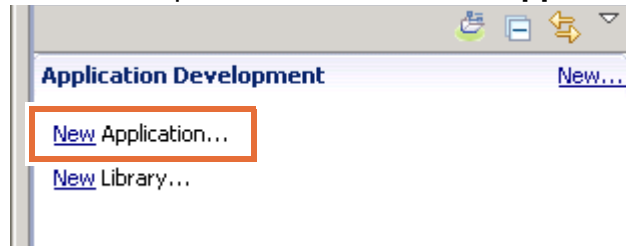
Part 1: Start components and prepare the workspace

- ___ 1. Make sure that all local IBM Integration Bus components are running.
- ___ 2. Open a new workspace that is named C:\Workspace\pattern.

Part 2: Create an exemplar message flow

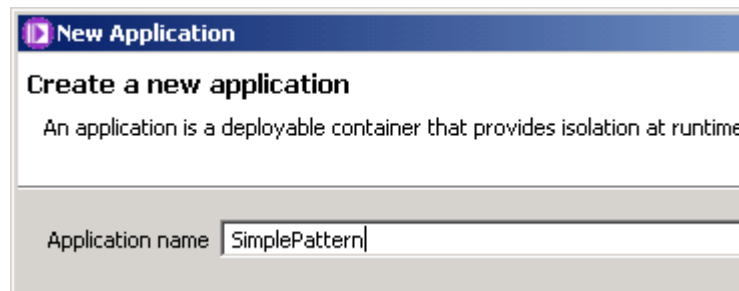
In this part of the exercise, you create an exemplar (template) message flow. This template contains the message flow that becomes the pattern.

- ___ 1. Create an application to hold the exemplar.
 - ___ a. In the Application Development view, click **New Application**.



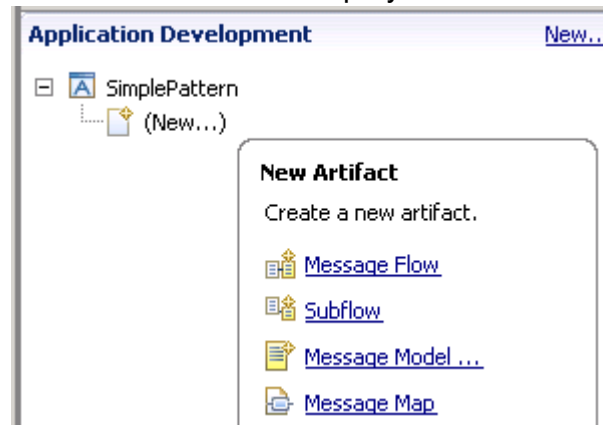
The **Create a new application** dialog box is displayed.

- ___ b. For **Application name**, enter: SimplePattern

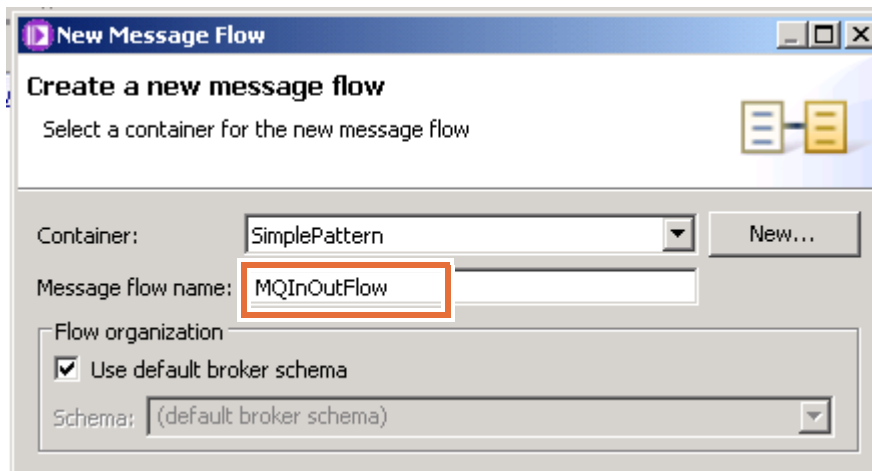


- ___ c. Click **Finish**. The new application is created and is displayed in the Application Development view.

- ___ 2. Create a container to hold the exemplar message flow.
- ___ a. In the Application Development view, expand **SimplePattern**, and then click **(New)**. The **New Artifact** menu is displayed.



- ___ b. Click **Message Flow**. The **Create a new message flow** dialog box is displayed.
- ___ c. For **Message flow name**, enter: MQInOutFlow



- ___ d. Click **Finish**. The Message Flow editor opens.
- ___ 3. Create the exemplar message flow.
- ___ a. From the **WebSphere MQ** drawer in the palette, add an MQInput node and an MQOutput node to the canvas.
- ___ b. Connect the **out** terminal of the MQInput node to the **in** terminal of the MQOutput node.
- ___ c. On the MQInput node **Basic** properties tab, set **Queue name** to: DEFAULT.IN
- ___ d. On the MQOutput node **Basic** properties tab, set **Queue name** to: DEFAULT.OUT

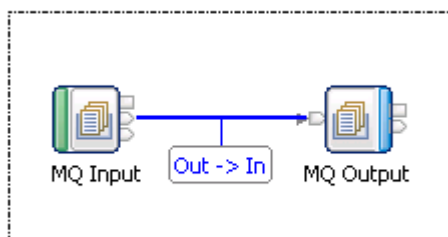
This basic message flow is what the user instantiates.



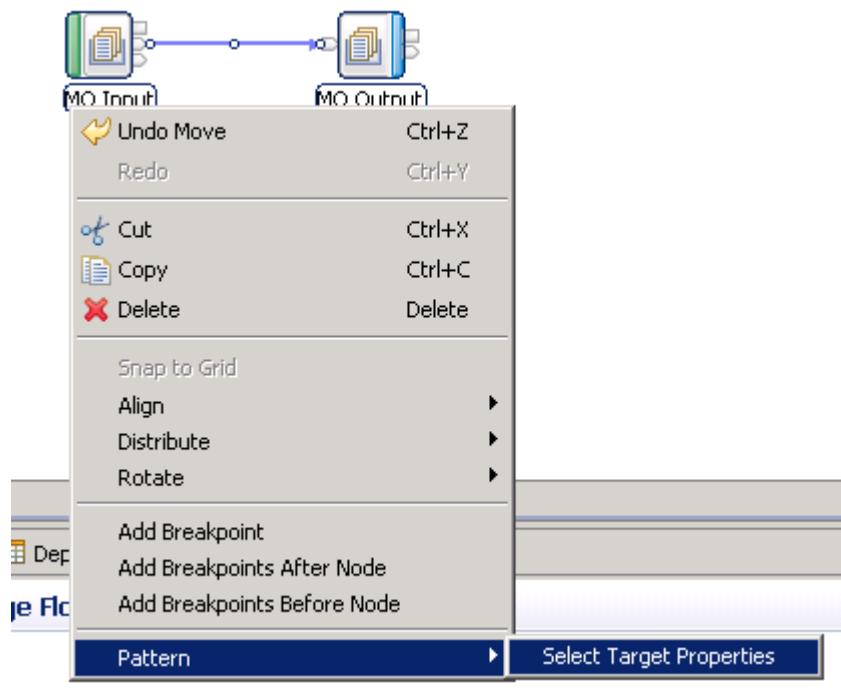
- ___ 4. Designate the properties that are visible to the user at pattern instantiation.

In this step, you indicate that properties that the user can configure when instantiating the pattern from the patterns gallery.

- ___ a. Draw a selection box around the message flow that you created. The message flow components are highlighted when you complete this operation.



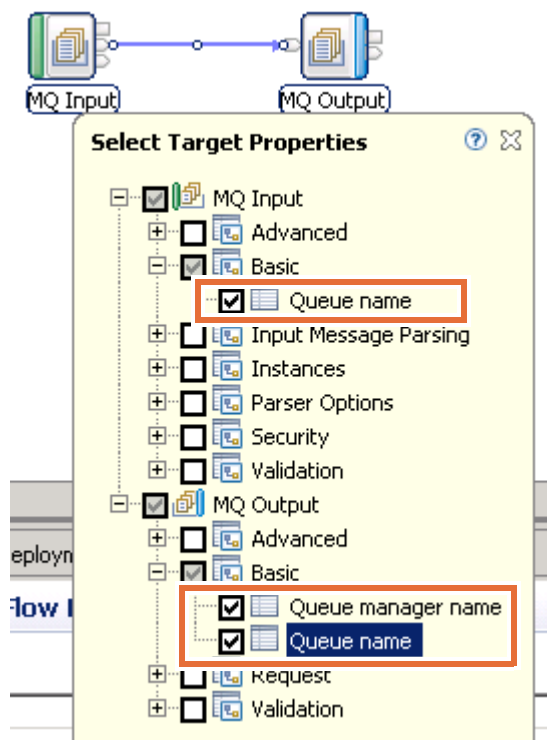
- ___ b. Right-click any component in the message flow, and then click **Pattern > Select Target Properties**.



The **Select Target Properties** menu is displayed.

- ___ c. Expand **MQ Input > Basic**, and then click **Queue name**.

- ___ d. Expand **MQ Output > Basic**, and then select both **Queue manager name** and **Queue name**.



- ___ e. Click the **X** in the upper right corner of the **Select Target Properties** menu to close it.

A decorator is displayed above the two nodes in the message flow. These indicators show that those nodes have properties that are eligible to be changed when the pattern is instantiated.

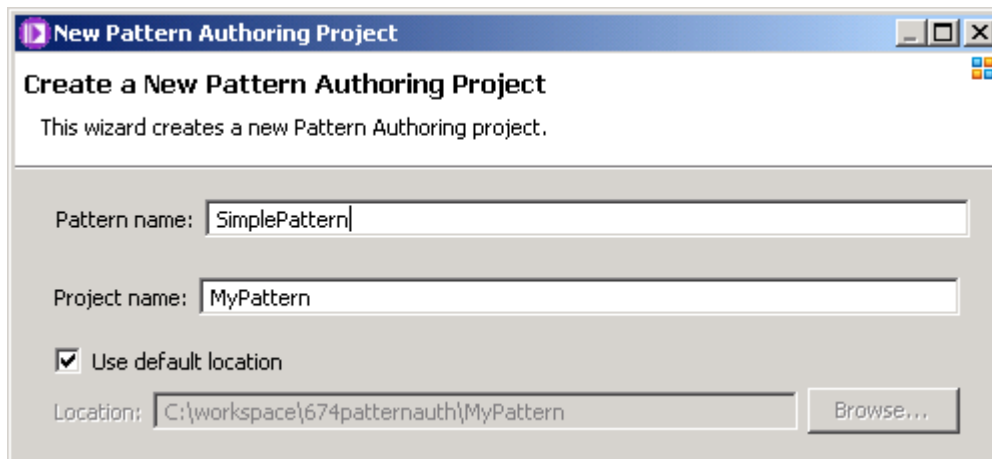
- ___ 5. Save the message flow.

Part 3: Create a pattern authoring project

In this part of the exercise, you create a pattern authoring project. The project contains pattern plug-in files, which you generate in this section.

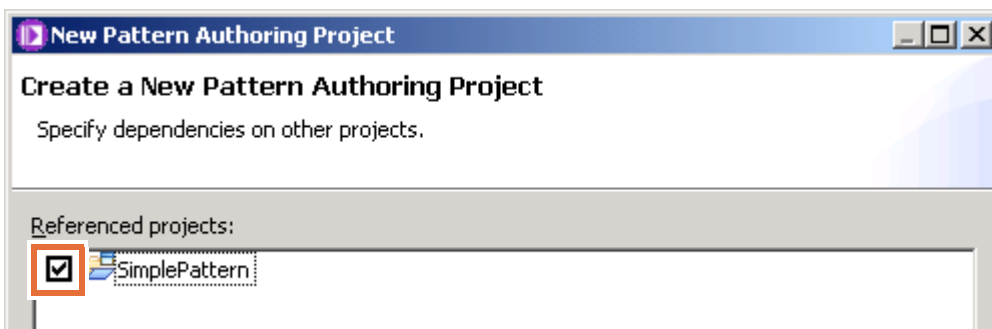
- ___ 1. From the menu bar, click **File > New > Pattern Authoring Project**. The **New Pattern Authoring Project** dialog box is displayed.

- ___ 2. For **Pattern Name**, enter: SimplePattern



- ___ 3. Click **Next**.

- ___ 4. In the **Referenced projects** section, click the check box to the left of **SimplePattern**



- ___ 5. Click **Finish**. The pattern authoring project is created, and the pattern opens in the pattern editor.

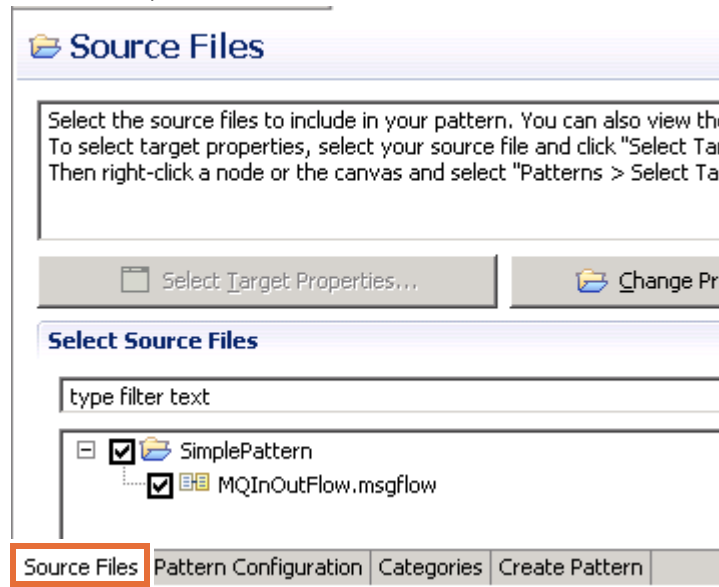
Part 4: Configure the pattern authoring project

In this part of the exercise, you configure the appearance of the newly created pattern. You configure the pattern by using the four tabs that are displayed in the pattern editor window:

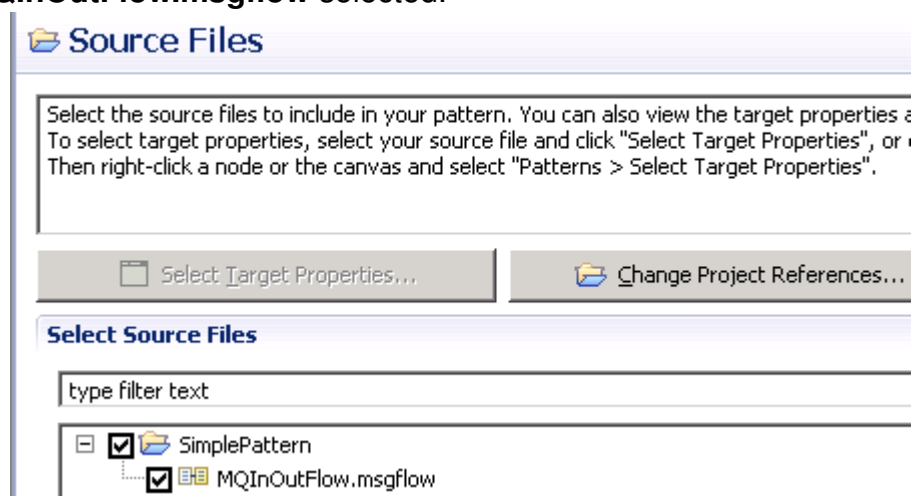
- Source files
- Pattern configuration
- Categories
- Create pattern

___ 1. Review the source files that are included when the user instantiates the pattern.

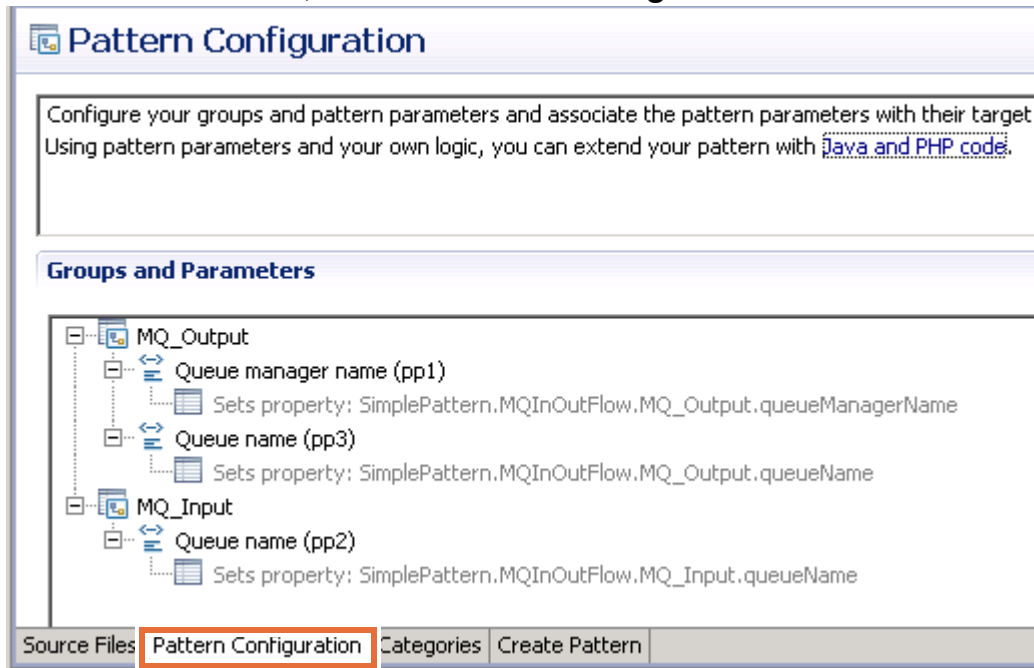
___ a. In the Pattern editor, click the **Source Files** tab.



___ b. Review the source files that are selected. In this case, because you want to include the message flow whenever the pattern is instantiated, leave the **MQInOutFlow.msgflow** selected.



- ___ 2. Configure the pattern parameters. The **Pattern Configuration** tab contains what users see when they configure the pattern parameters at pattern instantiation time.
- ___ a. In the Pattern editor, click the **Pattern Configuration** tab.



The **MQ_Output** and **MQ_Input** nodes that are displayed in the tree represent groups. These groups are used to keep related pattern parameters together. By default, the Pattern Editor displays one group per pattern node in the message flow. (This display format can be changed.)

At pattern instantiation time, users can provide values for the pattern parameters that are displayed under the groups. Under each parameter is an action that is shown in a gray font. These actions describe the effect of setting each parameter.

- ___ b. Select the **MQ_Input** group for editing. There are two ways to do so:
- Double-click the group name, or
 - Select the group name and then click **Edit** on the right side of the window. It might be necessary to maximize the Pattern editor (by double-clicking the **SimplePattern.pattern** tab) to do so.

The **Configure group** dialog box is displayed.

Edit Group: MQ_Input

Configure group

Configure the pattern parameter group and how it is displayed to pattern users.
Configure an XPath expression that controls when this group is enabled in the Pattern Instance editor.

Basic | Enable

Group Display

Display name: MQ_Input

Description: Pattern Parameters

Group Options

☒ Generate help documentation Select this option to create help information.

☒ Display parameters in a group box Select this option to display this group.

- ___ c. In the **Group Display** section, change **Display name** to: `Input queue name` and set **Description** to: `Where the incoming message originates`.

Edit Group: MQ_Input

Configure group

Configure the pattern parameter group and how it is displayed to pattern users.
Configure an XPath expression that controls when this group is enabled in the Pattern Instance editor.

Basic | Enable

Group Display

Display name: Input queue name

Description: Where the incoming message originates

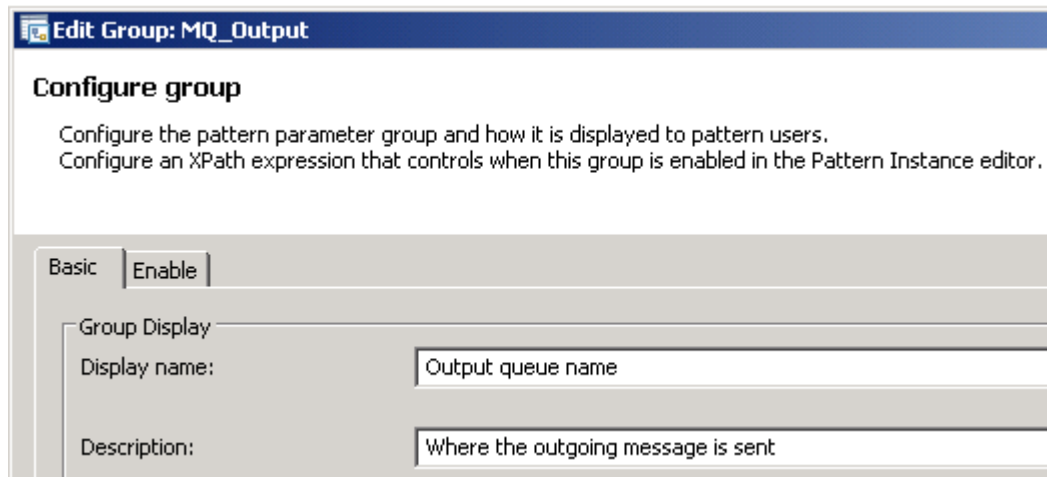
Group Options

☒ Generate help documentation Select this option to create help information.

☒ Display parameters in a group box Select this option to display this group.

- ___ d. Click **OK** at the bottom of the window. It might be necessary to move the window to see this control.
- ___ e. Make the **Input queue name** group display before the **MQ_Output** group by clicking **Input queue name** and dragging it above the **MQ_Output** group.
- You can also use the up-arrow and down-arrow controls on the right side of the window to reorder the groups.
- ___ f. Select the **MQ_Output** group for editing by using the same steps that you used for the **MQ_Input** group in step b.

- ___ g. In the **Group Display** area, change **Display name** to: Output queue name and set **Description** to: Where the outgoing message is sent.



- ___ h. Click **OK** at the bottom of the window. It might be necessary to move the window to see this control.
- ___ 3. Customize the pattern parameters and how they are displayed to the pattern user.
- ___ a. From the **Input queue name** group, select the **Queue name** property for editing. Again, you can do so by double-clicking the property, or by selecting it and clicking **Edit** on the right side of the window. The **Edit parameter** dialog box is displayed.
- ___ b. In the **Basic** tab, change the following parameters:
- In the **Parameter Display** section, change **Display name** to:
Input queue name
 - In the **Parameter Options** section, change **Field prompt** to:
Enter the input queue name
 - In the **Help Text (HTML)** section, change the help text to any text you want, for example:

```
<p>Enter the name of the input queue from which the message is read.
This is a <b>mandatory</b> parameter.</p>
```

As you enter the text, you see that the formatted message is displayed in the preview pane to the right. You can include most valid HTML markup tags in this field.

Edit Parameter: Queue name

Configure pattern parameter

Configure the pattern parameter and how it is displayed to pattern users.

Basic | Editor | Transform | Enable

Parameter Display

Display name:

Parameter Options

☐ Hide the parameter Select this option to hide the parameter from the pattern user interface. Use an XPath expression to set the value of the parameter.

☒ Mandatory parameter Select this option if the pattern user must enter a value for this parameter. Mandatory parameters also display a field prompt to the user.

Field prompt:

Help Text (HTML)

Enter any HTML or text that you want to display as help text for this parameter. Do not include <html> or <head> tags because the text is inserted into an HTML file.

- ___ c. Click **OK**.
- ___ d. From the **Output queue name** group, click the **Queue name** property for editing.
- ___ e. In the **Basic** tab, change the following parameters:
 - In the **Parameter Display** section, change **Display name** to:
Output queue name
 - In the **Parameter Options** section, change **Field prompt** to:
Enter the output queue name

- In the **Help Text (HTML)** section, change the help text to any text you want, for example:

<p>Enter the name of the output queue to which the message is written. This is a mandatory parameter.</p>

Edit Parameter: Queue name

Configure pattern parameter

Configure the pattern parameter and how it is displayed to pattern users.

Basic | Editor | Transform | Enable

Parameter Display

Display name:

Parameter Options

☐ Hide the parameter Select this option to hide the parameter from the pattern user. Use an XPath expression to set the value of the parameter.

☒ Mandatory parameter Select this option if the pattern user must enter a value. Mandatory parameters also display a field prompt to the user.

Field prompt:

Help Text (HTML)

Enter any HTML or text that you want to display as help text for this parameter. Do not include <html> or <head> tags because the text is inserted into an HTML file.

- ___ f. Click **OK**.
- ___ g. From the **Output queue name** group, select the **Queue manager name** property for editing.
- ___ h. In the **Basic** tab, change the following parameters:
 - In the **Parameter Display** section, change **Display name** to:
Output queue manager name
 - In the **Parameter Options** section, clear **Mandatory parameter**.

- In the **Help Text (HTML)** section, change the help text to any text you want, for example:

<p>Enter the name of the output queue manager. If no name is supplied, the integration node queue manager is used.</p>

Edit Parameter: Queue manager name

Configure pattern parameter

Configure the pattern parameter and how it is displayed to pattern users.

Basic | Editor | Transform | Enable

Parameter Display

Display name: Queue manager name

Parameter Options

☐ Hide the parameter Select this option to hide the parameter from the palette. Use an XPath expression to set the value of the parameter.

☐ Mandatory parameter Select this option if the pattern user must enter a value. Mandatory parameters also display a field prompt to the user.

Field prompt: Enter the queue manager name

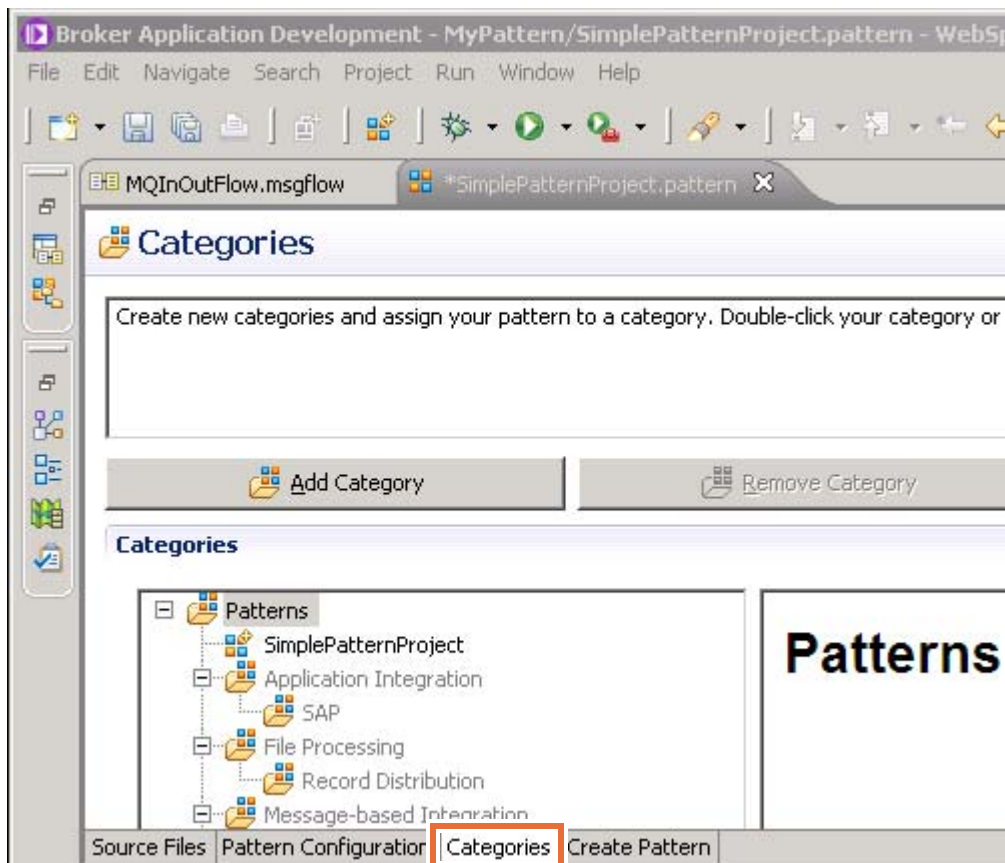
Help Text (HTML)

Enter any HTML or text that you want to display as help text for this parameter. Do not include <html> or <head> tags because the text is inserted into an HTML file.

<p>Enter the name of the output queue manager. If no name is supplied, the broker's default queue manager is used.</p>

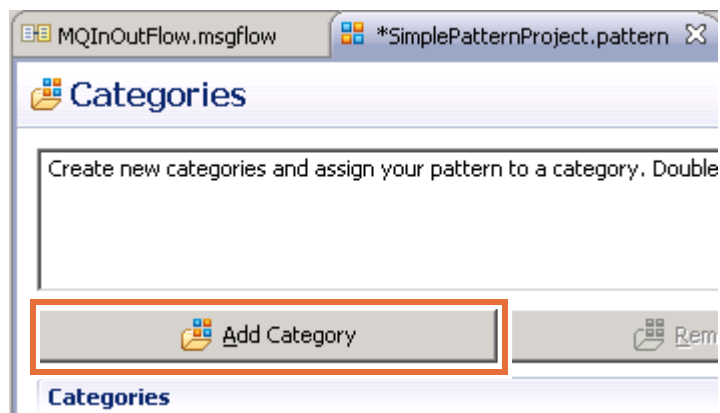
- Click **OK**.

- ___ 4. Set the category and explanatory text that is associated with the pattern. The category determines where the pattern is displayed in the **Patterns Explorer**.
- ___ a. In the Pattern editor, click the **Categories** tab.



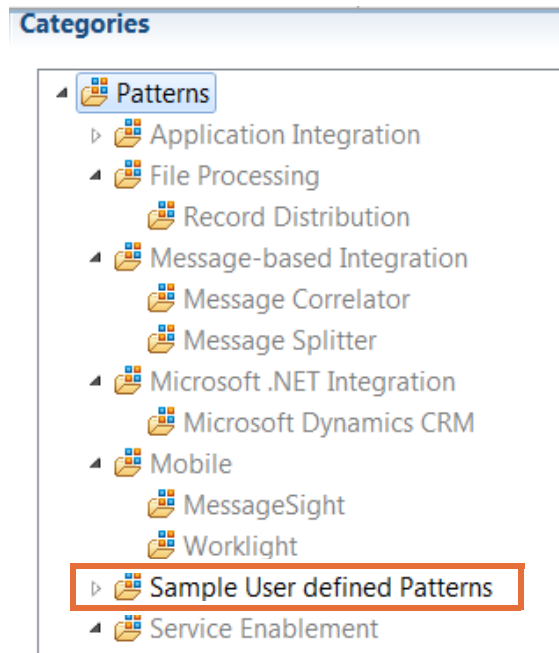
The SimplePattern is displayed under the top level of the hierarchy, with no category associated with it.

- ___ b. To create a category for the pattern, click **Add Category**.

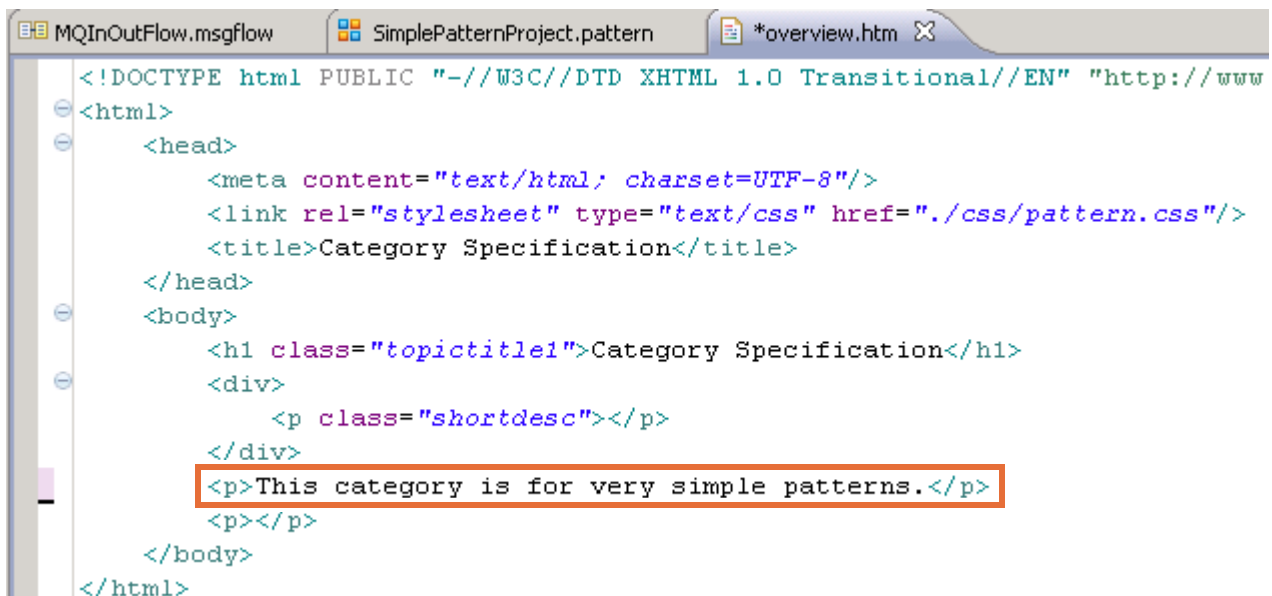


- ___ c. Name the category: Sample user defined Patterns, and then click **OK**.

- ___ d. When you are prompted to save the pattern file, click **Yes**. The **Sample User defined patterns** category is displayed in the category hierarchy.

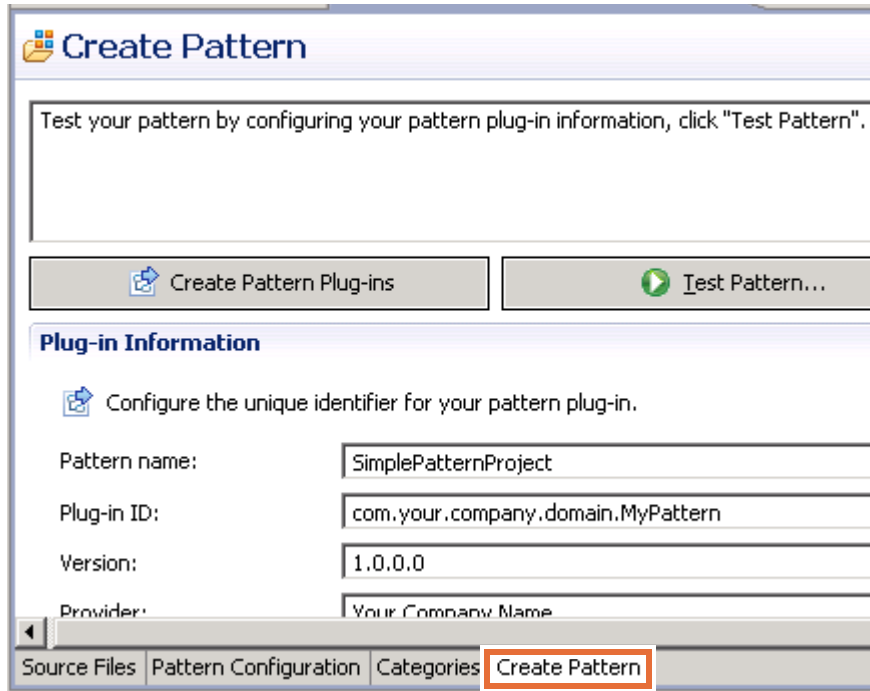


- ___ 5. Update the text that is associated with the **Sample User defined Patterns** category.
- ___ a. Double-click the category name, or select the category name and click **Edit HTML** from the controls on the right side of the page.
- ___ b. Change the text under the **shortdesc** division as you want; for example:
- This category is for very simple patterns.

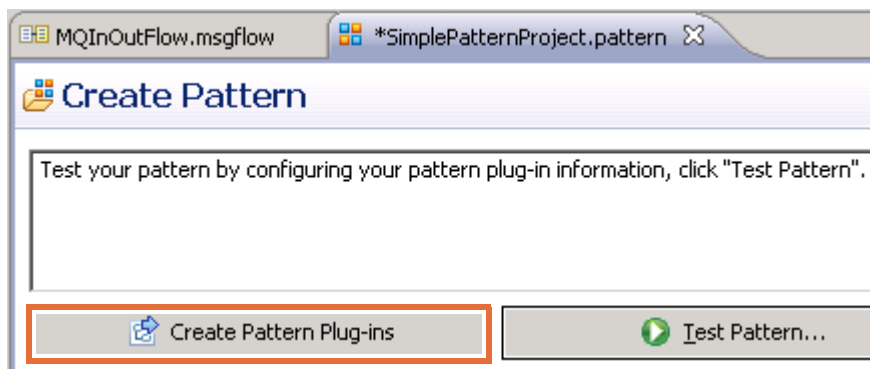


- ___ c. Press Ctrl + S to save the HTML changes, and then close the **overview.htm** tab. The updated category text is displayed in the right pane.
- ___ d. Drag the **SimplePattern** to the **Sample User defined Patterns** category.

- ___ e. Optionally, update the text that is associated with the **SimplePattern** pattern. Double-click the pattern name, or select the category name and click **Edit HTML**. Change the HTML as you want.
- ___ f. Press Ctrl + S to save the HTML changes, and then close the **overview.htm** tab. The updated category text is displayed in the right pane.
- ___ 6. Finalize the pattern and test it to prepare it for deployment.
 - ___ a. In the Pattern editor, click the **Create Pattern** tab.



- ___ b. Click **Create Pattern Plug-ins** to generate the distributable plug-ins for the pattern.



The generation process can take several minutes. Wait for the operation to complete before proceeding.

**Note**

The generation process is resource-intensive. You can expect the processor usage to reach 100% for a number of minutes. The VMware image might become unresponsive during this time, but it returns to normal after the generation is complete.

- ___ c. Click **Test Pattern**. You test the pattern instantiation in the same way that an actual user does. After a few moments, a second copy of the IBM Integration Bus Toolkit is started, and workspace launcher is displayed.
- ___ d. Enter a workspace name, such as: `C:\Workspace\patternstest`, and then click **OK**.

**Note**

The IBM Integration Toolkit prevents you from using the same workspace that you are using for the pattern development.

- ___ e. Close the Welcome page. The Application Development view is displayed.

- ___ f. In the **Quick Starts** pane, click **Start from patterns**.

Quick Starts

Start building your application with one of the following tasks.

 [Start by creating an application](#)

An **Application** is a container for all the resources that are required to create a solution. [More...](#)

 [Start by creating a library](#)

A **Library** is a logical grouping of related code, data, or both. [More...](#)

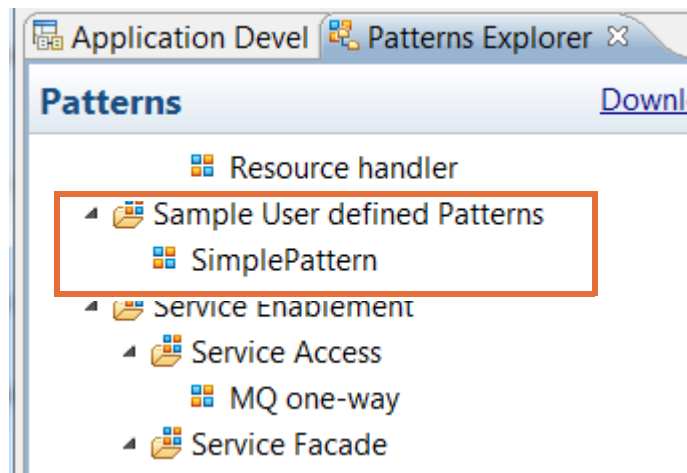
 [Start from WSDL and/or XSD files](#)

Use this task to create an **Application** or **Library** which includes your WSDL and/or XSD files.

 [Start from patterns](#)

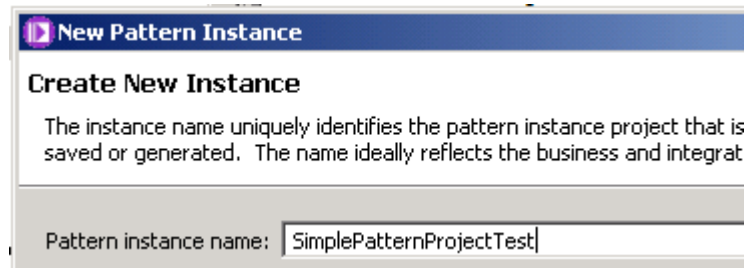
A **Pattern** is a reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task. [More...](#)

The Patterns Explorer opens in the left pane. The **Sample User defined Patterns** category is displayed with the **SimplePattern**.



- ___ g. Click **SimplePattern**. The pattern specification text is displayed in the right pane, including the changes that you made.
- ___ h. Click **Create New Instance** at the bottom of the pane. The **New Pattern Instance** dialog box is displayed.

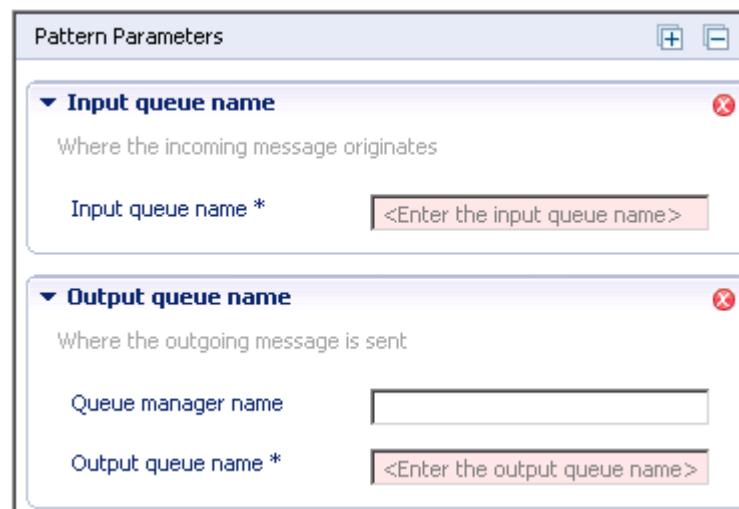
- ___ i. For **Pattern instance name**, enter: SimplePatternProjectTest, and then click **OK**.



The dialog box titled "New Pattern Instance" has a blue header bar with a purple icon. Below the header, the title "Create New Instance" is followed by a descriptive paragraph: "The instance name uniquely identifies the pattern instance project that is saved or generated. The name ideally reflects the business and integrat". At the bottom, there is a text field labeled "Pattern instance name:" containing the text "SimplePatternProjectTest".

The **Configure Pattern Parameters** pane is displayed.

- ___ j. Under **Pattern Parameters**, expand **Input Queue Name** and **Output Queue Name**. Verify that the help text and the field labels display as you configured them.



The "Pattern Parameters" pane has a title bar with a plus icon, a minus icon, and a close icon. It contains two expandable sections. The first section, "Input queue name", has a red close icon and the help text "Where the incoming message originates". It contains a label "Input queue name *" and a text field with the placeholder "<Enter the input queue name>". The second section, "Output queue name", also has a red close icon and the help text "Where the outgoing message is sent". It contains two labels: "Queue manager name" with an empty text field, and "Output queue name *" with a text field containing the placeholder "<Enter the output queue name>".

- ___ k. Under **Pattern Parameters Details**, expand **Input Queue Name** and **Output Queue Name**. Verify that the help text and the field labels display as you configured them.

Input queue name

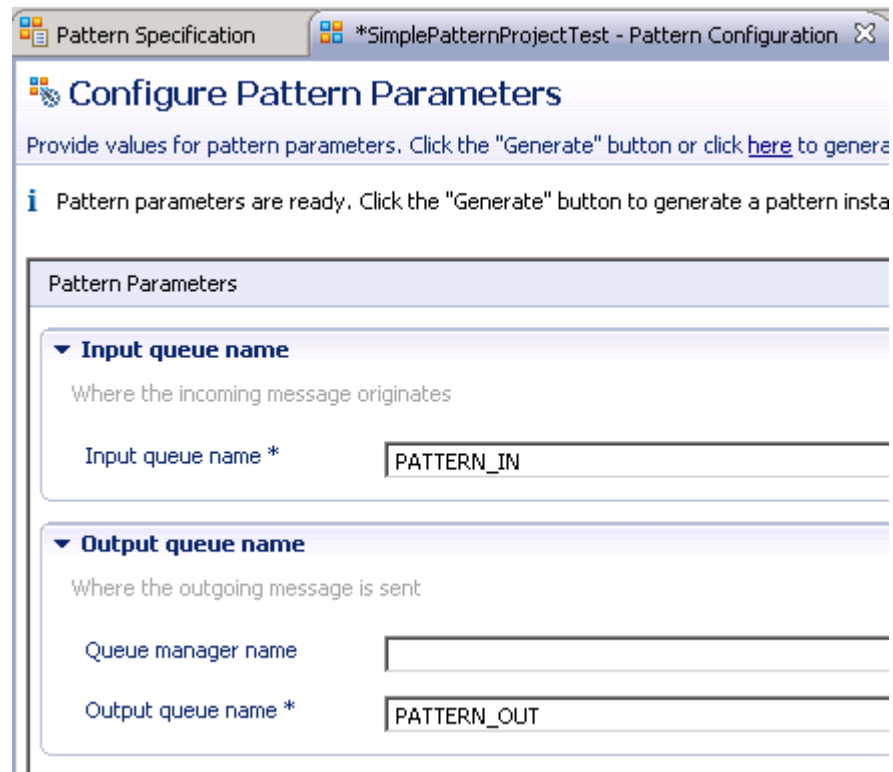
Pattern parameter	Description
Input queue name	Enter the name of the input queue from which the message is to be read. This is a mandatory parameter.

Output queue name

Pattern parameter	Description
Queue manager name	Enter the name of the output queue manager. If no name is supplied, the broker's default queue manager is used.
Output queue name	Enter the name of the output queue to which the message is to be written. This is a mandatory parameter.

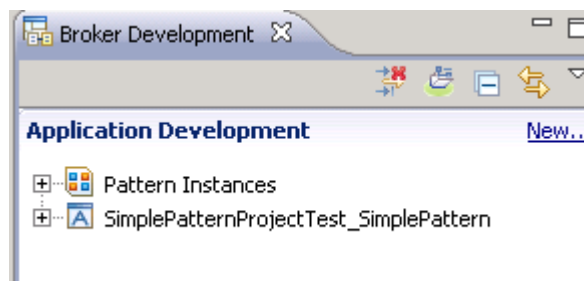
- ___ l. Supply values for the input queue name and the output queue name fields; for example:
- For **Input queue name**, enter: PATTERN_IN

- For **Output queue name**, enter: PATTERN_OUT



- ___ m. Click **Generate** at the bottom of the window to instantiate the pattern.

Pattern generation takes a few minutes. When generation is complete, the pattern instance is displayed in the Application Development view.

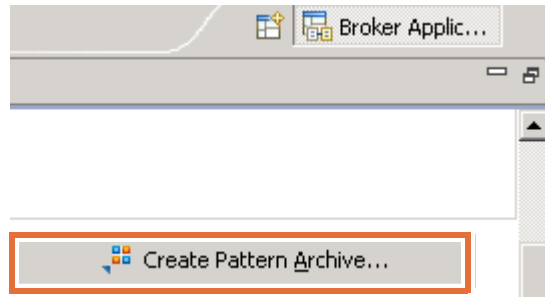


- ___ n. Expand **SimplePatternProjectTest_SimplePattern > Flows**, and then double-click **MQInOutFlow.msgflow**. The message flow editor opens, and displays the message flow.
- ___ o. Select the **MQInput** node and the **MQOutput** node and verify that the queue names match what you used at pattern instantiation (PATTERN_IN and PATTERN_OUT).
- ___ p. After you examine any other elements of the generated pattern, close the IBM Integration Toolkit for that session. You are returned to the IBM Integration Toolkit session that contains the pattern.

Part 5: Package the pattern authoring project for distribution

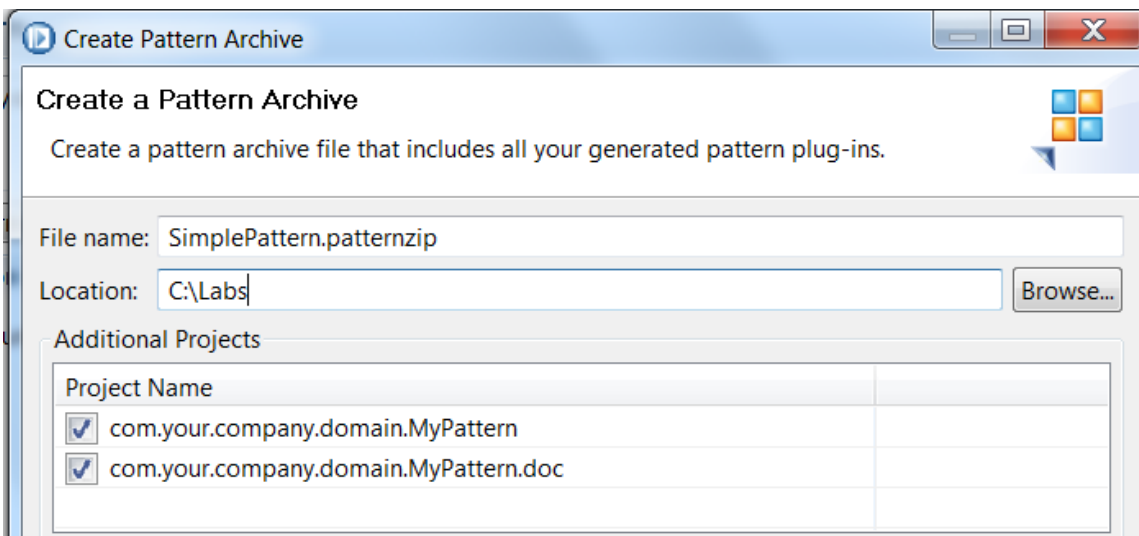
In this part of the exercise, you create a pattern archive, which contains the components that are needed to distribute the new pattern to other developer workstations.

- ___ 1. On the **Create Pattern** tab of the Pattern editor, click **Create Pattern Archive**.



After a few moments, the Create Pattern Archive dialog box is displayed.

- ___ 2. Use the default **File name** that is shown, and change the **Location** to C:\Labs
- ___ 3. Review the project name values (you do not need to change them), and then click **Finish**.



The values for the project names are taken from the **Plug-in Information** section of the Pattern editor. In a production environment, change these values to match your nomenclature standards before creating the pattern archive.

It can take 15-20 minutes to create the pattern archive.



Note

The pattern archive generation process is resource-intensive. Expect the same conditions as you saw when you generated the pattern before testing.

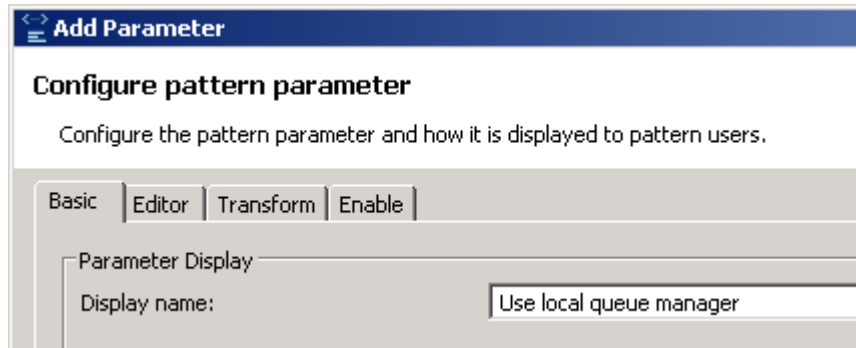
- ___ 4. When the pattern archive is generated, open a Windows Explorer session and browse to the C:\Labs directory. The file type on the pattern archive is .patternzip.

When you distribute this file to another workstation and then double-click the file, the files that are needed to implement the pattern are extracted from the archive.

Part 6: (Optional) Advanced pattern authoring

In this optional section, you add logic to the pattern. This logic gives you more control over the instantiation process.

- ___ 1. Return to the Pattern editor, and click the **Pattern Configuration** tab.
- ___ 2. Add a **Local Queue Manager** option to the **Output queue name** group. You configure this option to be displayed as a check box in the parameter group. If the user selects the check box, the **Queue Manager** parameter is disabled, and the user cannot specify a queue manager name. If the user clears the check box, the **Queue Manager** value can be specified.
- ___ a. In the Groups and Parameters section, select the **Output queue name** group.
- ___ b. Click **Add Parameter** on the right side of the window. The **Add Parameter** dialog box is displayed.
- ___ c. In the **Parameter Display** section, for **Display name**, enter:
Use local queue manager



- ___ d. Click the **Editor** tab. This action opens the parameter editor, where you can specify the parameter type that is presented to the user (for example, check box, text field, drop-down list, and other controls).

- ___ e. For **Parameter editor**, select **Check Box (True/False)**.
For **Default value**, select **false**.

Add Parameter

Configure pattern parameter

Configure the pattern parameter and how it is displayed to pattern users.

Basic Editor Transform Enable

Parameter Editor

Parameter editor: Check Box (True/False)

Type selection: [dropdown]

Default value: false

- ___ f. Click **OK**. The **Use local queue manager** parameter is displayed in the **Output queue name** group.
- ___ g. Double-click the **Queue manager name** parameter (or select it, and then click **Edit** on the right side of the window).
- ___ h. Click the **Enable** tab in parameter editor. Here you specify whether the **Queue manager name** parameter is visible to the user at instantiation time, according to the value of the **Use local queue manager** check box.
- ___ i. In the **Pattern Parameters** section, click **Use local queue manager** (under **Output queue name**).
- ___ j. You want the **Queue manager name** field to be visible if the **Use local queue manager** is set to **false** (that is, cleared).

Set **Test value** to **false**, and then click **Set**.

Pattern Parameters

Groups and Parameters	Parameter ID	Test Value
[-] Input queue name		
Input queue name	pp2	
[-] Output queue name		
Queue manager name	pp1	
Output queue name	pp3	
Use local queue manager	pp4	false

Test value: false [Set]

- ___ k. Click **Use**. This action propagates the test expression to the **Expression Evaluation** section.

- ___ l. Click **Evaluate**. This action evaluates the test expression.

Expression Evaluation	
Expression:	pp:getValue('pp4')
Result:	Enabled (false)

In this case, the result shows that the **Queue manager name** field is **Enabled** because the test (the **Use local queue manager** field) is false (cleared).

- ___ m. Click **OK**.
- ___ 3. Generate the pattern.
- ___ a. Click the **Create Pattern** tab.
- ___ b. If you want, change the values of some of the **Plug-In Information** fields to see how they affect the generation process.
- ___ c. Click **Create Pattern Plug-Ins**. The pattern generation process starts. Remember that it takes several minutes for the generation process to complete.
- ___ d. After the pattern is generated, click **Test Pattern**. After a few moments, a second copy of the toolkit opens again.
- ___ e. When the workspace launcher is displayed, enter a directory name such as C:\workspace\patternrest2, and then click **OK**.
- ___ 4. Test the revised pattern. Follow the steps that are outlined in Part 4, step 4 of this exercise, beginning with step e on page 10-19.

Part 7: Clean up

There are no specific cleanup instructions for this exercise; you did not deploy any components to an integration server. When you switch workspaces for the next exercise, the artifacts you created in this exercise will no longer be visible, unless you choose to import the pattern authoring package.

End of exercise

