IBM Integration Bus V9 Application Development I

(Course code WM665 / VM665)

Student Exercises

ERC 1.0

Authorized

Training

WebSphere Education

August 2013 edition

Contents

Trademarks

Exercises description

This course includes the following exercises:

- Exercise 1: Importing and testing a message flow
  - Import a project
  - Open a simple message flow
  - Examine the message flow components and properties
  - Use the Test Client to test a simple message flow
- Exercise 2: Implementing a message flow pattern
  - Select and generate a message flow application from an IBM Integration Bus pattern
  - Import a message model
  - Build and deploy a broker archive file
  - Test the message flow
- Exercise 3: Analyzing runtime error scenarios
  - Determine the location of a message when an exception occurs in a message flow
- Exercise 4: Using problem determination tools
  - Enable user traces and system traces, and retrieve the collected trace data
  - Add a Trace node to a message flow
  - Use the Debugger view to step through a message flow application
  - Use RFHUtil to send test data, receive message output, and display message data
  - Configure and use the Test Client Component Trace
  - Examine the IBM Integration Bus logs and system logs to diagnose problems
- Exercise 5: Implementing a message flow
  - Create a message flow
  - Use a Compute node or JavaCompute node in a message flow to transform a message
- Exercise 6: Adding flow control to a message flow
  - Use the Route node to control message processing

  - Wire a Failure terminal and Catch terminals to handle exceptions
- Exercise 7: Implementing explicit error handling
  - Implement a generic error handling routine in the form of a subflow
  - Use a ResetContentDescriptor node to force the message to be reparsed according to the parser domain that is specified in the node properties
  - Use the Debug perspective to examine the ExceptionList
- Exercise 8: Implementing a message model
  - Create a DFDL message definition schema file
  - Configure the logical and physical properties for the message model elements
  - Test a DFDL schema by parsing test input
  - Test a DFDL schema by serializing test output data
  - Write ESQL or Java transformations to create a reply message
  - Add an MQReply node to a message flow to send a response to the originator of the input message
  - Configure the Test Client for WebSphere MQ headers and multiple output messages
- Exercise 9: Referencing a database in a map
  - Discover database definitions
  - Add a database node to a message flow
  - Create the logic to store a message in a database
  - Use the Graphical Data Mapping editor to map message elements
  - Reference an external database when mapping message elements
- Exercise 10: Creating a runtime-aware message flow
  - Configure nodes, message flows, integration servers, and integration node properties
  - Define, configure, and use user-defined properties to provide runtime awareness within a message flow application
  - Query various runtime attributes to allow the message flow to display runtime data and alter the flow of message processing

Exercise 1. Importing and testing a message flow

What this exercise is about

In this exercise, you are introduced to the IBM Integration Bus development environment. To help you familiarize yourself with the IBM Integration Toolkit views and navigator, you import a simple message flow project and examine the message flow components and properties. You also use the Test Client to test the message flow.

What you should be able to do

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

- Import a project
- Open a simple message flow
- Examine the message flow components and properties
- Use the Test Client to test a simple message flow

Introduction

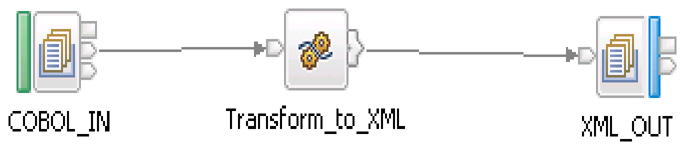
In this exercise, you import an application that transforms the message contents from COBOL to XML.

The following COBOL Copybook defines the COBOL document:

```
01 CUSTOMER-COMPLAINT.
      10 VERSION         PIC 9.
      10 CUSTOMER-NAME.
         15 N-FIRST      PIC X(10).
         15 N-LAST       PIC X(10).
      10 CUSTOMER-ADDRESS.
         15 A-LINE   PIC X(20) OCCURS 2 TIMES.
         15 TOWN         PIC X(10).
         15 ZIP          PIC X(10).
         15 COUNTRY      PIC X(2).
      10 COMPLAINT.
         15 C-TYPE       PIC X(10).
         15 C-REF        PIC X(10).
         15 C-TEXT       PIC X(200).
```

The message flow contains three nodes.

- An MQInput node (that is named "`COBOL_IN`") retrieves the message that contains the COBOL data from a WebSphere MQ input queue that is named `COBOL_IN`.
- A Compute node (that is named "`Transform_to_XML`") transforms the COBOL data to XML.
- An MQOutput node (that is named "`XML_OUT`") puts the transformed message onto a WebSphere MQ output queue that is named "`XML_OUT`".



Requirements

This lab requires the following elements:

- A lab environment with WebSphere MQ and IBM Integration Bus
- The IBM Integration Bus default configuration
- The lab files in `C:\Labs\Lab01-TestSimpleFlow`

How to use the course exercise instructions

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

Note

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

For example:

__ 1. Create a user account named `IIBADMIN`.

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

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

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

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

Logging on

To log on to your lab computer, use the user name and password that are given to you by your instructor. The user name and password for the host computer differs depending upon how the computers are configured at your site.

For this course, log on with the following credentials:

- User name: `IIBAdmin`
- Password: `web1sphere`

Exercise instructions

Part 1: Start the IBM Integration Toolkit

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

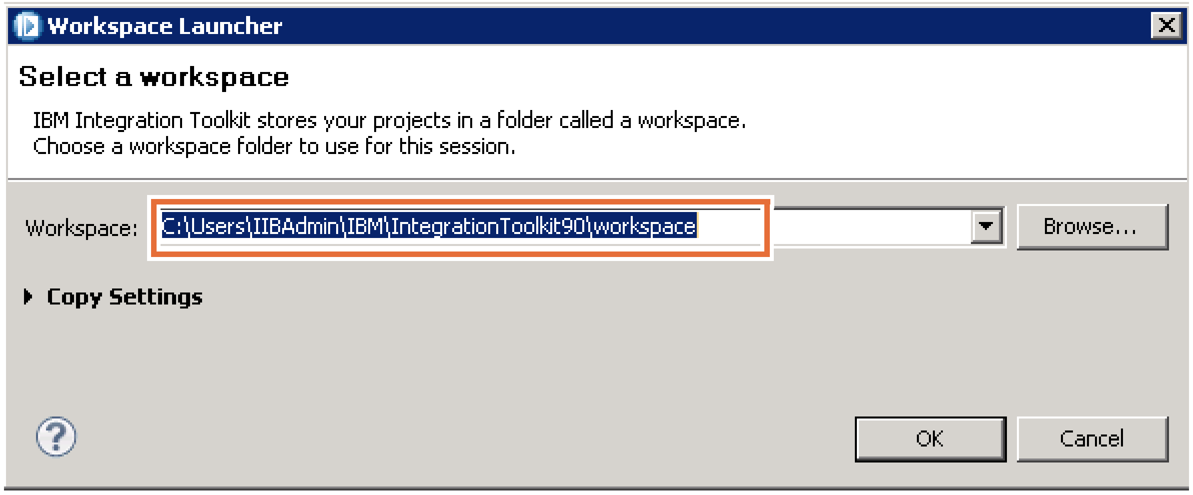
__ 1. Start the IBM Integration Toolkit by double-clicking the **Integration Toolkit9.0.0.0** shortcut on the desktop.

Note

You can also start the IBM Integration Bus Toolkit by selecting **Start** > **All Programs** > **IBM Integration Toolkit** > **IBM Integration Toolkit 9.0.0.0** > **Integration Toolkit 9.0.0.0** from the Windows desktop.

After several moments, the Toolkit starts.

__ 2. When the **Workspace Launcher** is displayed, accept the default workspace of `C:\Users\IIBAdmin\IBM\IntegrationToolkit90\workspace` and then click **OK**.


Workspace Launcher

Select a workspace

IBM Integration Toolkit stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\IIBAdmin\IBM\IntegrationToolkit90\workspace

After several moments, the **Welcome** page is displayed.

__ 3. In the upper right corner of the page, click **Go to the Integration Toolkit** to go to the Integration Development perspective.

The Integration Development perspective is displayed.

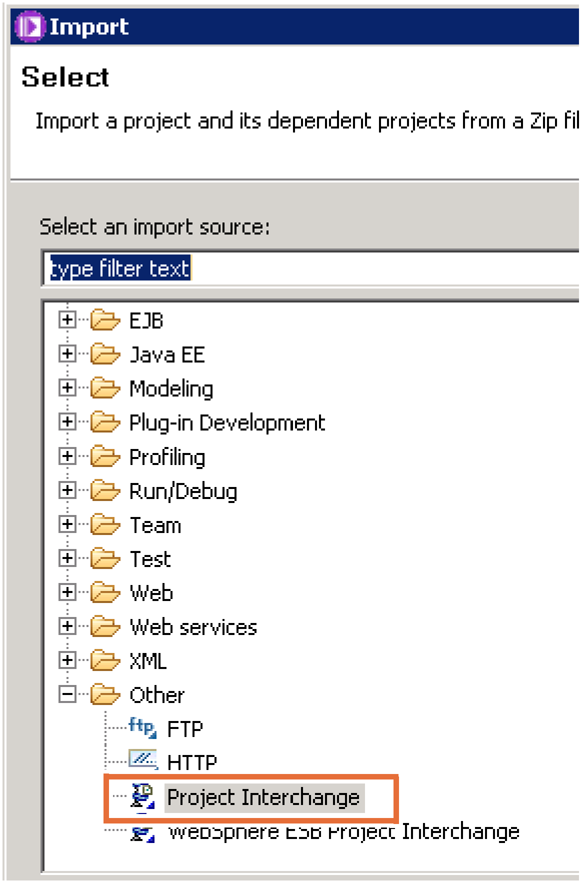
Part 2: Import a project interchange file

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

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

__ 2. In the **Import** source selection, expand **Other** by clicking the plus sign (**+**) in front of it.

__ 3. Click **Project Interchange,** and then click **Next**.

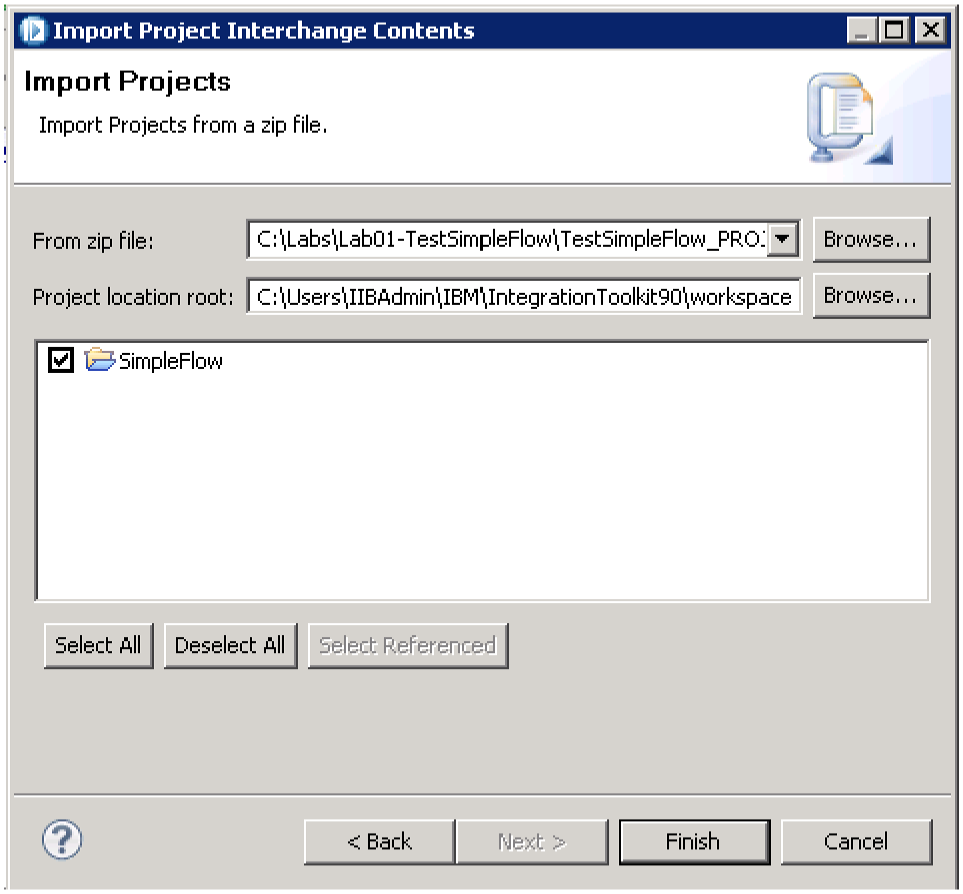


__ 4. To the right of **From zip file**, click **Browse** to locate the project interchange file.

__ 5. Browse to the `C:\Labs\Lab01-TestSimpleFlow` directory, select the file `TestSimpleFlow_PROJECTS.zip` file, and then click **Open**.

__ 6. Ensure that the **Project location root** is set to the current workspace of `C:\Users\IIBAdmin\IBM\IntegrationToolkit90\workspace`.

__ 7. This project interchange file contains one application that is named **SimpleFlow**. Ensure that it is selected and then click **Finish**.



Import Project Interchange Contents

Import Projects

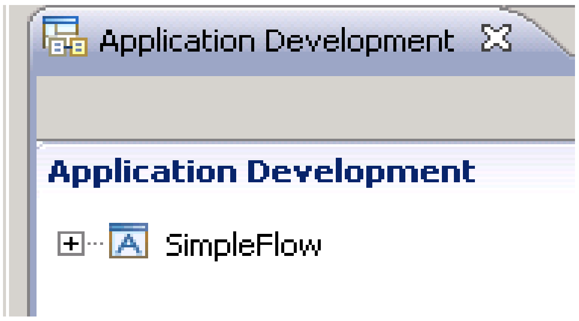
Import Projects from a zip file.

From zip file: C:\Labs\Lab01-TestSimpleFlow\TestSimpleFlow_PRO:

Project location root: C:\Users\IIBAdmin\IBM\IntegrationToolkit90\workspace

☑ SimpleFlow

__ 8. Verify that the **SimpleFlow** application is imported into the Application Development navigator.

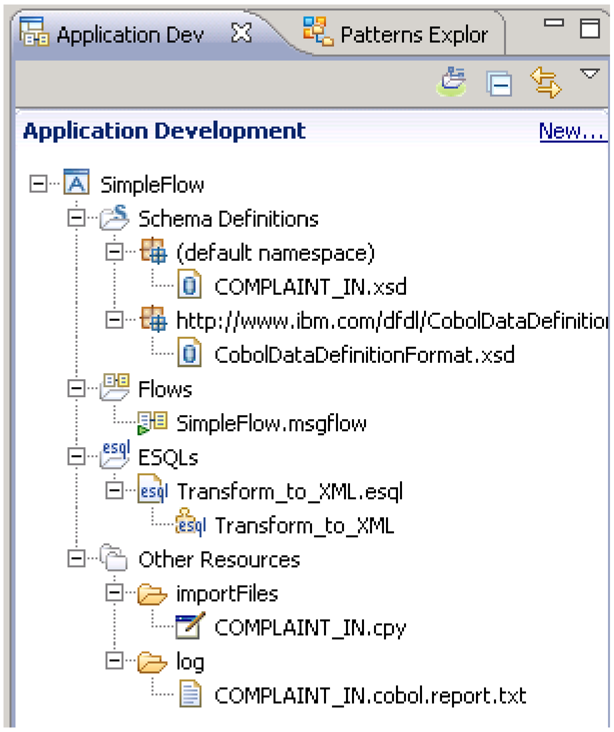


Part 3: Examine the message flow components and properties

__ 1. In the Application Development navigator, expand the application artifacts.

The figure shows the artifacts that are included in the **SimpleFlow** application. The application includes:

- The message flow that transforms the data (`SimpleFlow.msgflow`)
- The Compute node ESQL code that transforms the data from COBOL to XML (`Transform_to_XML.esql`)
- The DFDL model of the COBOL Copybook input data (`COMPLAINT_IN.xsd`) and the helper schema (`CobolDataDefinitionFormat.xsd`)
- Other resources such as the COBOL Copybook file that the DFDL schema defines.



```
SimpleFlow
  Schema Definitions
    (default namespace)
      COMPLAINT_IN.xsd
    http://www.ibm.com/dfdl/CobolDataDefinitio
      CobolDataDefinitionFormat.xsd
  Flows
    SimpleFlow.msgflow
  ESQLs
    Transform_to_XML.esql
      Transform_to_XML
  Other Resources
    importFiles
      COMPLAINT_IN.cpy
    log
      COMPLAINT_IN.cobol.report.txt
```

__ 2. Double-click the message flow **SimpleFlow.msgflow** in the navigator to open it in the Message Flow editor.

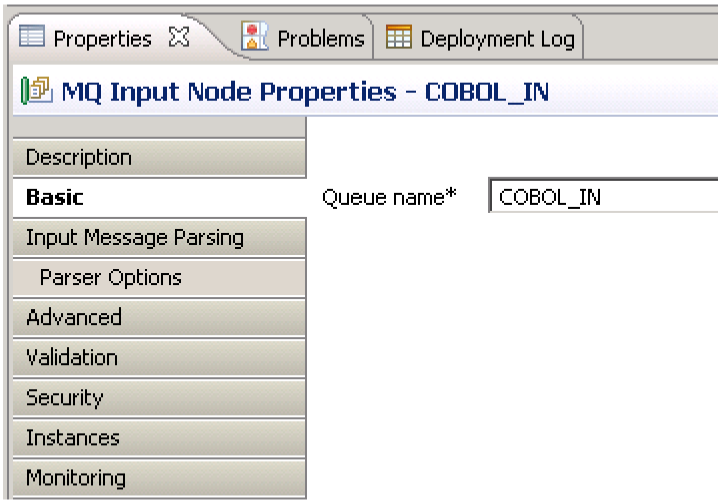
__ 3. The message flow contains three nodes:

- An MQInput node that is named `COBOL_IN`
- A Compute node that is named `Transform_to_XML`
- An MQOutput node that is named `XML_OUT`

In the Message Flow editor, double-click the MQInput node that is named `COBOL_IN`. The properties of the node are shown in the **Properties** view near the bottom of the page.

Note

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



Queue name* COBOL_IN

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

__ 4. Click each of the **Properties** tabs to examine the node properties.

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

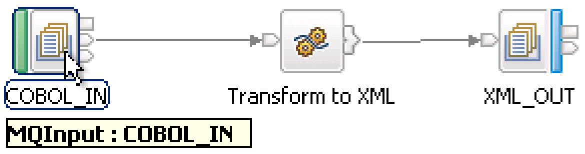
__ a. What is the queue name for the MQInput node? ____________________

__ b. What is the name of the ESQL module in the Compute node?

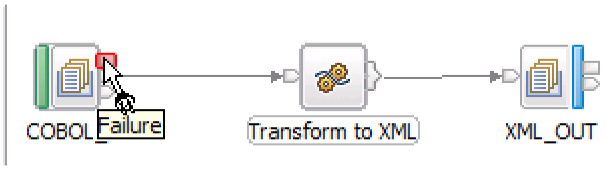
____

__ c. What is the queue name for the MQOutput node? ____________________

__ 6. In the Message Flow editor, hover the mouse pointer over each of the nodes. The node type and node name are displayed in the format `NodeType : NodeName`.

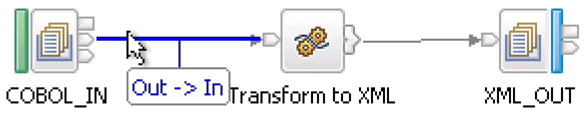


__ 7. In the Message Flow editor, hover the mouse pointer over the node terminals to display the terminal names.



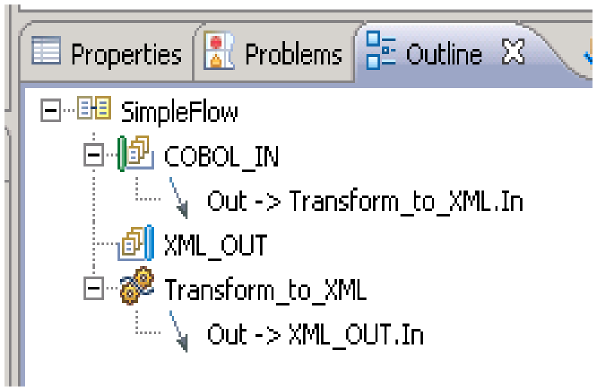
__ 8. In the Message Flow editor, hover the mouse pointer over the wires that connect the nodes to display the connection information.

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



__ 9. Click the **Outline** tab (in the same view as the **Integration Nodes** tab) to display the connection information.

__ 10. Expand the **COBOL_IN** and **Transform_to_XML** node entries in the **Outline** tab.



```
SimpleFlow
  COBOL_IN
    Out -> Transform_to_XML.In
  XML_OUT
  Transform_to_XML
    Out -> XML_OUT.In
```

The **Outline** tab shows that the **Out** terminal of the MQInput node that is named `COBOL_IN` is wired to the **In** terminal of the Compute node that is named `Transform_to_XML` as denoted by `Transform_to_XML.In`.

The **Out** terminal of the Compute node that is named `Transform_to_XML` is wired to the **In** terminal of the MQOutput node that is named `XML_OUT` as denoted by `XML_OUT.In`.

Part 4: Test with the Test Client

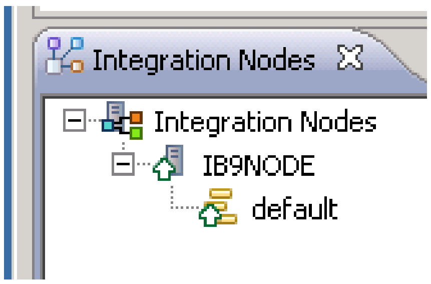
In this part of the exercise, you run the message flow by using the IBM Integration Toolkit Test Client and a test message.

__ 1. The IBM Integration Toolkit Default Configuration wizard created a queue manager, integration node, and integration server that can be used for development. Verify that the integration node and the integration server are running.

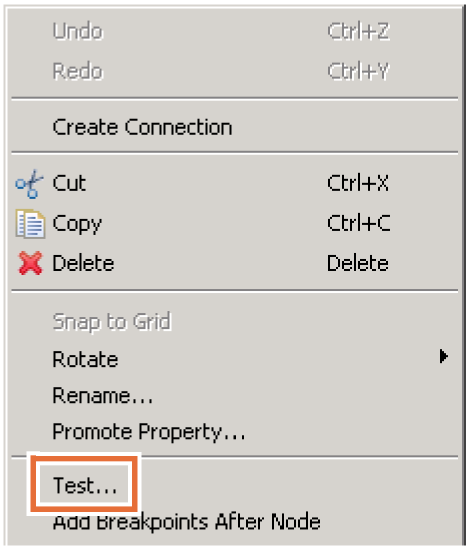
__ a. In the lower left of the Application Development perspective, click the **Integration Nodes** tab.

__ b. Verify that the integration node that is named **IB9NODE** is running as indicated by the green up arrow. If the integration node is not running, start it by right-clicking **IB9NODE** and then selecting **Start**.

__ c. Verify that the integration server that is named **default** is running as indicated by the green up arrow. If the integration server is not running, start it by right-clicking **default** and then selecting **Start**.



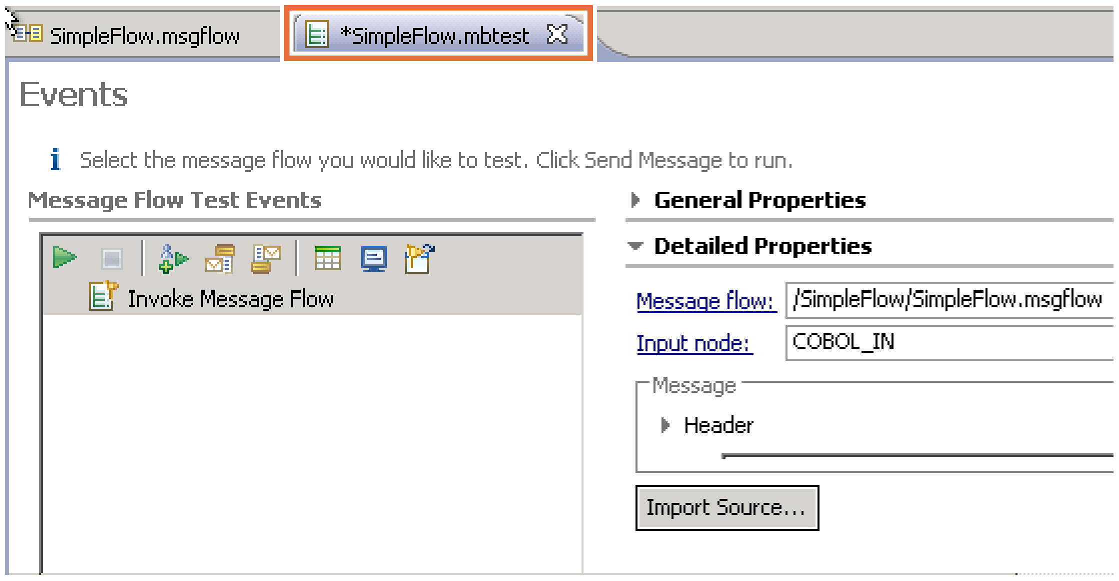
__ 2. Start the Test Client by right-clicking the MQInput node that is named `COBOL_IN` in the Message Flow editor, and then selecting **Test** from the menu.



__ 3. You receive a notification message that an application owns the message flow and that the owning application will be deployed.

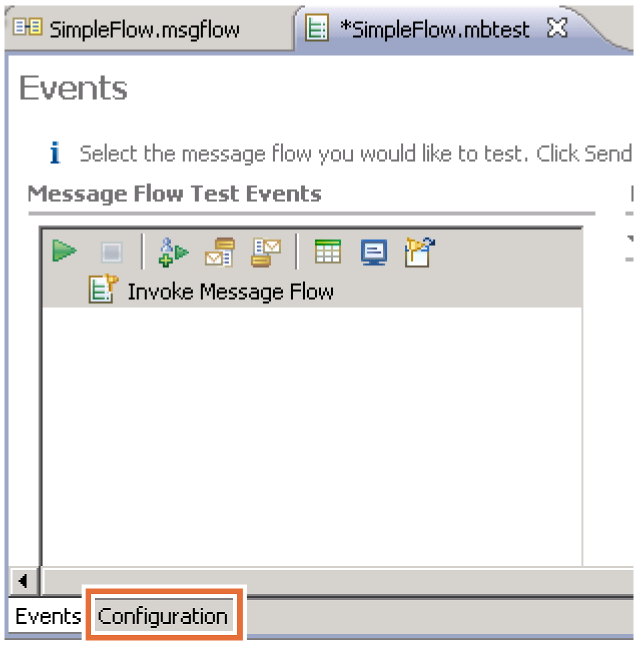
Click **OK**.

__ 4. After a few moments, the Test Client perspective is shown. The Test Client creates a file that is named `SimpleFlow.mbtest` that you use to configure and run the message flow.

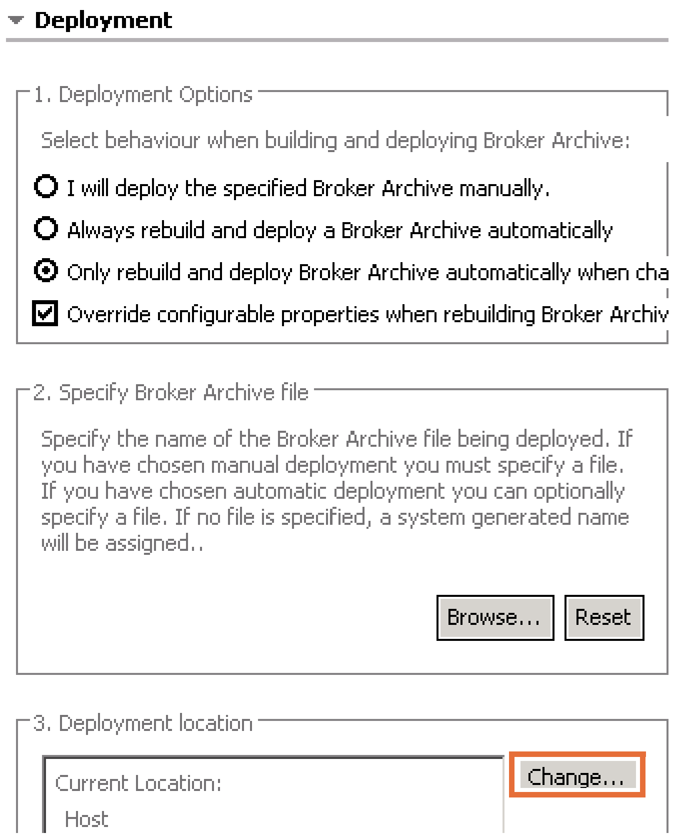


__ 5. Specify the integration node and integration server to use for the test.

__ a. Click the **Configuration** tab on the **SimpleFlow.mbtest** view.



__ b. Under the **Deployment** header, in the **Deployment location** section, click **Change**.



Deployment

1. Deployment Options

Select behaviour when building and deploying Broker Archive:

- ○ I will deploy the specified Broker Archive manually.
- ○ Always rebuild and deploy a Broker Archive automatically
- ● Only rebuild and deploy Broker Archive automatically when cha
- ☑ Override configurable properties when rebuilding Broker Archiv

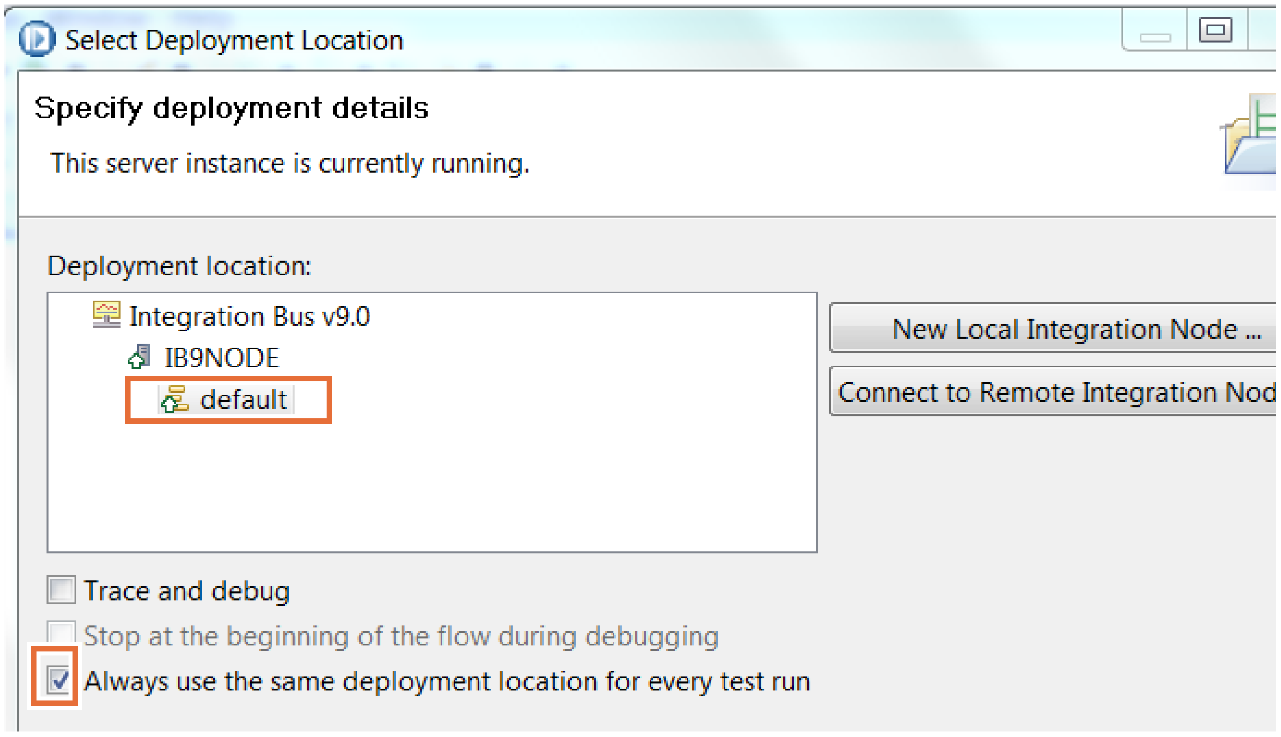
2. Specify Broker Archive file

Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

3. Deployment location

Current Location:
Host

__ c. Under **Deployment location**, click the **default** integration server, and ensure that **Always use the same deployment location for every test run** is selected.



Select Deployment Location

Specify deployment details

This server instance is currently running.

Deployment location:

- Integration Bus v9.0
  - IB9NODE
    - default

New Local Integration Node ...
Connect to Remote Integration Nod

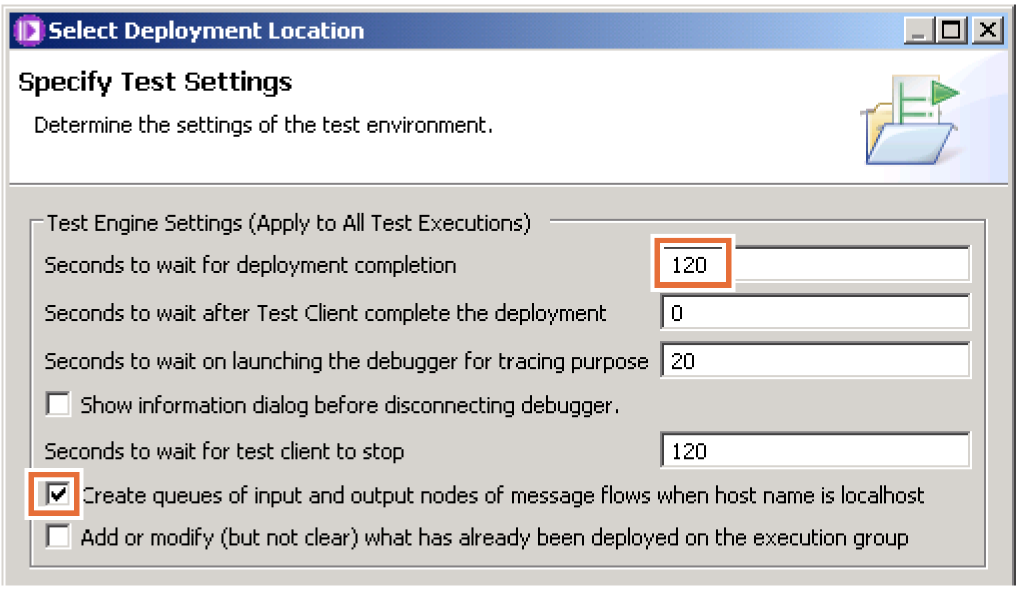
- ☐ Trace and debug
- ☐ Stop at the beginning of the flow during debugging
- ☑ Always use the same deployment location for every test run

__ d. Click **Next**. The **Specify Test Settings** page is displayed.

__ e. Change **Seconds to wait for deployment completion** to: `120`

__ f. Select **Create queues of input and output nodes of message flows when host name is localhost**.

This action automatically creates the queues that are required to run the SimpleFlow message flow by using the Test Client in the lab environment.


Select Deployment Location

Specify Test Settings

Determine the settings of the test environment.

Test Engine Settings (Apply to All Test Executions)

Seconds to wait for deployment completion: 120

Seconds to wait after Test Client complete the deployment: 0

Seconds to wait on launching the debugger for tracing purpose: 20

☐ Show information dialog before disconnecting debugger.

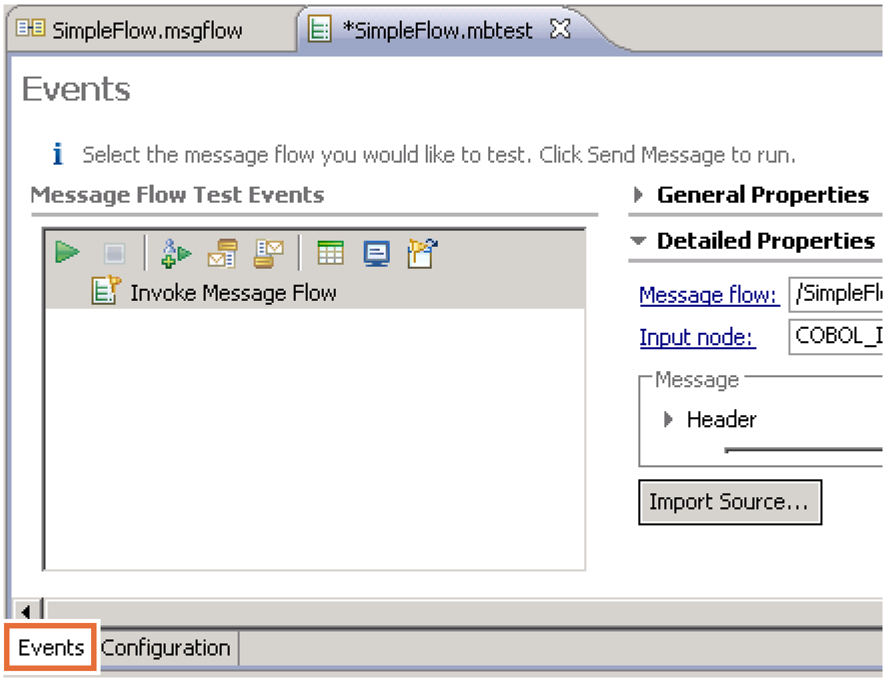
Seconds to wait for test client to stop: 120

☑ Create queues of input and output nodes of message flows when host name is localhost

☐ Add or modify (but not clear) what has already been deployed on the execution group

__ g. Click **Finish**.

__ 6. Click the **Events** tab on the **SimpleFlow.mbtest** view.

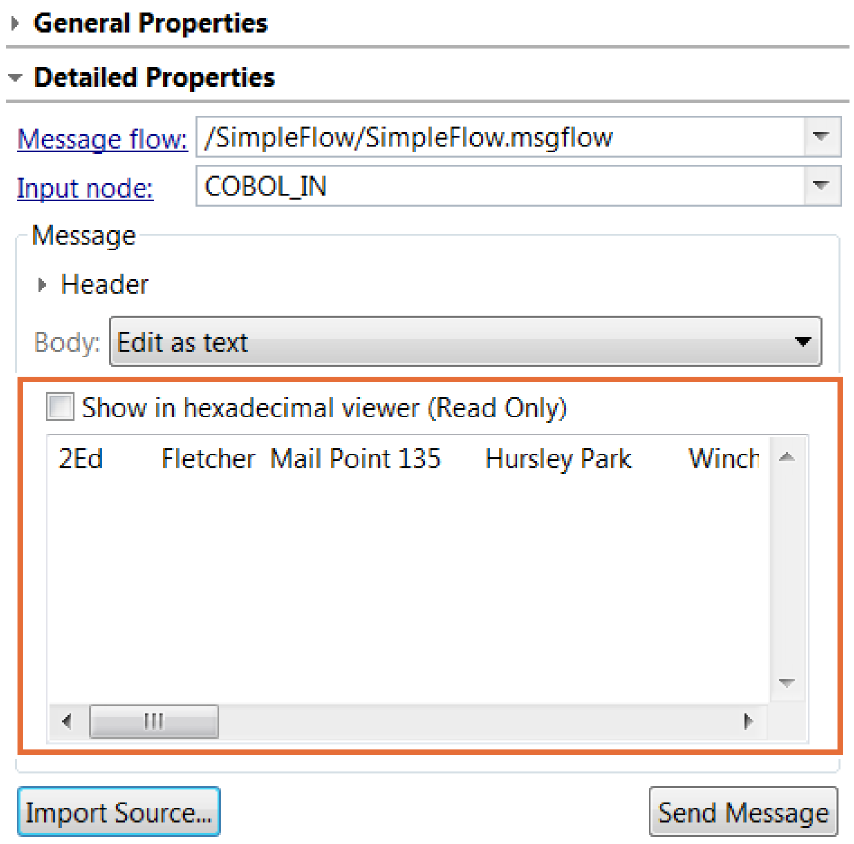


__ 7. The message flow expects a COBOL input file. Import the source data.

__ a. Under **Detailed Properties** in the right pane, click **Import Source**.

__ b. Browse to `C:\Labs\Lab01-TestSimpleFlow\data`, select **Complaint_cwf.txt**, and then click **Open**.

__ c. Under **Detailed Properties**, review the **Message** content. It might be necessary to scroll down or enlarge the view to see this area.


General Properties
Detailed Properties
Message flow: /SimpleFlow/SimpleFlow.msgflow
Input node: COBOL_IN
Message
Header
Body: Edit as text
Show in hexadecimal viewer (Read Only)
2Ed Fletcher Mail Point 135 Hursley Park Winch
Import Source...
Send Message

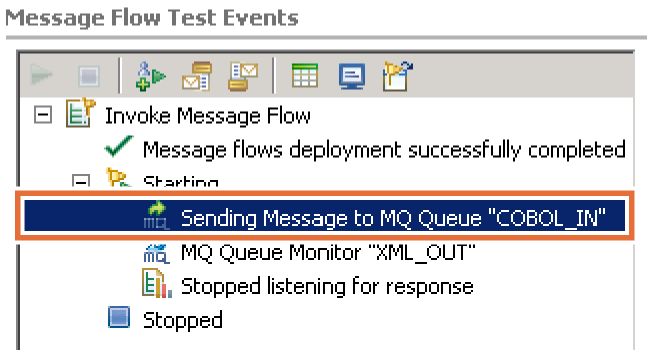
__ 8. Run the test by clicking **Send Message**.

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

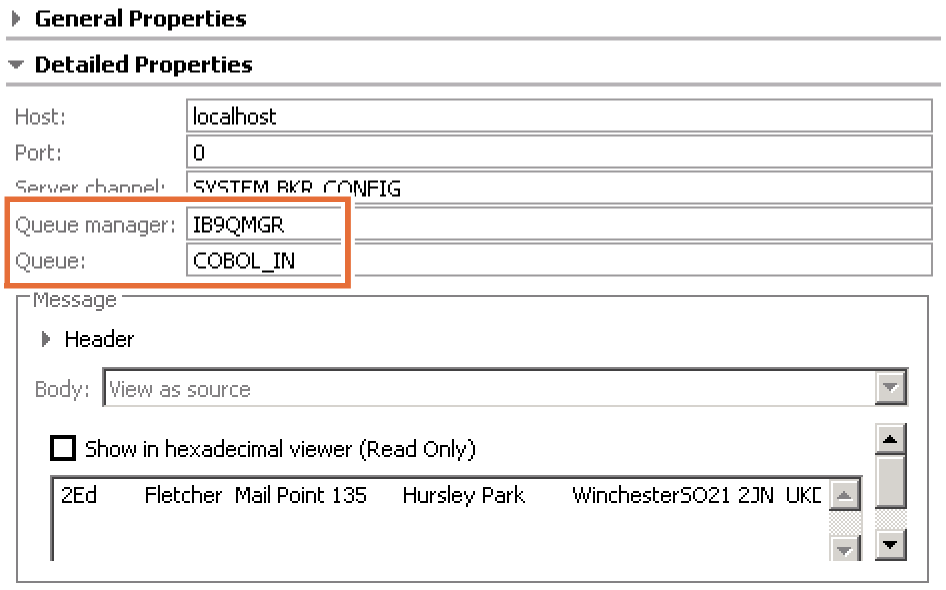
After some moments, the result data and Flow Test Events are displayed.

__ 9. Under the **Message Flow Test Events** pane, click each of the events. As you do so, the details of each event are shown under **Detailed Properties** on the right.

For example, when you click the **Sending Message to MQ Queue "COBOL_IN"** event, you see information such as the queue name, queue manager name, and message body under **Detailed Properties**.


Message Flow Test Events

Invoke Message Flow
Message flows deployment successfully completed
Starting
Sending Message to MQ Queue "COBOL_IN"
MQ Queue Monitor "XML_OUT"
Stopped listening for response
Stopped


General Properties

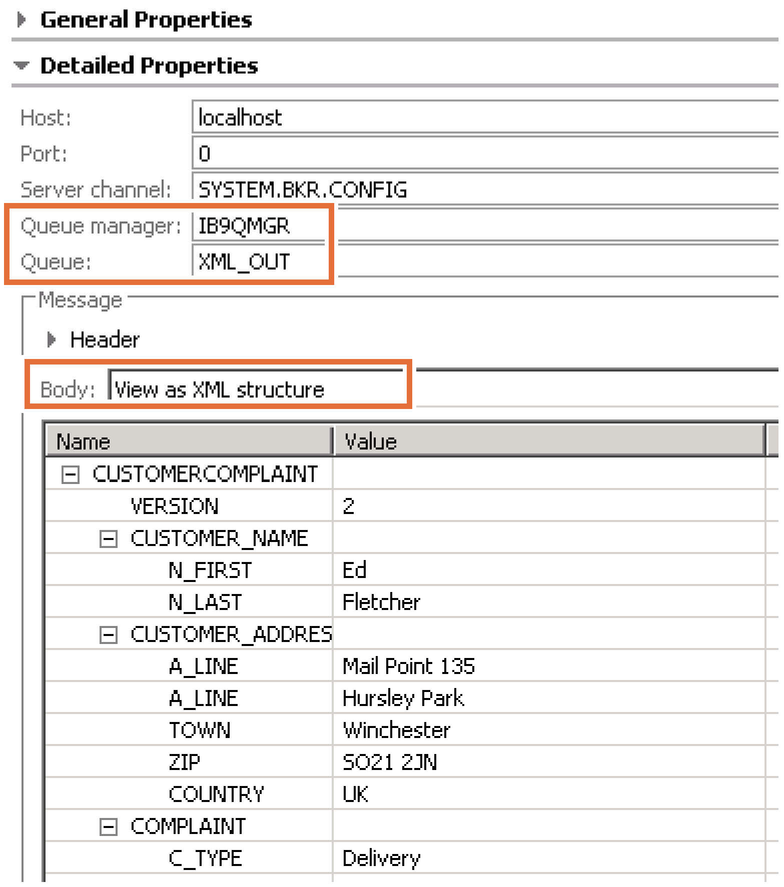
Detailed Properties

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

Message
Header
Body: View as source
Show in hexadecimal viewer (Read Only)

```
2Ed      Fletcher  Mail Point 135     Hursley Park      WinchesterSO21 2JN  UKD
```

When you click the **MQ Queue Monitor "XML_OUT"** event**,** you see the results of the MQOutput node. The queue and queue manager information is again displayed, along with formatted XML message that was written to the queue.

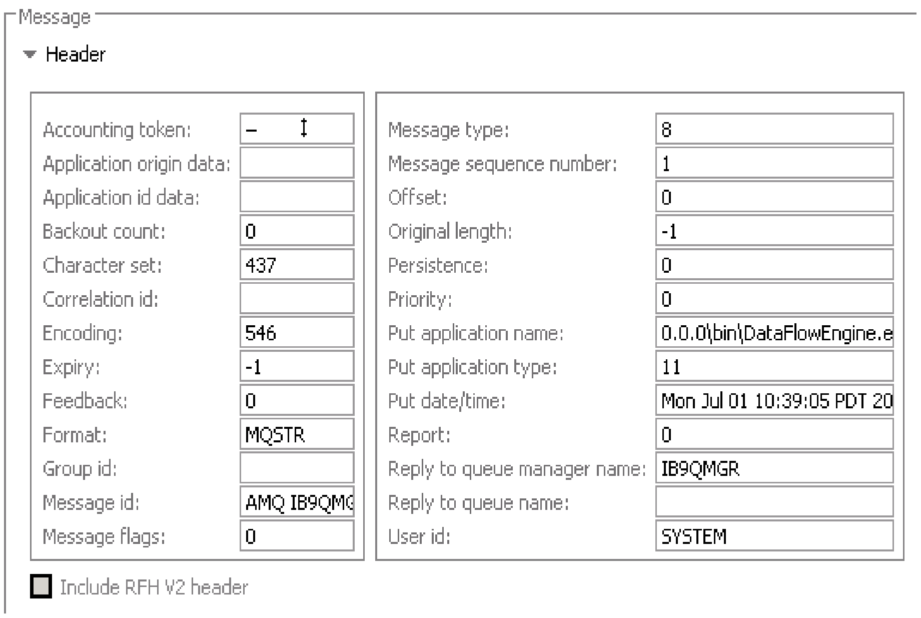


Host: localhost
Port: 0
Server channel: SYSTEM.BKR.CONFIG
Queue manager: IB9QMGR
Queue: XML_OUT

Body: View as XML structure

Name	Value
CUSTOMERCOMPLAINT	
VERSION	2
CUSTOMER_NAME	
N_FIRST	Ed
N_LAST	Fletcher
CUSTOMER_ADDRES	
A_LINE	Mail Point 135
A_LINE	Hursley Park
TOWN	Winchester
ZIP	SO21 2JN
COUNTRY	UK
COMPLAINT	
C_TYPE	Delivery

__ 10. Notice that you can view the WebSphere MQ header information by clicking **Header** in the **Message** area.



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:	437	Persistence:	0
Correlation id:		Priority:	0
Encoding:	546	Put application name:	0.0.0\bin\DataFlowEngine.e
Expiry:	-1	Put application type:	11
Feedback:	0	Put date/time:	Mon Jul 01 10:39:05 PDT 20
Format:	MQSTR	Report:	0
Group id:		Reply to queue manager name:	IB9QMGR
Message id:	AMQ IB9QMG	Reply to queue name:	
Message flags:	0	User id:	SYSTEM

Also, notice that the Test Client automatically recognizes that the output message is XML, so it chooses **View as XML structure** for the **Body** type.

Click the **Body** drop-down arrow to see what other formats can be used to display the output message.

__ 11. Compare the input message that was sent to COBOL_IN, with the output message as displayed by `MQ Queue Monitor XML_OUT`. The Compute node transformed the COBOL input message into XML.

You learn more about message transformation throughout the course.

Note

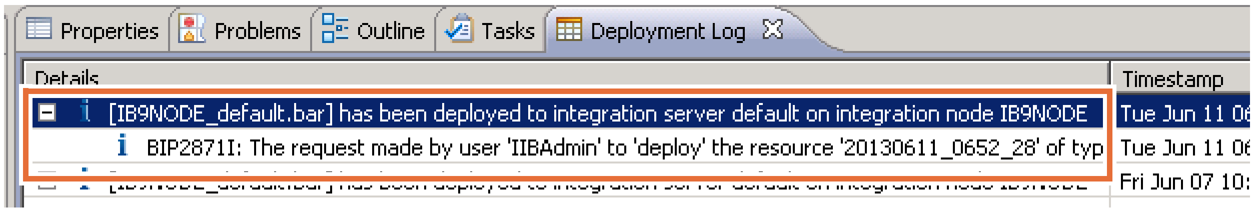
If you use WebSphere MQ Explorer to look at the output queue for the message, you see that the queue is empty because, by default, the Test Client purges the WebSphere MQ queue.

Troubleshooting

If the test does not start, click **Send message** again.

If you do not have any message on XML_OUT, the reason is likely a typing error. Ask your instructor for help. In a subsequent exercise, you learn how to debug your message flows.

__ 12. Examine the **Deployment Log** view near the bottom of the Application Development perspective page. You see an entry in the log for the deployment to the **default** integration server on the **IB9NODE** integration node.



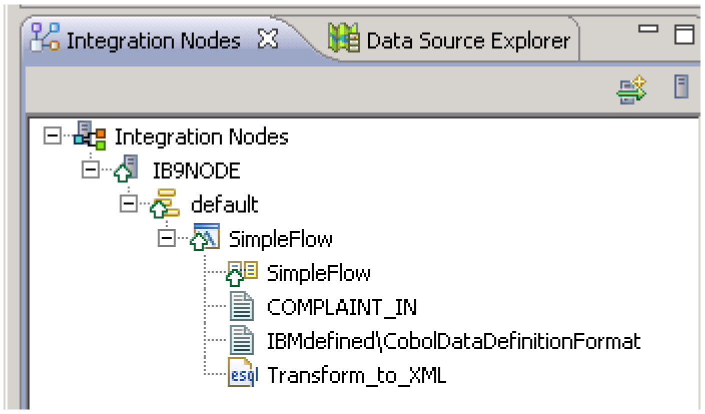
```
[IB9NODE_default.bar] has been deployed to integration server default on integration node IB9NODE    Tue Jun 11 06
    BIP2871I: The request made by user 'IIBAdmin' to 'deploy' the resource '20130611_0652_28' of typ    Tue Jun 11 06
                                                                                                         Fri Jun 07 10:
```

__ 13. Examine the **Integration Nodes** view.

You see the **default** integration server on that **IB9NODE** integration node contains a single application that is named **SimpleFlow**.

Expand the application to view the runtime components that the integration node requires. Hover your mouse pointer over each component to get more information about the component.

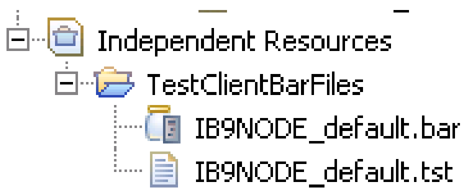
- **SimpleFlow** is the message flow.
- **COMPLAINT_IN** is the DFDL model that defines the COBOL data.
- **IBMdefined\CobolDataDefinitionFormat** is the DFDL “Helper” schema.
- **Transform_to_XML** is the ESQL code that is contained in the Compute node.



__ 14. Examine the Application Development navigator. You should now see a project that is named **TestClientBarFiles** that is under the **Independent Resources** folder.

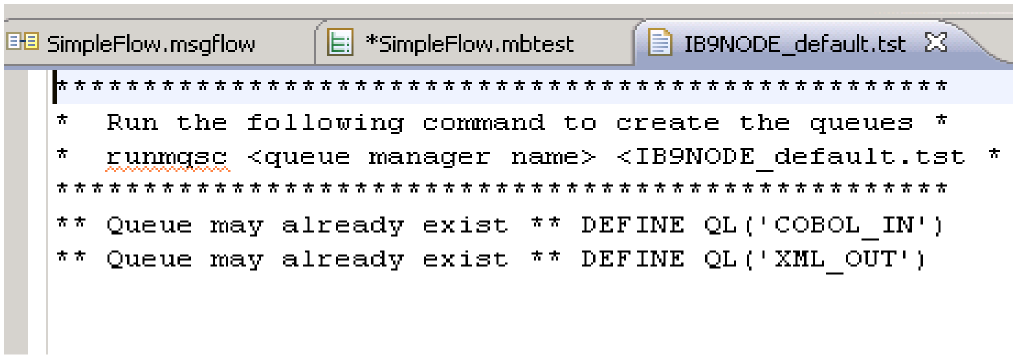
The **TestClientBarFiles** project contains the BAR file (with a `.bar` extension) that the Test Client built and deployed.

Double-click the BAR file to open it in the BAR file editor.



__ 15. The **TestClientBarFiles** project also contains a text file (with a `.tst` extension) that contains the WebSphere MQ commands that the Test Client ran to create the queues that the message flow requires.

Double-click the text file to display the WebSphere MQ commands that the Test Client ran.



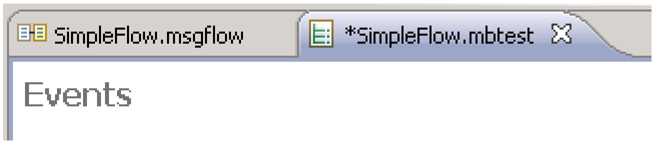
```
**************************************************
*  Run the following command to create the queues *
*  runmqsc <queue manager name> <IB9NODE_default.tst *
**************************************************
** Queue may already exist ** DEFINE QL('COBOL_IN')
** Queue may already exist ** DEFINE QL('XML_OUT')
```

__ 16. To ensure that there are no conflicts with runtime components for other exercises, delete the contents of the **default** integration server.

In the **Integration Nodes** view, right-click the integration server **default** and then select **Delete > All flows and resources**. When prompted, confirm the deletion.

When the deletion is complete, there should not be any objects under the **default** integration server.

__ 17. Close the Integrated Test Client by clicking the **X** on the right side of the **SimpleFLow.mbtest** tab. When you are asked to save changes, click **No**.



__ 18. Close all open editor tabs.

End of exercise

Exercise 2. Implementing a message flow pattern

What this exercise is about

In this exercise, you generate a message flow from a pattern and add the user-specific parameters. You also import a DFDL message model library and add the library reference to the application. Finally, you build and deploy the BAR file, and then test the message flow.

What you should be able to do

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

- Select and generate a message flow application from an IBM Integration Bus pattern
- Import a message model
- Build and deploy a broker archive file
- Test the message flow

Introduction

In this exercise, you generate a message flow from an IBM Integration Bus pattern that generates one WebSphere MQ message for each record in an input file.

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

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

The input file is in the `C:\Labs\Lab02-Pattern\data` directory. The DFDL message model that describes the file is provided for you in a project interchange file.

The pattern can also be generated to support message routing. For this exercise, however, no additional routing capability is necessary.

Requirements

- A workstation with WebSphere MQ and the IBM Integration Bus

- The IBM Integration Bus default configuration
- The lab exercise files in `C:\Labs\lab02-Pattern\`
- The WebSphere MQ queues ERROR and EMPLOYEE_OUT

Exercise instructions

Part 1: Examine the pattern requirements

__ 1. Clean up from the previous exercise and ensure that the environment is ready.

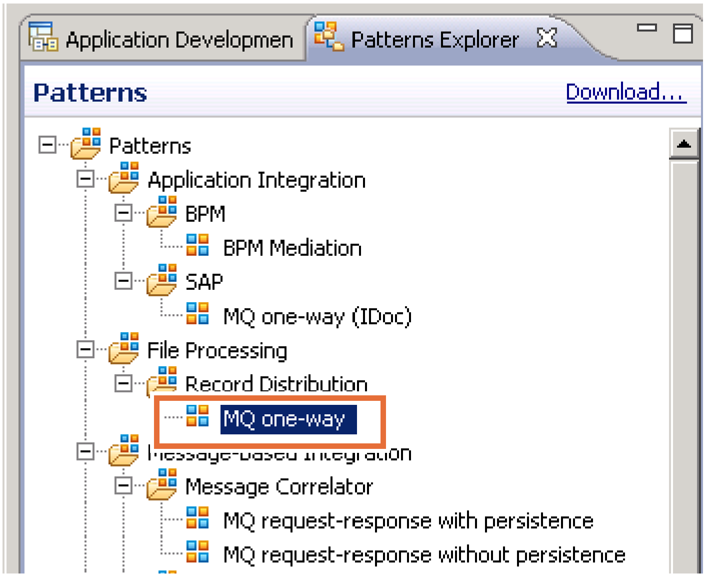
__ a. Start the IBM Integration Toolkit, if it is not already running.

__ b. Close all open editors.

__ c. Remove all flows and resources from the **default** integration server if you did not already remove them.

__ 2. Click the **Patterns Explorer** tab in the Application Development perspective.

__ 3. Under **Patterns > File Processing > Record Distribution**, click **MQ one-way**.

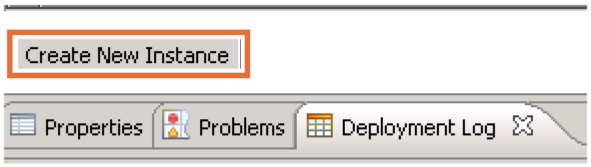


__ 4. Read the description of the pattern in the **Pattern Specification** view.

__ 5. Read each of the sections under **Related tasks**.

Part 2: Generate a pattern instance

__ 1. Click **Create New Instance** at the bottom of the **Pattern Specification** view.



The Configure Pattern Parameters window is displayed. This window is where the parameters are specified that are used to generate the instance of the pattern. In the left pane, you supply the configuration values. The right pane provides help information for each of the parameters.

__ 2. For **Pattern instance name**, enter `EmployeeRecordToQ` and then click **OK**. The **Configure Pattern Parameters** pane is displayed.

__ 3. Expand the **Input file** section. This section identifies the input file to be processed.

  __ a. For **Directory of input file**, enter: `C:\Labs\Lab02-Pattern`

  __ b. For **File pattern**, enter: `EMPLOYEE.TXT`

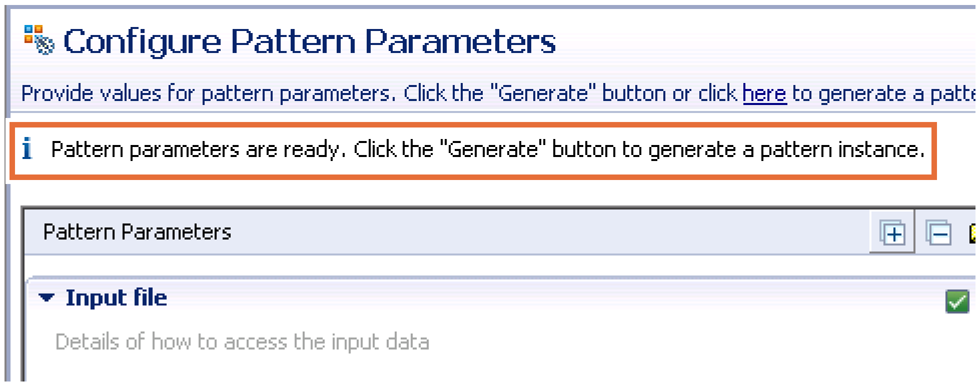
__ 4. Expand the **Input record processing** section. This section describes the characteristics of the input data file. Change **Record length** to `60`.

__ 5. Expand the **No routing** section. This section defines the parameters for output to a single destination.

  __ a. For **Output queue manager**, enter: `IB9QMGR`

  __ b. For **Output queue**, enter: `EMPLOYEE_OUT`

__ 6. Verify that the **Pattern parameters are ready** message is displayed at the top of the view. This message indicates that the required parameters are supplied.



Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click here to generate a patte

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

Input file

Details of how to access the input data

__ 7. Click **Generate** at the bottom of the window.

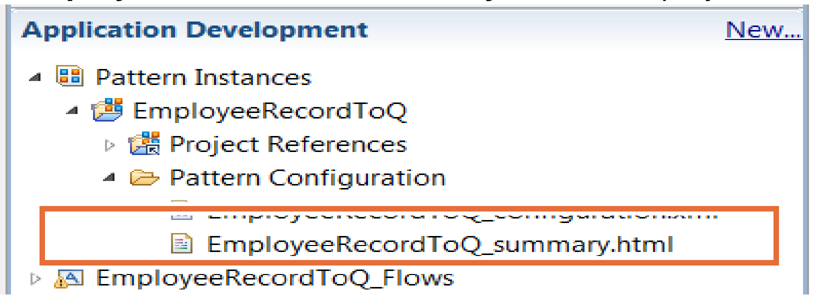
The components are automatically created in a new working set. When generation is complete, a summary of the actions that were taken is displayed.

Note

The summary page that is displayed after the pattern is generated might be a blank page. If a blank page is displayed:

1) Close the tab.

2) Under the **Pattern Configuration** folder in the Application Development navigator, double-click **EmployeeRecordToQ_summary.html** to display the summary page.



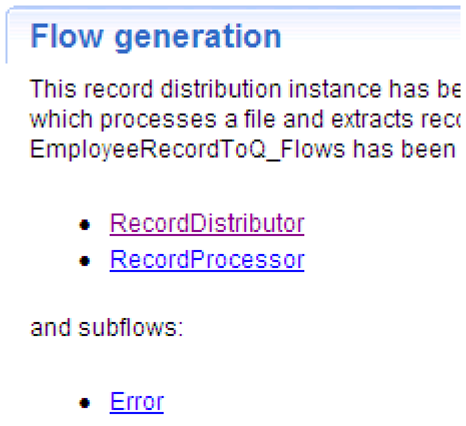
__ 8. Review the **Tasks to complete** section on the **EmployeeRecordToQ_summary.html** tab. This section describes the steps that must be taken before the generated pattern instance can run.

- Based on the parameters you specified, the pattern instance that was generated requires two queues: ERROR and EMPLOYEE_OUT.

  Open WebSphere MQ Explorer and verify that the local queues ERROR and EMPLOYEE_OUT exist; if they do not, create them.

- The pattern instance must be added to a BAR file and deployed to an integration server. You do this step later in this exercise.

__ 9. Examine the flows. Several flows were generated from the pattern. To view the flows, click the links that are shown in the **Summary** pane.



Flow generation

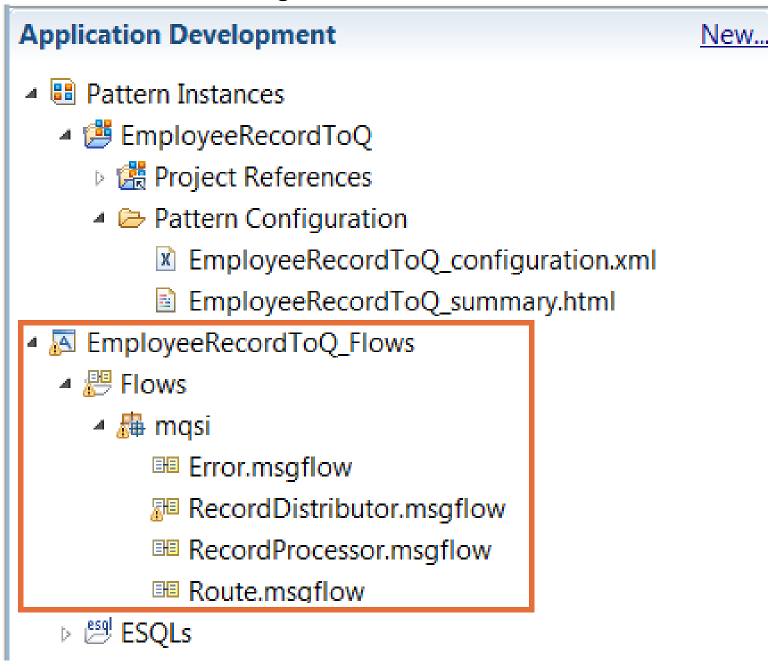
This record distribution instance has be
which processes a file and extracts rec
EmployeeRecordToQ_Flows has been

- RecordDistributor
- RecordProcessor

and subflows:

- Error

You can also view the flows in the Application Development navigator by expanding **EmployeeRecordToQ_Flows > Flows > mqsi** and then double-clicking each of the flows to open them in the Message Flow editor.



Application Development New...

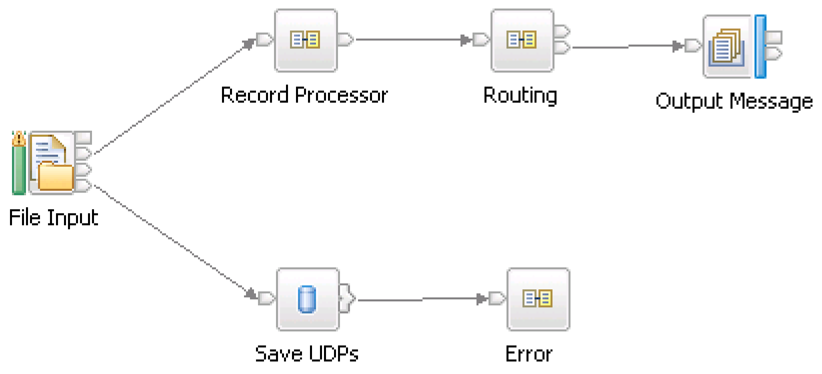
- Pattern Instances
  - EmployeeRecordToQ
    - Project References
    - Pattern Configuration
      - EmployeeRecordToQ_configuration.xml
      - EmployeeRecordToQ_summary.html
  - EmployeeRecordToQ_Flows
    - Flows
      - mqsi
        - Error.msgflow
        - RecordDistributor.msgflow
        - RecordProcessor.msgflow
        - Route.msgflow
    - ESQLs

The main flow is `RecordDistributor.msgflow`. It consists of the File Input node, three subflows, a Database node, and an Output Message flow.

The other message flow files are subflows that handle recording processing, routing, and errors.

Note

A subflow is a message flow that another message flow references. Subflows are described in detail later in the course.



Part 3: Import the message model for the input file

To run the generated pattern, you need a message model that can be used during message processing. A DFDL message model is already created and stored in a project interchange file in the `C:\Labs\Lab02-Pattern\Resources` directory.

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

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

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

__ b. Expand **Other**, select **Project Interchange**, and then click **Next**.

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

__ d. Browse to the `C:\Labs\Lab02-Pattern\Resources` directory.

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

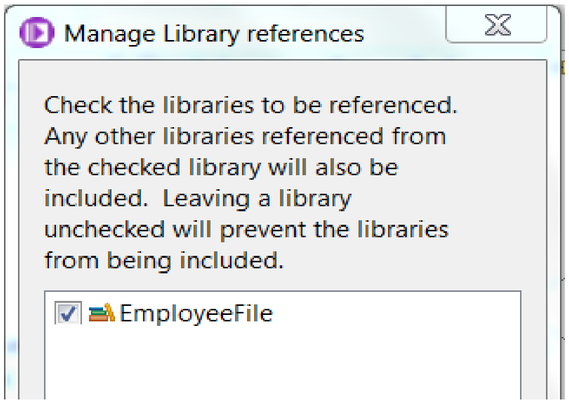
__ f. The project interchange file contains one artifact, which is a library that is named `EmployeeFile`. Ensure that it is selected and then click **Finish**.

__ g. Verify that `EmployeeFile` is listed in the Application Development navigator.

__ 2. Change the `EmployeeRecordToQ_Flows` project to reference the `EmployeeFile` library.

__ a. In the Application Development navigator, right-click the **EmployeeRecordToQ_Flows** folder and then select **Manage Library References** from the menu.

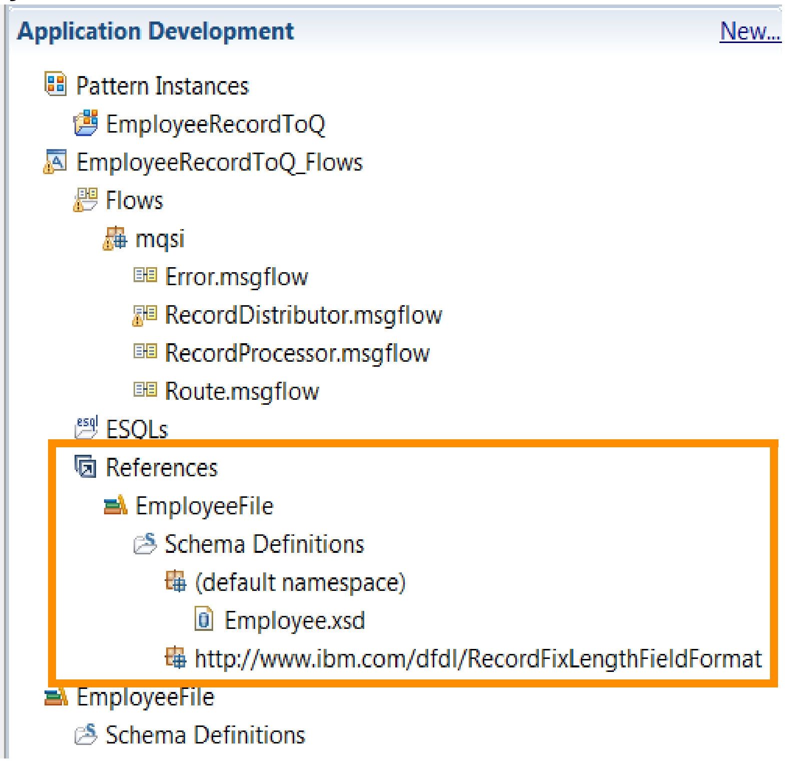
__ b. Select **EmployeeFile** and then click **OK**.


Manage Library references

Check the libraries to be referenced. Any other libraries referenced from the checked library will also be included. Leaving a library unchecked will prevent the libraries from being included.

☑ EmployeeFile

__ c. The **EmployeeFile** library should now be displayed under the **EmployeeRecordToQ_Flows** folder in the **References** folder.


```
Application Development                                    New...
  Pattern Instances
    EmployeeRecordToQ
  EmployeeRecordToQ_Flows
    Flows
      mqsi
        Error.msgflow
        RecordDistributor.msgflow
        RecordProcessor.msgflow
        Route.msgflow
    ESQLs
    References
      EmployeeFile
        Schema Definitions
          (default namespace)
            Employee.xsd
          http://www.ibm.com/dfdl/RecordFixLengthFieldFormat
  EmployeeFile
    Schema Definitions
```

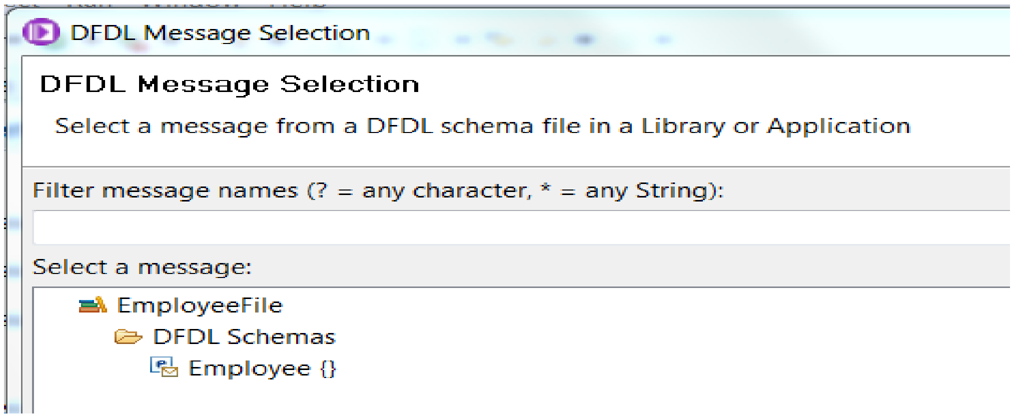
__ 3. Set the File node properties to use the new message model.

__ a. Double-click **RecordDistributor.msgflow** in the Application Development navigator to return to the Message Flow editor window where the message flow is open.

__ b. Right-click the **File Input** node, and then select **Properties** to open the properties, if it is not already open.

__ c. On the **Input Message Parsing** tab of the node properties, select **DFDL** for the **Message domain**.

__ d. For **Message**, click **Browse**, select **Employee {}**, and then click **OK**.



DFDL Message Selection

DFDL Message Selection

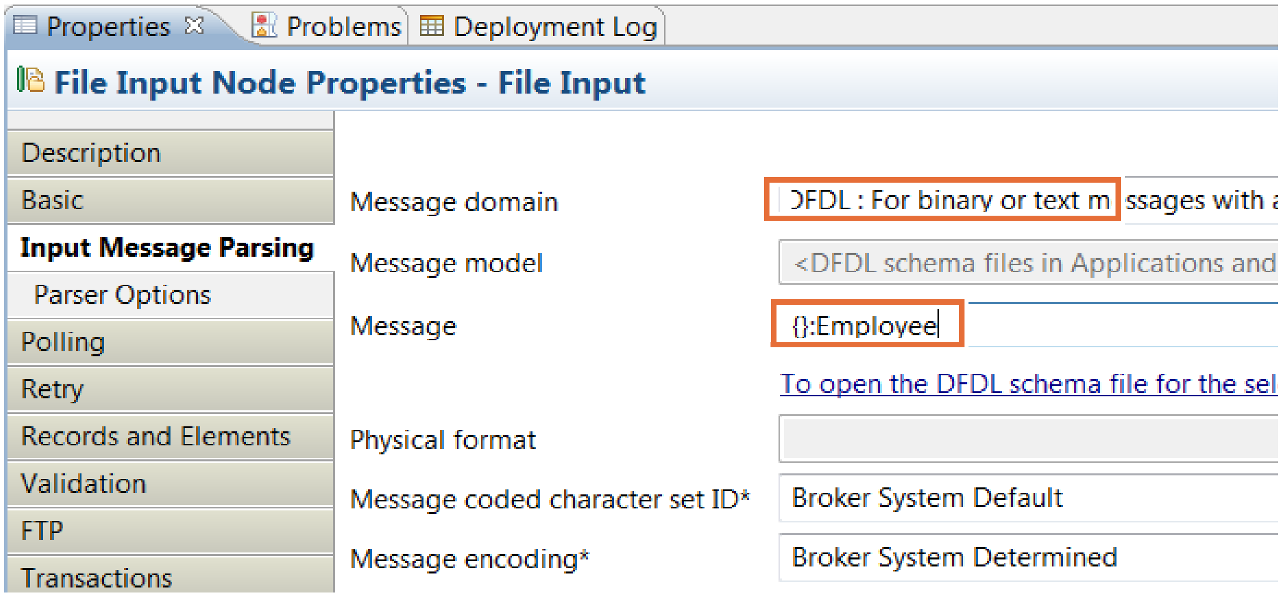
Select a message from a DFDL schema file in a Library or Application

Filter message names (? = any character, * = any String):

Select a message:

EmployeeFile
DFDL Schemas
Employee {}

The Input Message Parsing properties should be displayed as follows:



File Input Node Properties - File Input

Message domain ⊃FDL : For binary or text m ssages with a

Message model <DFDL schema files in Applications and

Message {}:Employee

To open the DFDL schema file for the sel

Physical format

Message coded character set ID* Broker System Default

Message encoding* Broker System Determined

__ 4. Save all changes by clicking **File > Save All** from the toolbar. You can also use **Ctrl + Shift + S** instead.

Part 4: Deploy and test the flow

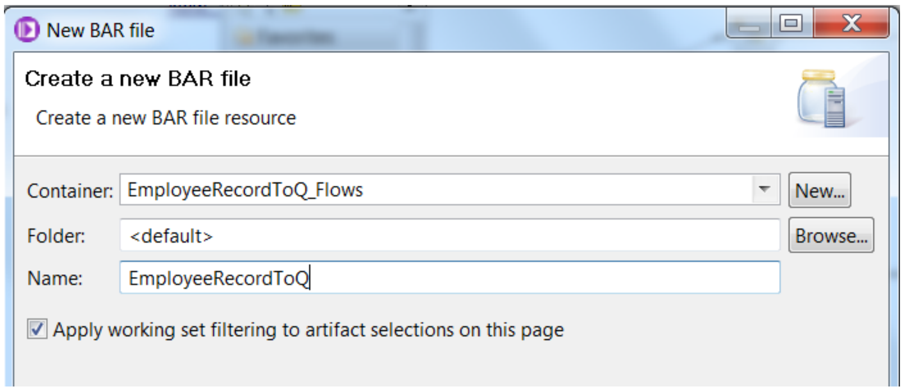
In this part of the exercise, you create a BAR file. You package the message flow application that contains the message flows and message model schemas in the BAR file, and then deploy the BAR file to an integration server for testing.

__ 1. Create a BAR file that is named `EmployeeRecordToQ` in the `EmployeeRecordToQ_Flows` application.

__ a. From the menu bar, select **File > New > BAR file**.

__ b. For **Container**, select **EmployeeRecordToQ_Flows** from the menu.

__ c. For **Name**, enter: `EmployeeRecordToQ`



Create a new BAR file

Create a new BAR file resource

Container: EmployeeRecordToQ_Flows

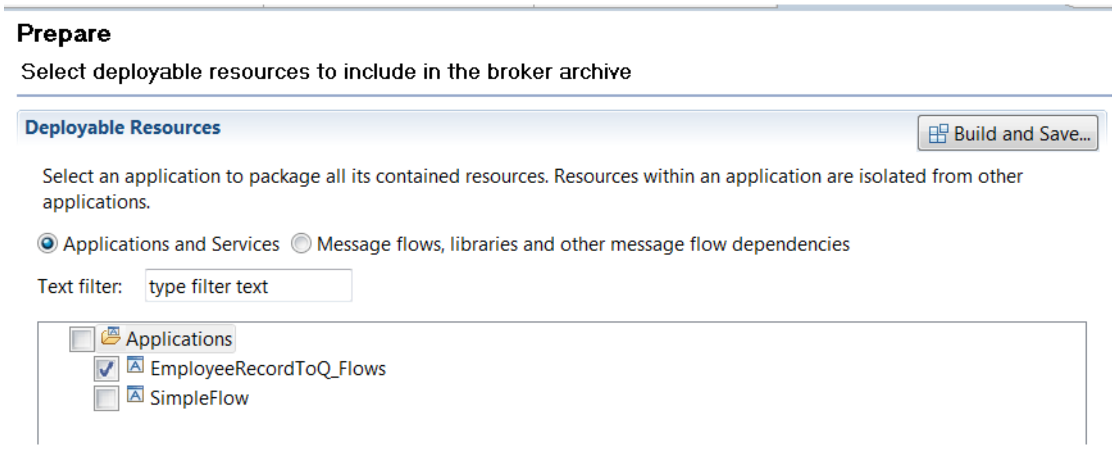
Folder: <default>

Name: EmployeeRecordToQ

✓ Apply working set filtering to artifact selections on this page

__ d. Click **Finish**. The BAR file editor opens on the **Prepare** tab.

__ e. In the **Deployable Resources** section, select the **EmployeeRecordToQ_Flows** application.



Prepare

Select deployable resources to include in the broker archive

Deployable Resources

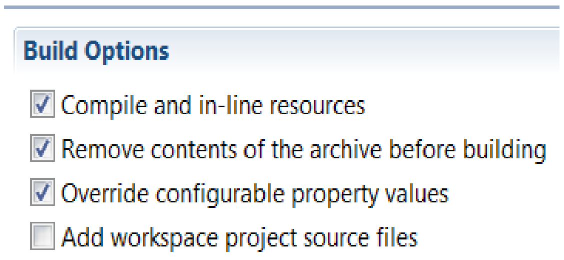
Select an application to package all its contained resources. Resources within an application are isolated from other applications.

● Applications and Services ○ Message flows, libraries and other message flow dependencies

Text filter: type filter text

☐ Applications
✓ EmployeeRecordToQ_Flows
☐ SimpleFlow

___ f. The message flow includes a subflow. To ensure that the subflow is compiled and included in the BAR file, select **Compile and in-line resources** under the **Build options** on the right of the **Prepare** tab.



- [x] Compile and in-line resources
- [x] Remove contents of the archive before building
- [x] Override configurable property values
- [ ] Add workspace project source files

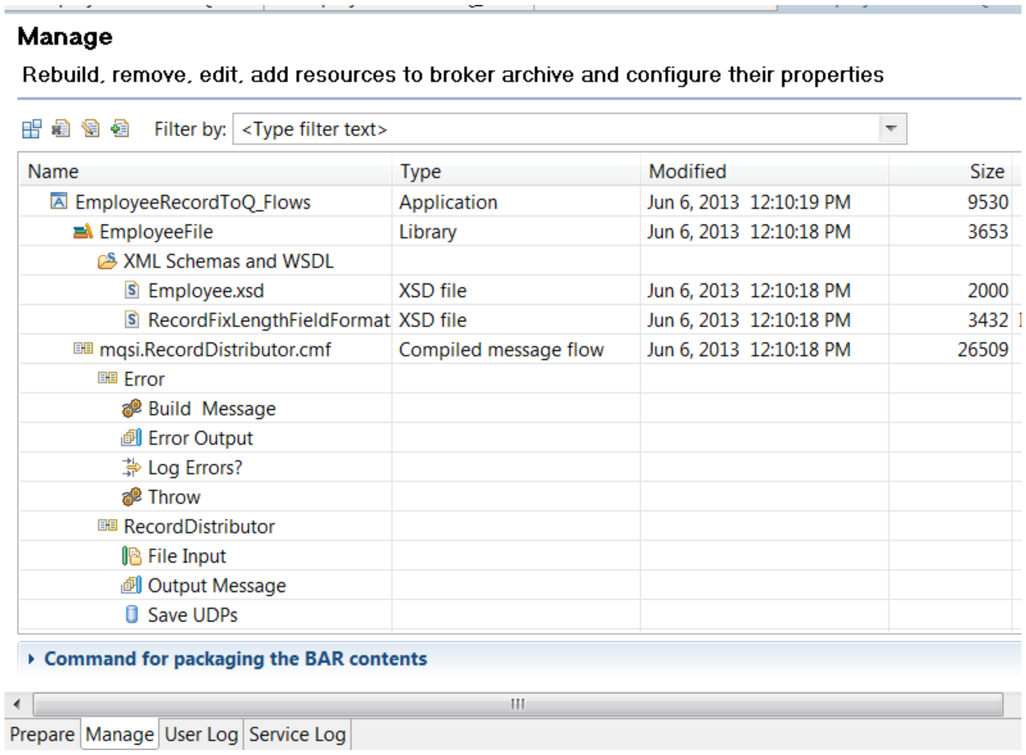
___ g. Click **Build and Save**. The BAR file is built.

___ h. When the **Operation completed successfully** message is displayed, click **OK**.

___ i. Click the **User Log** tab in the BAR file editor and review the BAR file build actions.

___ j. Click the **Service Log** tab in the BAR file editor to verify that the application **EmployeeRecordToQ_Flows** was successfully added to the BAR file.

___ k. Click the **Manage** tab in the BAR file editor to view the components that were added to the BAR file.


Manage
Rebuild, remove, edit, add resources to broker archive and configure their properties
Filter by: <Type filter text>

Name	Type	Modified	Size
EmployeeRecordToQ_Flows	Application	Jun 6, 2013 12:10:19 PM	9530
EmployeeFile	Library	Jun 6, 2013 12:10:18 PM	3653
XML Schemas and WSDL			
Employee.xsd	XSD file	Jun 6, 2013 12:10:18 PM	2000
RecordFixLengthFieldFormat	XSD file	Jun 6, 2013 12:10:18 PM	3432
mqsi.RecordDistributor.cmf	Compiled message flow	Jun 6, 2013 12:10:18 PM	26509
Error			
Build Message			
Error Output			
Log Errors?			
Throw			
RecordDistributor			
File Input			
Output Message			
Save UDPs			

___ 2. Deploy the BAR file to the integration server.

___ a. In the Application Development navigator, expand the **EmployeeRecordToQ_Flows > BARs** folder.

__ b. Right-click **EmployeeRecordToQ.bar**, and then select **Deploy** from the menu. The Deploy menu is displayed.

__ c. Select **default** as the integration server, and then click **Finish**. After several moments, the BAR file is deployed.

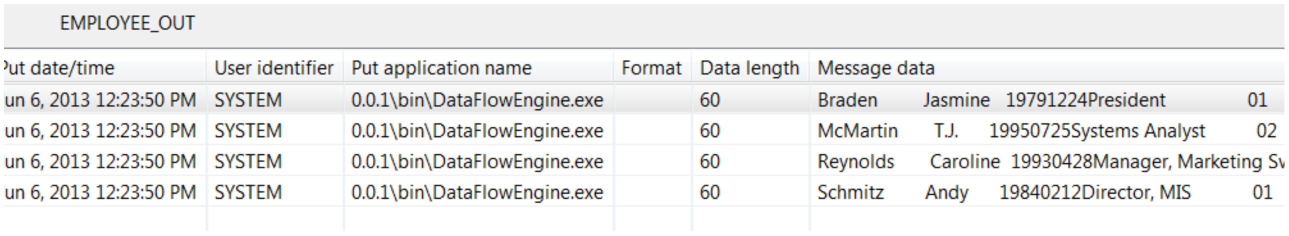
__ d. Select the **Deployment Log** tab and verify that the BAR file was deployed successfully.

__ e. Select the **Integration Nodes** tab and look at the components that were deployed to the integration server.

__ 3. To run the flow, use Windows Explorer to copy the file `EMPLOYEE.TXT` from `C:\Labs\lab02-Pattern\data` into the `C:\Labs\lab02-Pattern` directory.

The flow runs as soon as the file is written to the input directory that is specified in the FileInput node properties.

__ 4. Open IBM Integration Explorer or use the RFHUtil application to check the EMPLOYEE_OUT queue for messages. The EMPLOYEE_OUT queue should contain four messages, one for each record in the input file.


EMPLOYEE_OUT

ut date/time	User identifier	Put application name	Format	Data length	Message data
un 6, 2013 12:23:50 PM	SYSTEM	0.0.1\bin\DataFlowEngine.exe		60	Braden Jasmine 19791224President 01
un 6, 2013 12:23:50 PM	SYSTEM	0.0.1\bin\DataFlowEngine.exe		60	McMartin T.J. 19950725Systems Analyst 02
un 6, 2013 12:23:50 PM	SYSTEM	0.0.1\bin\DataFlowEngine.exe		60	Reynolds Caroline 19930428Manager, Marketing Sv
un 6, 2013 12:23:50 PM	SYSTEM	0.0.1\bin\DataFlowEngine.exe		60	Schmitz Andy 19840212Director, MIS 01

If the flow ran successfully, a copy of the file is also written to the `C:\Labs\lab02-Pattern\mqsiarchive` directory.

If the flow failed, a copy of the file is written to the `C:\Labs\lab02-Pattern\mqsibackout` directory.

Part 5: Clean up the environment

__ 1. Close all open editor windows by right-clicking the **X** on each tab in the editor. You can also right-click any of the tabs, and then click **Close All**.

__ 2. In the **Integration Nodes** view, delete the components from the integration server by right-clicking the integration server and then clicking **Delete > All flows and resources**.

End of exercise

Exercise 3. Analyzing runtime error scenarios

What this exercise is about

In this exercise, you analyze message flows with runtime exceptions. You predict what happens to the message based on the transaction mode, the wiring of the message processing nodes in the message flow, and the presence or absence of a back-out queue or dead-letter queue.

What you should be able to do

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

- Determine the location of a message when an exception occurs in a message flow

Introduction

The exercise shows eight message flow scenarios, each indicating where runtime exceptions occurred. Your task is to find out what happened with the message, and where you would find it.

Hint

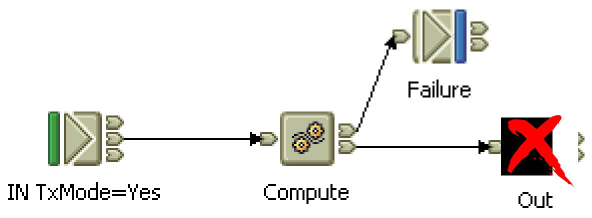
It is helpful to discuss your findings with another student.

Requirements

You need these instructions and a pen.

Exercise instructions

__ 1. In this flow, an MQInput node is wired to a Compute node. The Compute node Failure terminal is wired to an MQOutput node. The message fails on the node that is labeled **Out**.

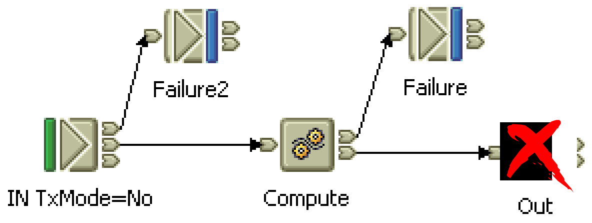


Questions

What happened with the message, and where you would find it, if:

- The DeadLetterQueue of queue manager = DLQ
- The BackoutQueue of IN queue = IN_BOQ

__ 2. In this flow, an MQInput node is wired to a Compute node and has an MQOutput node that is wired to its Failure terminal. The Compute node has a wired Out terminal and an MQOutput node that is wired to its Failure terminal. The message fails on the node that is labeled **Out**.

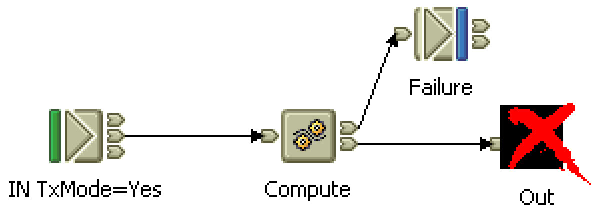


Questions

What happened with the message, and where would you find it, if:

- The DeadLetterQueue queue manager = DLQ
- The BackoutQueue IN queue= IN_BOQ

__ 3. In this flow, an MQInput node is wired to a Compute node. The Compute node has a wired Out terminal and an MQOutput node that is wired to its Failure terminal. The message fails on the node that is labeled **Out**.

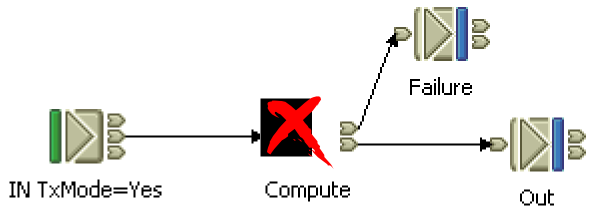


Questions

What happened with the message, and where would you find it, if:

- The DeadLetterQueue queue manager = DLQ
- IN queue has ***no*** BackoutQueue

__ 4. In this flow, an MQInput node is wired to a Compute node. The Compute node has MQOutput nodes that are wired to the Out terminal and Failure terminal. The message fails on the Compute node.



Questions

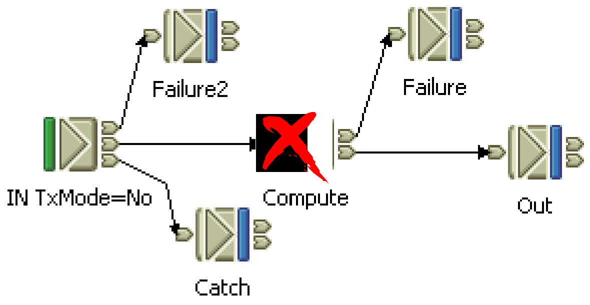
What happened with the message, and where would you find it, if:

- The DeadLetterQueue of QMgr = DLQ
- The BackoutQueue of IN queue = IN_BOQ

__ 5. In this flow, an MQInput node is wired to a Compute node and has an MQOutput node that is wired to its Failure terminal and another MQOutput node that is wired to its Catch terminal.

The Compute node has an MQOutput node that is wired to its Out terminal and an MQOutput node that is wired to its Failure terminal.

The message fails on the Compute node.



Questions

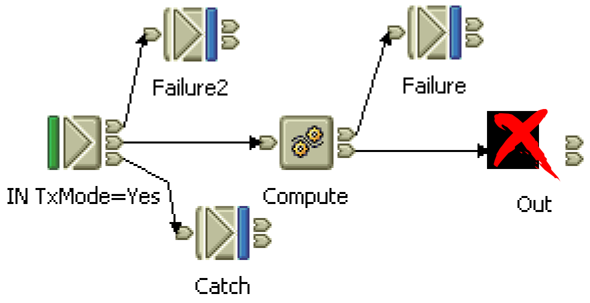
What happened with the message, and where would you find it, if

- The DeadLetterQueue of QMgr = DLQ
- The BackoutQueue of IN queue = IN_BOQ

__ 6. In this flow, an MQInput node is wired to a Compute node and has an MQOutput node that is wired to its Failure terminal and another MQOutput node that is wired to its Catch terminal.

The Compute node has an MQOutput node that is wired to its Out terminal and an MQOutput node that is wired to its Failure terminal.

The message fails on the node that is labeled **Out**.



Questions

What happened with the message, and where would you find it, if

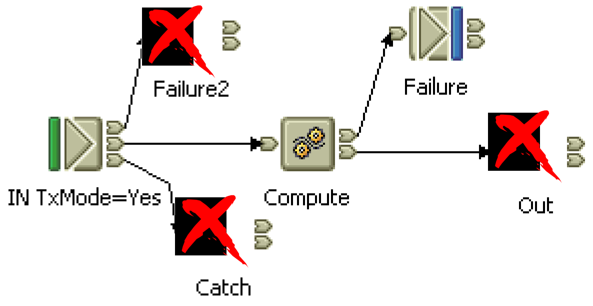
- The DeadLetterQueue of QMgr = DLQ

- The BackoutQueue of IN queue = IN_BOQ

___ 7. In this flow, an MQInput node is wired to a Compute node and has an MQOutput node that is wired to its Failure terminal and another MQOutput node that is wired to its Catch terminal.

The Compute node has an MQOutput node that is wired to its Out terminal and an MQOutput node that is wired to its Failure terminal.

There is a failure on the node that is connected to the Compute node Out terminal, and both the MQInput node Failure and Catch terminals.



Questions

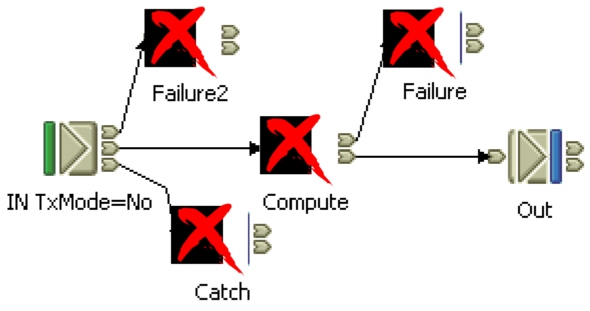
The original exception was thrown in the **Out** node, but more exceptions occurred during exception handling. What happened with the message, and where would you find it, if

- The DeadLetterQueue of QMgr = DLQ
- The BackoutQueue of IN queue = IN_BOQ

__ 8. In this flow, an MQInput node is wired to a Compute node and has an MQOutput node that is wired to its Failure terminal and another MQOutput node that is wired to its Catch terminal.

The Compute node has an MQOutput node that is wired to its Out terminal and an MQOutput node that is wired to its Failure terminal.

There is a failure on the node that is connected to Compute node Failure terminal, and both the MQInput node Failure and Catch terminals.



Questions

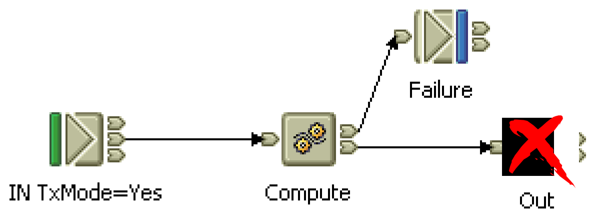
The original exception was thrown in Compute node, but more exceptions occurred during exception handling. What happened with the message, and where would you find it, if

- The DeadLetterQueue of QMgr = DLQ
- The BackoutQueue of IN queue = IN_BOQ

END OF EXERCISE

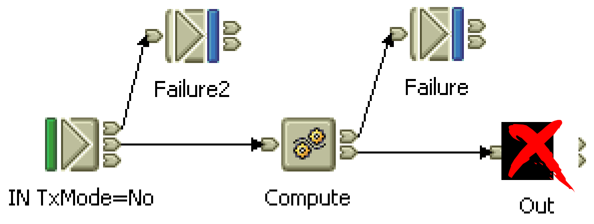
Exercise solutions

__ 1. Queue manager's DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ



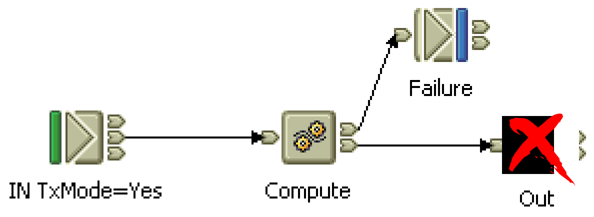
The message is rolled back to InputQ. After reaching the backout threshold, it is put to IN_BOQ.

__ 2. Queue manager's DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ



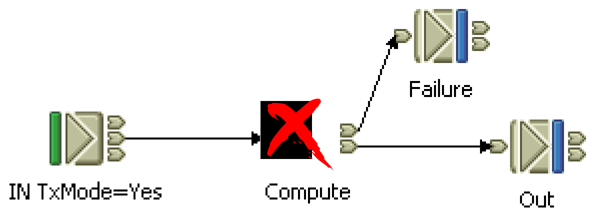
The message is discarded because the flow is not transactional, and therefore cannot be rolled back.

__ 3. Queue manager's DeadLetterQueue = DLQ; IN queue has NO BackoutQueue



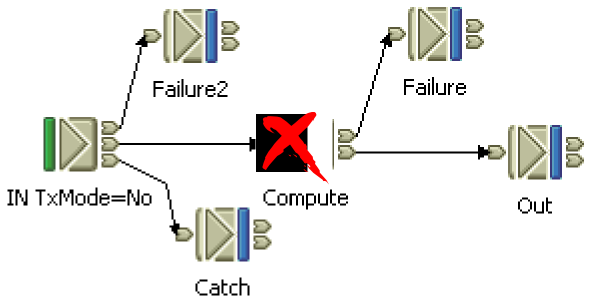
The message is rolled back to InputQ. After reaching backout threshold, it is put to the DLQ.

__ 4. Queue manager's DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ



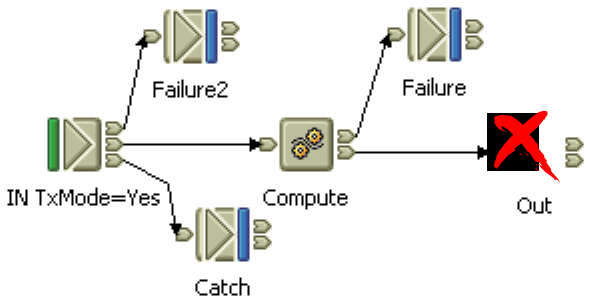
The message is propagated through the failure terminal of the Compute node, and thus written to the Failure queue.

__ 5. Queue manager’s DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ



The message is propagated through the failure terminal of the Compute node, and thus written to the Failure queue.

__ 6. Queue manager’s DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ

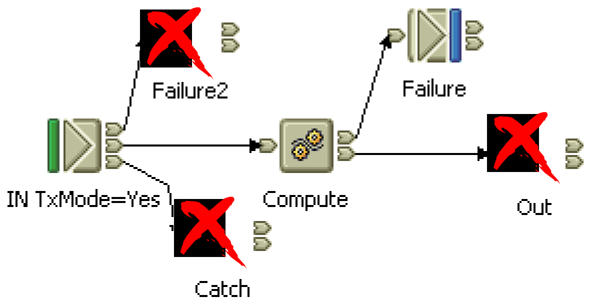


The message is written to the Catch queue.

__ 7. Queue manager’s DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ

Hint

The original exception was thrown in the Out node, but more exceptions occurred during explicit exception handling.

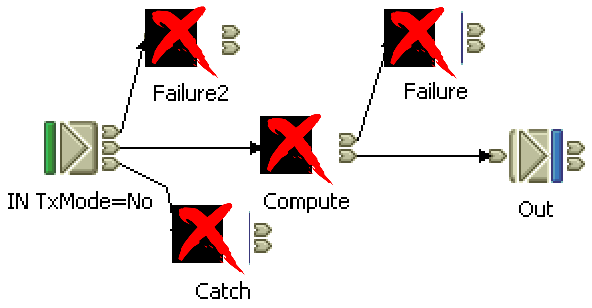


The message is put onto IN_BOQ after 2*BackoutThreshold retry attempts, which failed in FAILURE2. If IN_BOQ was full or no MQPUT was allowed, the integration node would try to put the message onto the DLQ. If that action also failed, the message would loop on the IN queue, as the integration node would constantly try to process it.

__ 8. Queue manager's DeadLetterQueue = DLQ; BackoutQueue of IN queue = IN_BOQ

Hint

The original exception was thrown in the Compute node, but more exceptions occurred during explicit exception handling.



The message is discarded after the exception in Catch. The Failure2 path is never taken because the flow is not transactional. There is no rollback.

End of exercise solutions

Exercise 4. Using problem determination tools

What this exercise is about

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

What you should be able to do

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

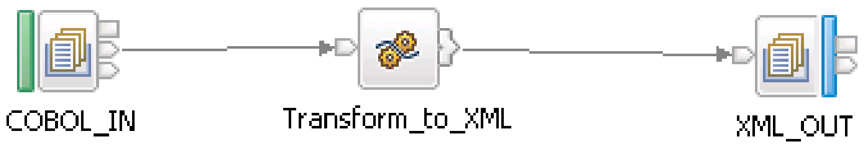
- Enable user traces and system traces, and retrieve the collected trace data
- Add a Trace node to a message flow
- Use the Debugger view to step through a message flow application
- Use RFHUtil to send test data, receive message output, and display message data
- Configure and use the Component Trace tool within the Test Client
- Examine the IBM Integration Bus logs and system logs to diagnose problems

Introduction

In this exercise, you apply various problem determination tools to a simple message flow. The message flow is that same message flow that you ran in Exercise 1.

The message flow has:

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



Requirements

- A workstation with the IBM Integration Bus and the default configuration created
- The files in `C:\Labs\Lab04-ProblemDetermination` and `C:\Lab\Tools`

Exercise instructions

Part 1: Prepare the environment and import resources

__ 1. Clean up from the previous exercise and ensure that the environment is ready.

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

__ b. Close all open editors.

__ c. Remove all flows and resources from the **default** integration server, if you did not already do so.

__ 2. Import the project interchange file that is named `ProblemDeterminationApp_PI.zip` file from the `C:\Labs\Lab04-ProblemDetermination\Resources` directory into your workspace.

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

__ b. Expand the **Other** folder.

__ c. Select **Project Interchange** and click **Next**.

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

__ e. Browse to the `C:\Labs\Lab04-ProblemDetermination\Resources` directory.

__ f. Select `ProblemDeterminationApp_PI.zip` and then click **Open**. The **Import Project Interchange Contents** menu is displayed.

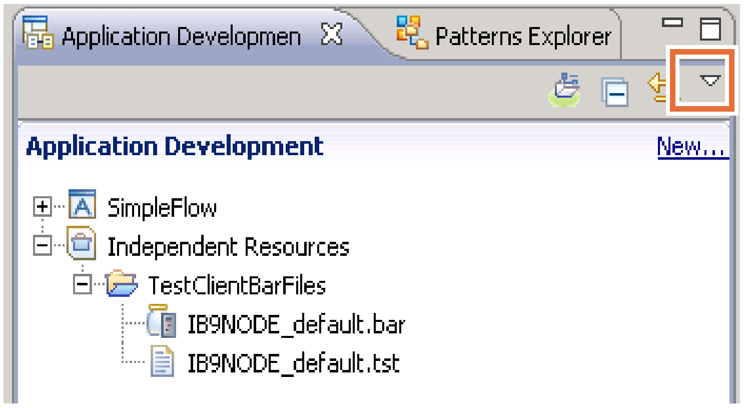
__ g. Ensure that **ProblemDetermination** is selected and then click **Finish**.

The ProblemDetermination application contains:

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

__ 3. Create a Working Set that is named **PDLAB** for the application that is used in this lab.

__ a. Click the **View Menu** icon (the downward pointing arrow) in the Application Development navigator toolbar.



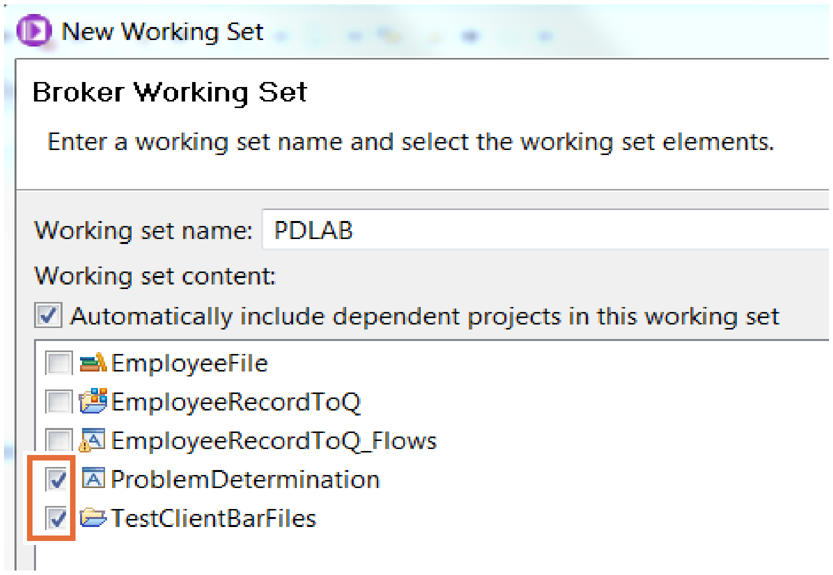
__ b. Click **Select Working Set** from the menu. The Select Working Set menu is displayed.

__ c. Click **New**. The New Working Set menu is displayed.

__ d. Select **Broker** as the working set type and then click **Next**. The Broker Working Set menu is displayed.

__ e. For **Working Set Name**, enter: PDLAB

__ f. For the working set content, select the **ProblemDetermination** application and the **TestClientBarFiles** project.


New Working Set

Broker Working Set

Enter a working set name and select the working set elements.

Working set name: PDLAB

Working set content:

[✓] Automatically include dependent projects in this working set

[ ] EmployeeFile
[ ] EmployeeRecordToQ
[ ] EmployeeRecordToQ_Flows
[✓] ProblemDetermination
[✓] TestClientBarFiles

__ g. Click **Finish**. The Select Working Set menu is displayed again.

__ h. Clear all items and then select **PDLAB** as the active working set.

__ i. Click **OK**. The components are added to the working set.

Part 2: Extend the message flow with a Trace node

In this part of the lab, you add a Trace node to the existing message flow and configure it.

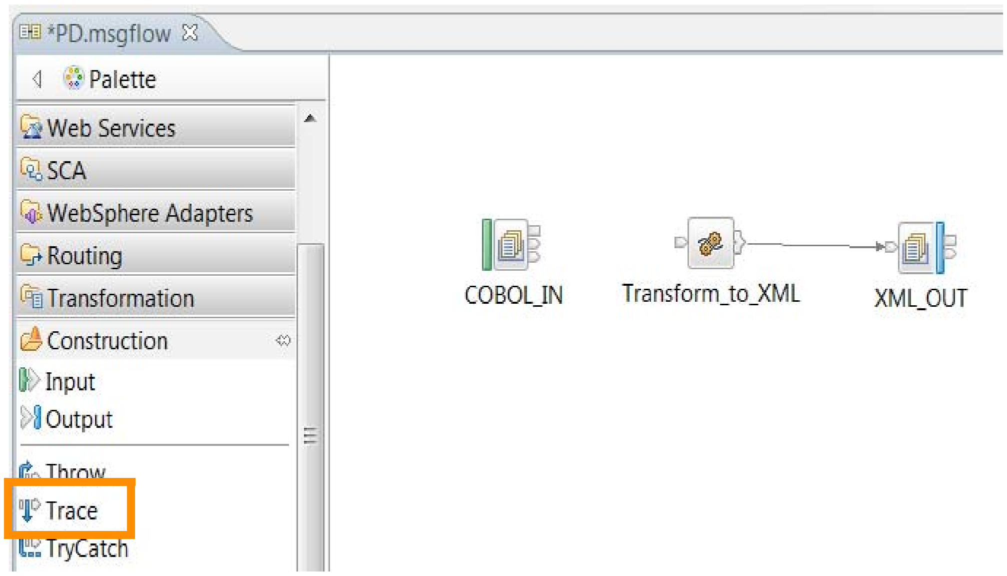
__ 1. Extend the message flow by wiring a **Trace** node between two existing nodes.

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

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

__ c. Delete the connection between the MQInput node (named **COBOL_IN**) and the Compute node (named **Transform_to_XML**) by selecting the wire and pressing **Delete**.

__ d. In the palette on the left side of the editor, click **Construction**. The message processing nodes within that folder are displayed.

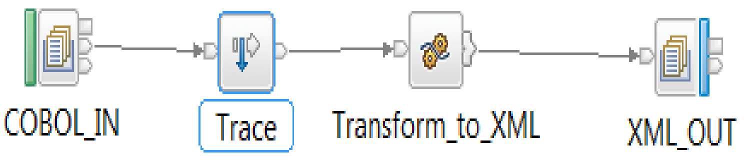


__ e. Click the **Trace** node, and then add it to the message flow between the **COBOL_IN** MQInput node and the **Transform_to_XML** Compute node.

You can click the node in the palette, and then click in the drawing canvas, or you can click and drag the node from the palette to the drawing canvas.

__ f. Connect the **Out** terminal of the MQInput node that is named **COBOL_IN** to the **In** terminal of the **Trace** node.

__ g. Connect the **Out** terminal of the **Trace** node to the **In** terminal of the **Compute** node that is named **Transform_to_XML**.



__ 2. Configure the Trace node properties to output a time stamp and the entire root tree to a file.

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

__ b. In **Properties** view, on the **Basic** tab, set **Destination** to **File**.

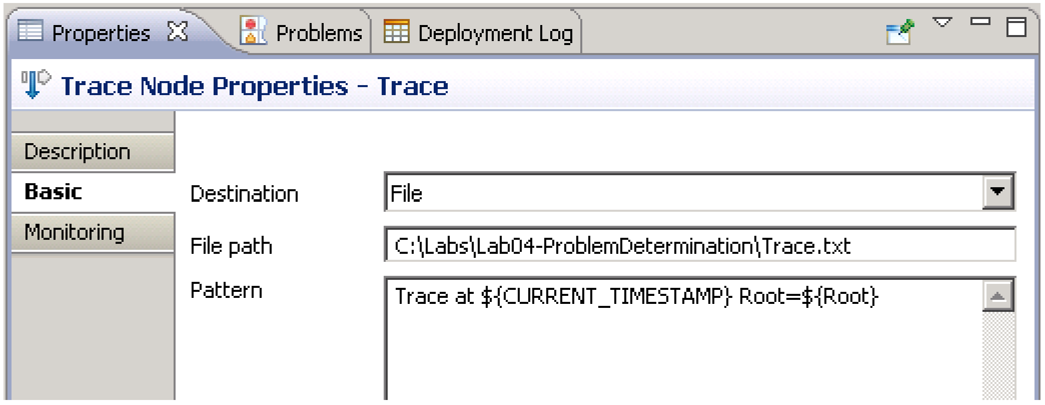
__ c. For the **File Path**, type: `C:\Labs\Lab04-ProblemDetermination\Trace.txt`

Important

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

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

The syntax is case-sensitive. A text file that is named `Cut&Paste.txt` in the `C:\Labs\Lab04-ProblemDetermination\Resources` directory contains the text for the **Pattern** field. To ensure that the syntax is correct, you can copy the text from this file and paste it into the **Pattern** field.



Trace Node Properties - Trace

Destination: File

File path: C:\Labs\Lab04-ProblemDetermination\Trace.txt

Pattern: Trace at ${CURRENT_TIMESTAMP} Root=${Root}

__ 3. Verify that the **Queue name** property of the MQInput node that is named COBOL_IN is set to **COBOL_IN**.

__ 4. Verify that **Queue name** property of the MQOutput node that is named XML_OUT is set to **XML_OUT**.

__ 5. Change the message flow layout (try different layouts).

__ a. Add bend-points to connections: click a connection and drag the line.

__ b. Rotate nodes: right-click a processing node and select **Rotate**.

__ c. Create right-angled connections: right-click in the Message Flow editor and select **Manhattan Layout.**

__ d. Use automated layout graphs: right-click in the Message Flow editor and select **Layout > Top to bottom**.

__ e. Move nodes: click and drag the mouse cursor to create a box around selected nodes. Release the mouse, and then move selected nodes on the pane.

__ f. Now that you selected multiple nodes, there are more layout choices, such as distribution and alignment. Options are available by right-clicking the selection, or by using the icon at the top of the Message Flow editor.

__ 6. Save the message flow by clicking **File > Save** from the menu bar, or by typing **Ctrl + S**.

__ 7. Verify that the ProblemDetermination application does not contain any errors.

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

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

Troubleshooting

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

In most cases, warning messages can be ignored. Also, messages from other projects in the workspace are displayed. If you see messages that are related to other projects, you can ignore them.

Part 3: Build the BAR file

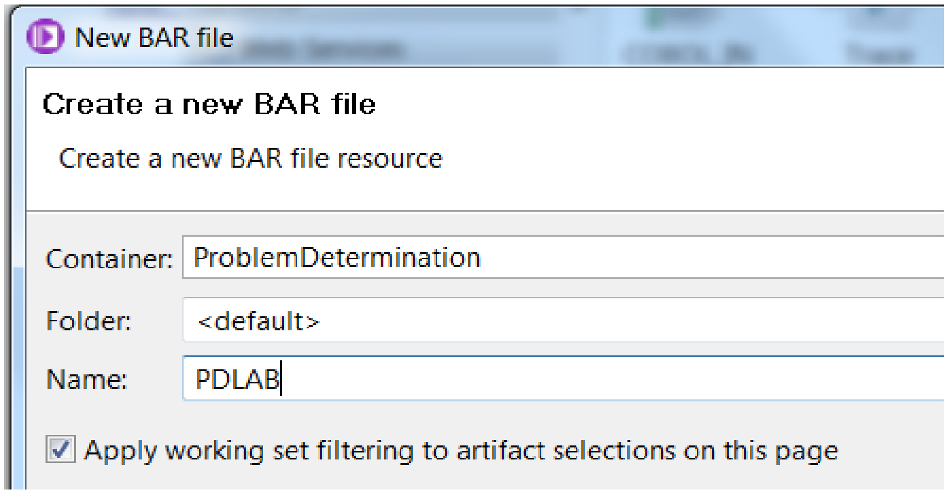
In this part of the lab, you create a BAR file that contains the message flow components.

__ 1. Create a BAR file that is named: `PDLAB.bar`

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

__ b. For the **Container**, select: `ProblemDetermination`

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


New BAR file

Create a new BAR file

Create a new BAR file resource

Container: ProblemDetermination

Folder: <default>

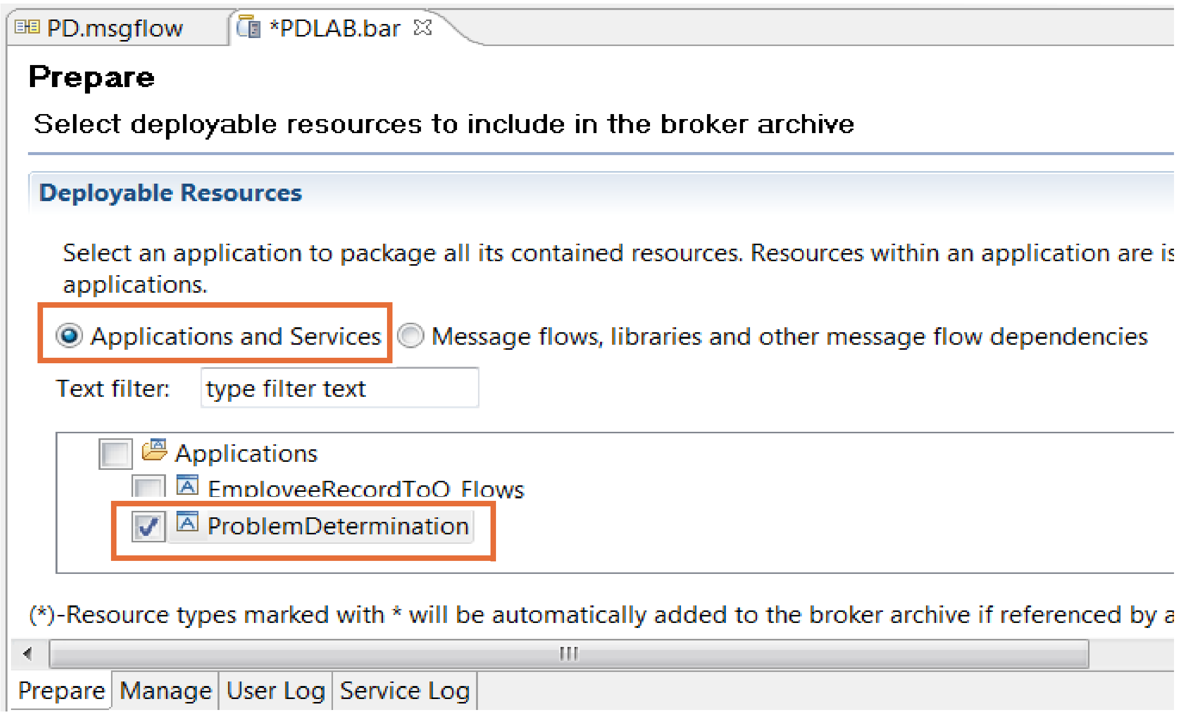
Name: PDLAB

✓ Apply working set filtering to artifact selections on this page

__ d. Click **Finish**.

The BAR file editor is opened on the **Prepare** tab.

__ 2. Select the components to include in the BAR file. In the **Deployable Resources** section, select the **ProblemDetermination** application.



Prepare

Select deployable resources to include in the broker archive

Deployable Resources

Select an application to package all its contained resources. Resources within an application are is
applications.

[● Applications and Services] ○ Message flows, libraries and other message flow dependencies

Text filter: type filter text

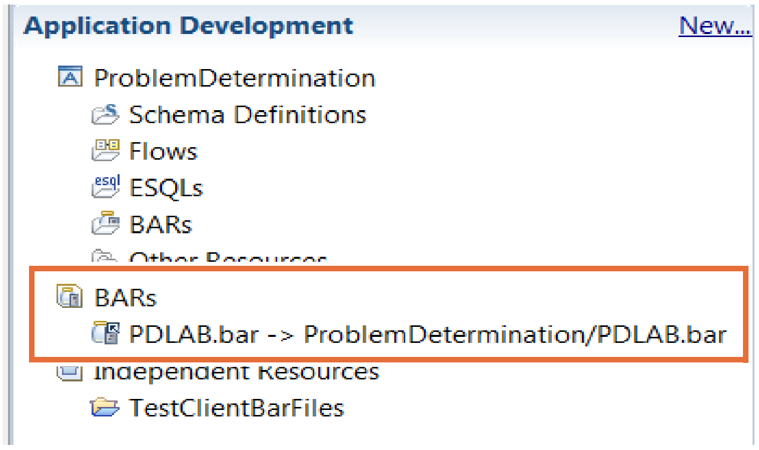
☐ Applications
☐ EmployeeRecordToO Flows
✓ ProblemDetermination

(*)-Resource types marked with * will be automatically added to the broker archive if referenced by a

__ 3. Click **Build and save**. The BAR file is built.

__ 4. Click **OK** to the **Operation completed successfully** message.

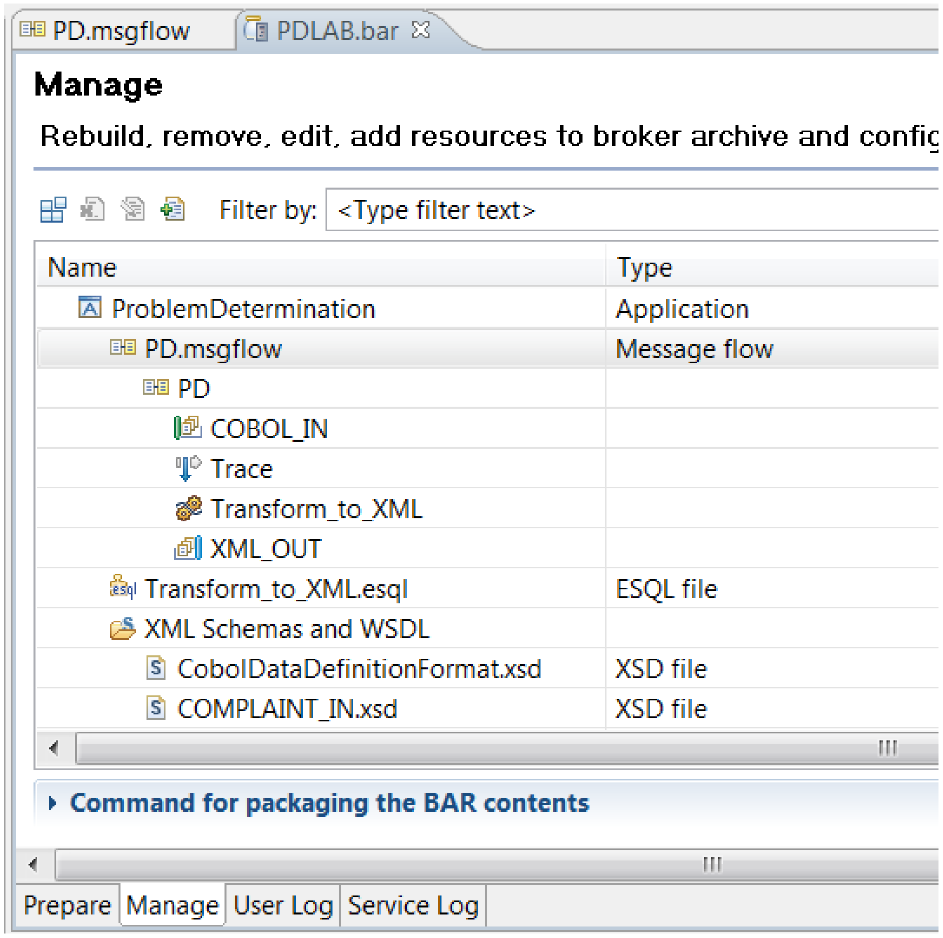
__ 5. In the Application Development navigator, verify that a new folder named `BARs` was created**,** and that it contains the `PDLAB.bar` file.



__ 6. At the bottom of the BAR file editor window, click the **Manage** tab.

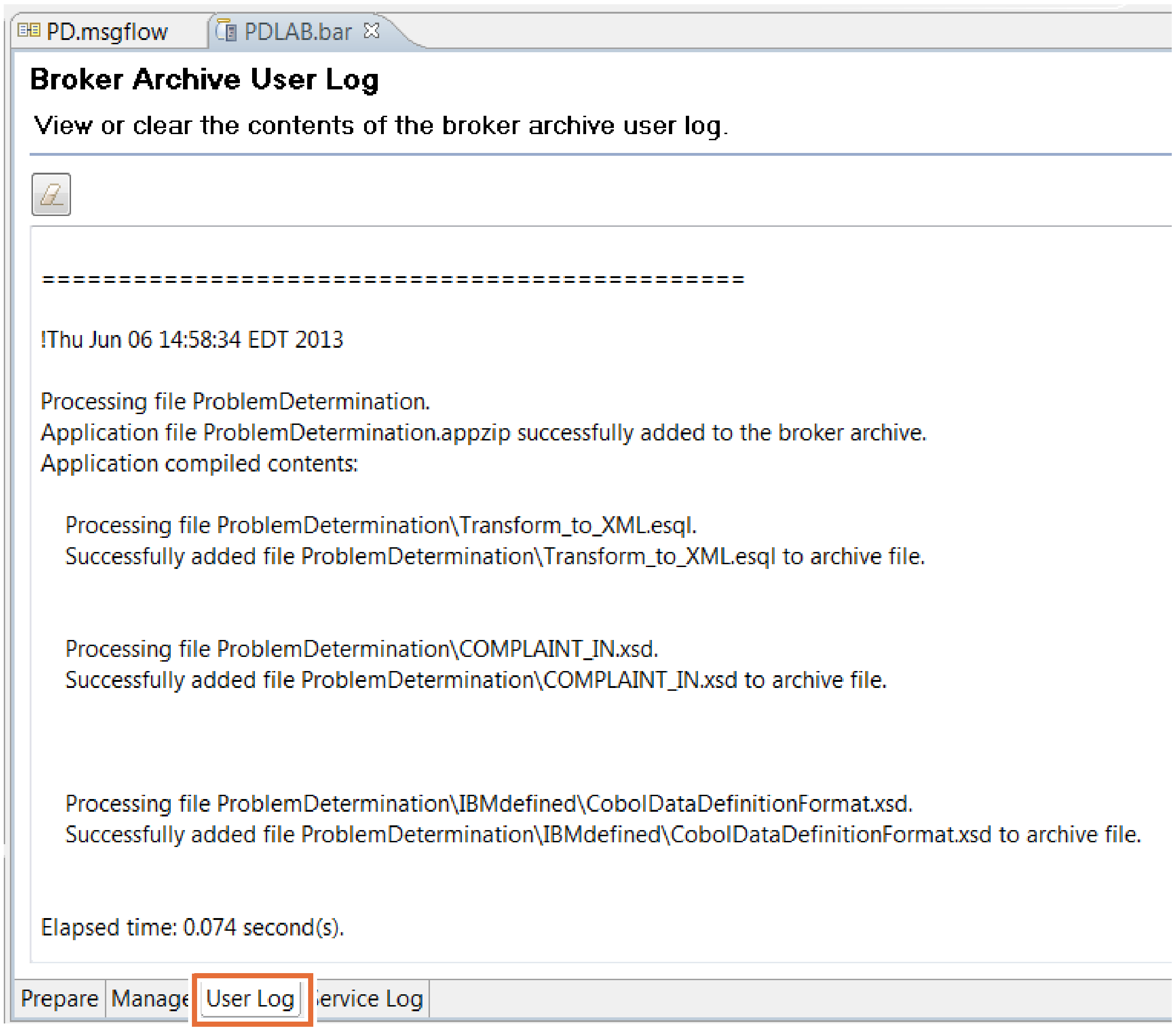
__ 7. Verify that the `PD.msgflow`, `Transform_to_XML.esql`, `CobolDataDefinitionFormat.xsd`, and `COMPLAINT_IN.xsd` files are included in the BAR file and that they show current time stamps.

You can expand the message flow (`PD.msgflow`) to see the message flow nodes that are included in the message flow.


PD.msgflow
PDLAB.bar
Manage
Rebuild, remove, edit, add resources to broker archive and config
Filter by:
<Type filter text>
Command for packaging the BAR contents
Prepare
Manage
User Log
Service Log

Name	Type
ProblemDetermination	Application
PD.msgflow	Message flow
PD	
COBOL_IN	
Trace	
Transform_to_XML	
XML_OUT	
Transform_to_XML.esql	ESQL file
XML Schemas and WSDL	
CobolDataDefinitionFormat.xsd	XSD file
COMPLAINT_IN.xsd	XSD file

__ 8. At the bottom of the BAR file editor, click the **User Log** tab, and review the messages that were generated during the creation of the BAR file. The messages indicate that the message flow was added to the BAR file. Again, in this instance, you can disregard any warning messages.



```
==============================================

!Thu Jun 06 14:58:34 EDT 2013

Processing file ProblemDetermination.
Application file ProblemDetermination.appzip successfully added to the broker archive.
Application compiled contents:

   Processing file ProblemDetermination\Transform_to_XML.esql.
   Successfully added file ProblemDetermination\Transform_to_XML.esql to archive file.


   Processing file ProblemDetermination\COMPLAINT_IN.xsd.
   Successfully added file ProblemDetermination\COMPLAINT_IN.xsd to archive file.



   Processing file ProblemDetermination\IBMdefined\CobolDataDefinitionFormat.xsd.
   Successfully added file ProblemDetermination\IBMdefined\CobolDataDefinitionFormat.xsd to archive file.


Elapsed time: 0.074 second(s).
```

Troubleshooting

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

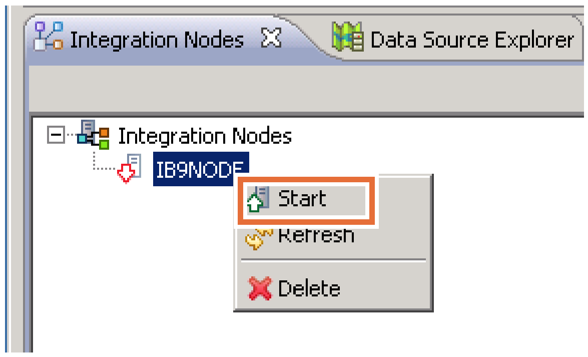
Any problems can be identified in the **Problems** view.

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

Part 4: Deploy

In this part of the exercise, you deploy the BAR file to an integration server in preparation for testing.

__ 1. Look at the **Integration nodes** view and ensure that the integration node `IB9NODE` is connected and running. If the integration node is not running, right-click the integration node and select **Start**.



__ 2. Deploy the BAR file.

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

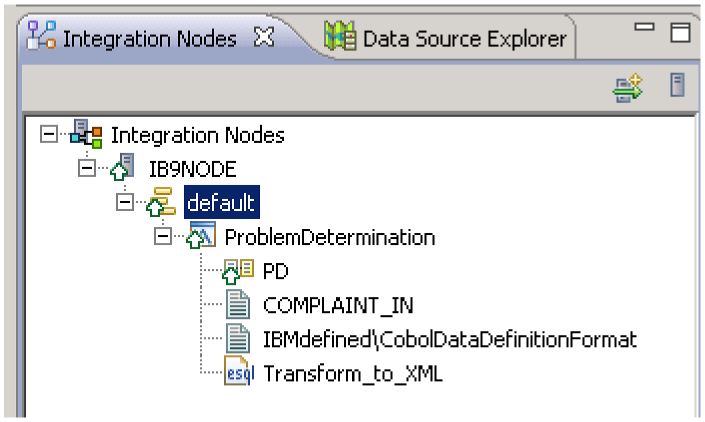
__ b. Right-click **PDLAB.bar**, and then select **Deploy** from the menu. The Deploy menu is displayed.

__ c. Select **default** as the integration server, and then click **Finish**. After several moments, the BAR file is deployed.

Note

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

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



__ 3. To ensure that the deployment operation completed successfully, check the deployment log in the **Deployment Log** view.

Troubleshooting

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

Check the local error log of the integration node. On Windows:

1. Select **Administrative Tools > Event Viewer**.
2. Expand **Windows Logs**.
3. Select **Application**.

On UNIX, you would use the **syslog** to help identify possible problems.

Part 5: Send a test message by using RFHUtil

In this part of the exercise, you:

- Use the RFHUtil program to send test data to the input queue and cause the message flow to run
- Review the output from the Trace node
- Disable the Trace node so that it no longer generates trace information

__ 1. Send test data from the `C:\Labs\Lab04-ProblemDetermination\data\Complaint_cwf.txt` file to queue COBOL_IN.

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

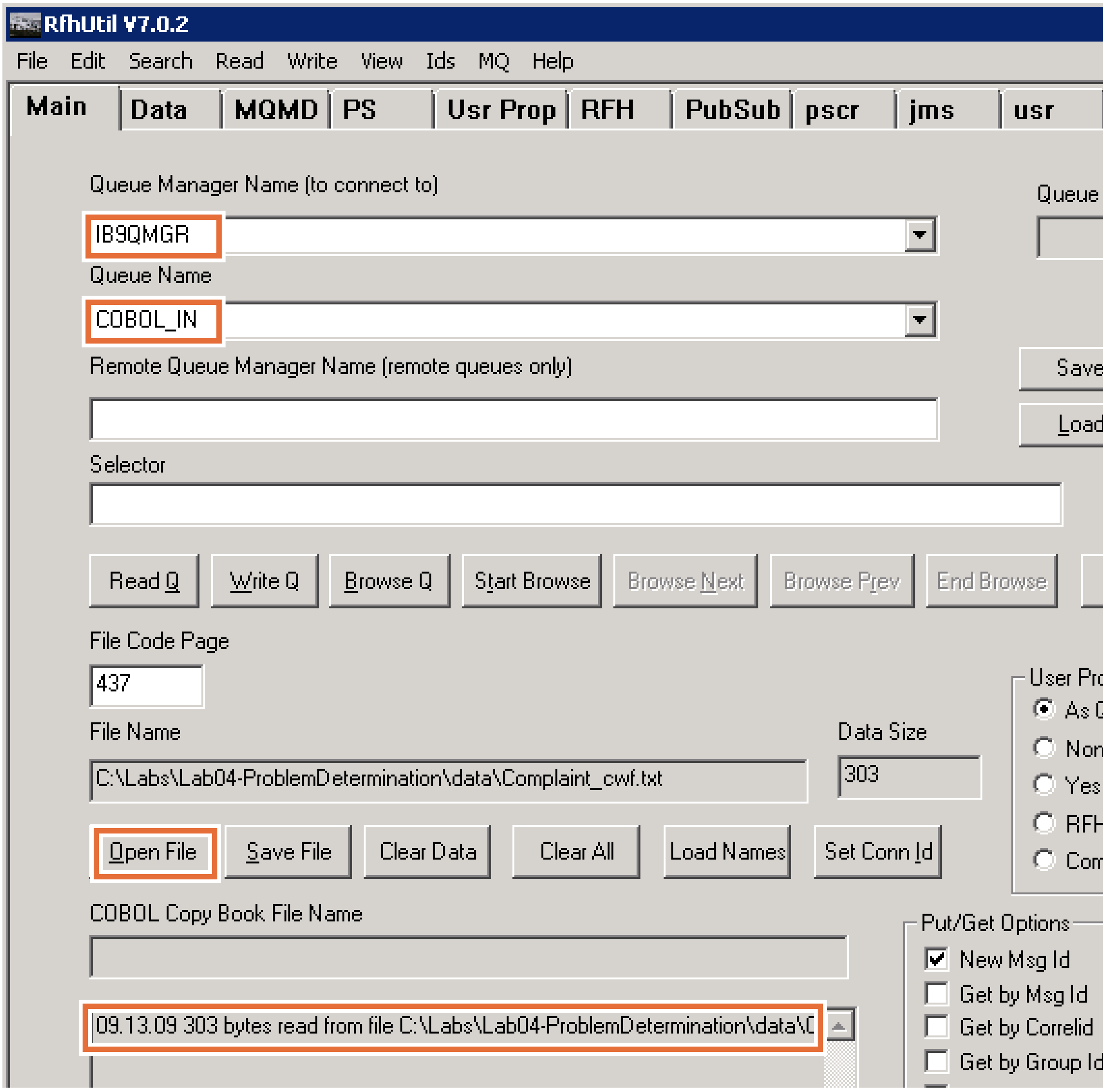
You can also start RFHUtil by using Windows Explorer to browse to the `C:\Labs\Tools` directory and then double-clicking `rfhutil`.exe.

__ b. In the **Main** tab of RFHUtil, for **Queue Manager Name**, select **IB9QMGR**.

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

__ d. Click **Open File**.

__ e. Select `C:\Labs\Lab04-ProblemDetermination\data\Complaint_cwf.txt` and select **Open**. The message window displays a time stamp with information about the data read from the file.


RfhUtil V7.0.2
File Edit Search Read Write View Ids MQ Help
Main Data MQMD PS Usr Prop RFH PubSub pscr jms usr
Read Q Write Q Browse Q Start Browse Browse Next Browse Prev End Browse
Open File Save File Clear Data Clear All Load Names Set Conn Id
Save
Load

Queue Manager Name (to connect to): IB9QMGR

Queue Name: COBOL_IN

Remote Queue Manager Name (remote queues only):

Selector:

File Code Page: 437

File Name: C:\Labs\Lab04-ProblemDetermination\data\Complaint_cwf.txt

Data Size: 303

COBOL Copy Book File Name:

09.13.09 303 bytes read from file C:\Labs\Lab04-ProblemDetermination\data\C

Put/Get Options: [✓ New Msg Id] Get by Msg Id Get by Correlid Get by Group Id

__ f. Click **Write Q**. The message is sent to the COBOL_IN queue.

__ 2. Review the test data that you just sent.

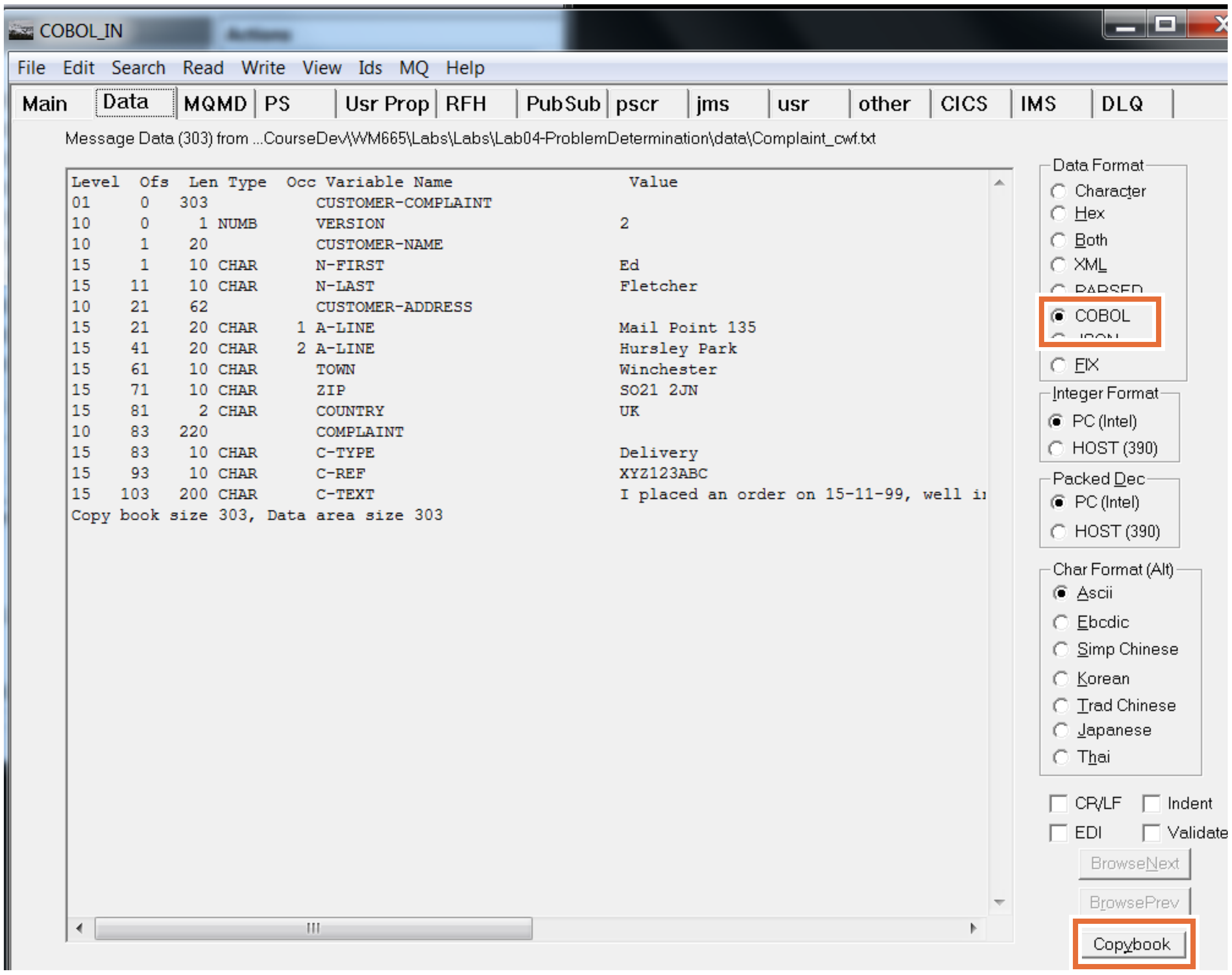
__ a. Click the **Data** tab. The data is presented in character format. You can select a different display format from the **Data Format** section on the right side of the window.

__ b. Because COBOL Copybook formats this message, you can view the message in its structured format by reading the Copybook data.

While still on the **Data** tab, click **Copybook** in the lower right corner, browse to the `C:\Labs\Lab04-ProblemDetermination\data\COMPLAINT_IN.cpy` directory, and then click **Open**.

__ c. Select **COBOL** in the **Data Format** section.

The COBOL Copybook loads, and the test message is displayed in COBOL format.



Message Data (303) from ...CourseDev\WM665\Labs\Labs\Lab04-ProblemDetermination\data\Complaint_cwf.txt

```
Level  Ofs  Len Type  Occ Variable Name                Value
01      0   303          CUSTOMER-COMPLAINT
10      0     1 NUMB     VERSION                        2
10      1    20          CUSTOMER-NAME
15      1    10 CHAR     N-FIRST                        Ed
15     11    10 CHAR     N-LAST                         Fletcher
10     21    62          CUSTOMER-ADDRESS
15     21    20 CHAR    1 A-LINE                        Mail Point 135
15     41    20 CHAR    2 A-LINE                        Hursley Park
15     61    10 CHAR     TOWN                           Winchester
15     71    10 CHAR     ZIP                            SO21 2JN
15     81     2 CHAR     COUNTRY                        UK
10     83   220          COMPLAINT
15     83    10 CHAR     C-TYPE                         Delivery
15     93    10 CHAR     C-REF                          XYZ123ABC
15    103   200 CHAR     C-TEXT                         I placed an order on 15-11-99, well i
Copy book size 303, Data area size 303
```

Data Format: [✓ COBOL]; Integer Format: [✓ PC (Intel)]; Packed Dec: [✓ PC (Intel)]; Char Format (Alt): [✓ Ascii]

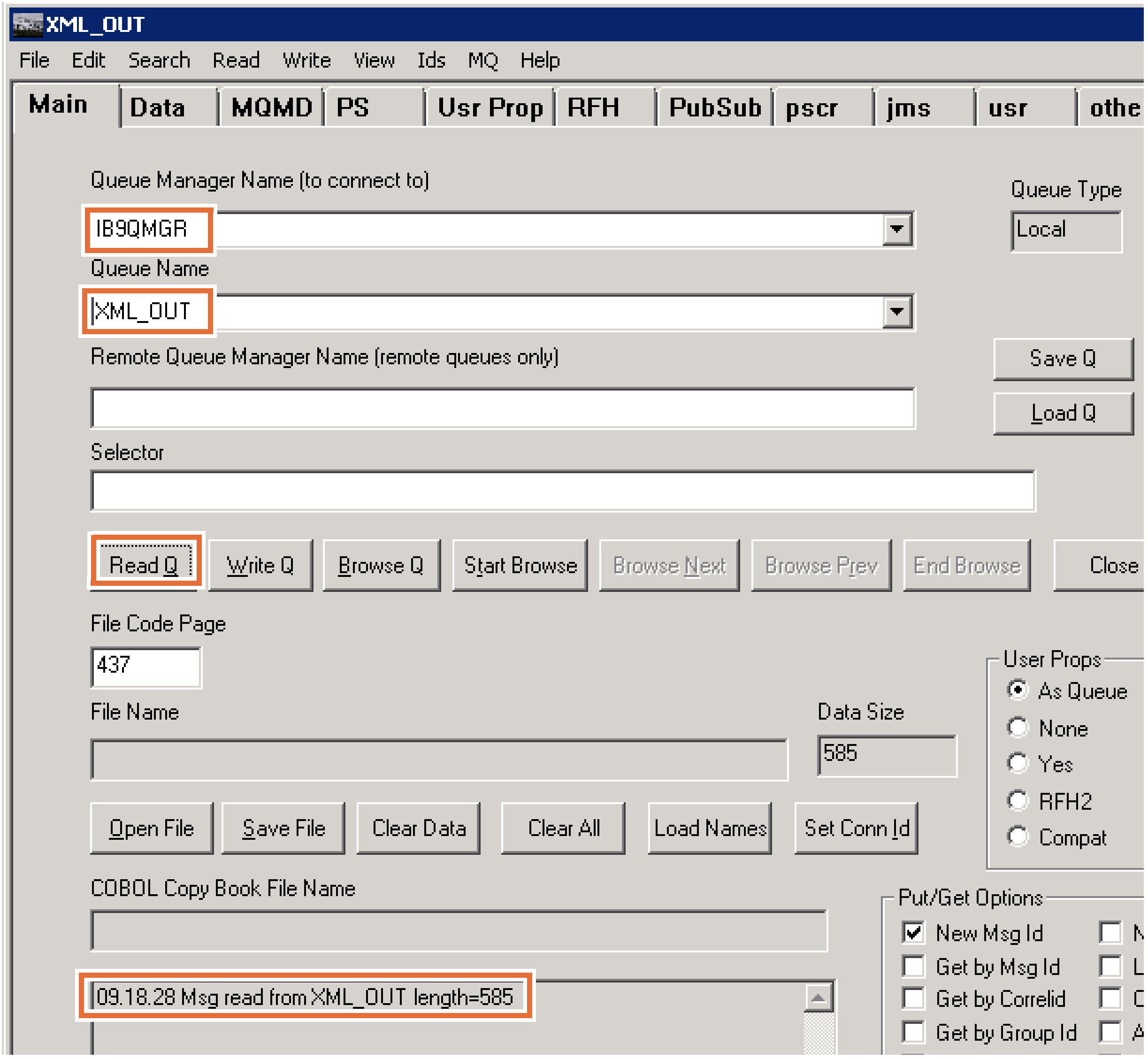
__ 3. Review the output message on queue **XML_OUT**.

__ a. Start a second instance of RFHUtil.

__ b. In the **Main** tab of RFHUtil, for **Queue Manager Name**, select **IB9QMGR**.

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

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



Queue Manager Name (to connect to): IB9QMGR

Queue Type: Local

Queue Name: XML_OUT

Remote Queue Manager Name (remote queues only):

Selector:

File Code Page: 437

File Name:

Data Size: 585

User Props: [✓ As Queue] None Yes RFH2 Compat

COBOL Copy Book File Name:

Put/Get Options: [✓ New Msg Id] Get by Msg Id Get by Correlid Get by Group Id

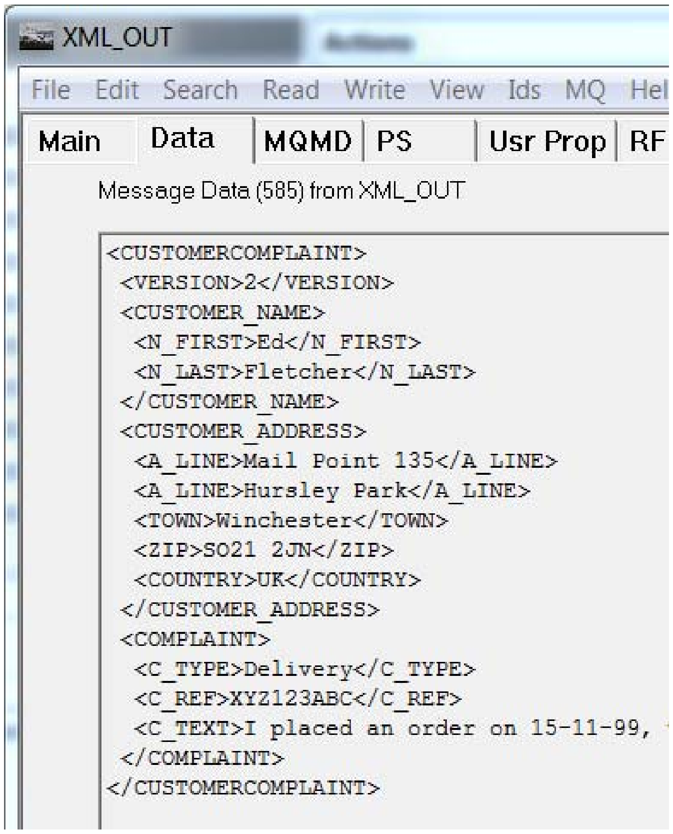
09.18.28 Msg read from XML_OUT length=585

Note

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

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

__ f. The message is formatted as XML; under **Data Format**, select **XML** to display the formatted data.


XML_OUT
File Edit Search Read Write View Ids MQ Hel
Main Data MQMD PS Usr Prop RF

Message Data (585) from XML_OUT

```
<CUSTOMERCOMPLAINT>
 <VERSION>2</VERSION>
 <CUSTOMER_NAME>
  <N_FIRST>Ed</N_FIRST>
  <N_LAST>Fletcher</N_LAST>
 </CUSTOMER_NAME>
 <CUSTOMER_ADDRESS>
  <A_LINE>Mail Point 135</A_LINE>
  <A_LINE>Hursley Park</A_LINE>
  <TOWN>Winchester</TOWN>
  <ZIP>SO21 2JN</ZIP>
  <COUNTRY>UK</COUNTRY>
 </CUSTOMER_ADDRESS>
 <COMPLAINT>
  <C_TYPE>Delivery</C_TYPE>
  <C_REF>XYZ123ABC</C_REF>
  <C_TEXT>I placed an order on 15-11-99,
 </COMPLAINT>
</CUSTOMERCOMPLAINT>
```

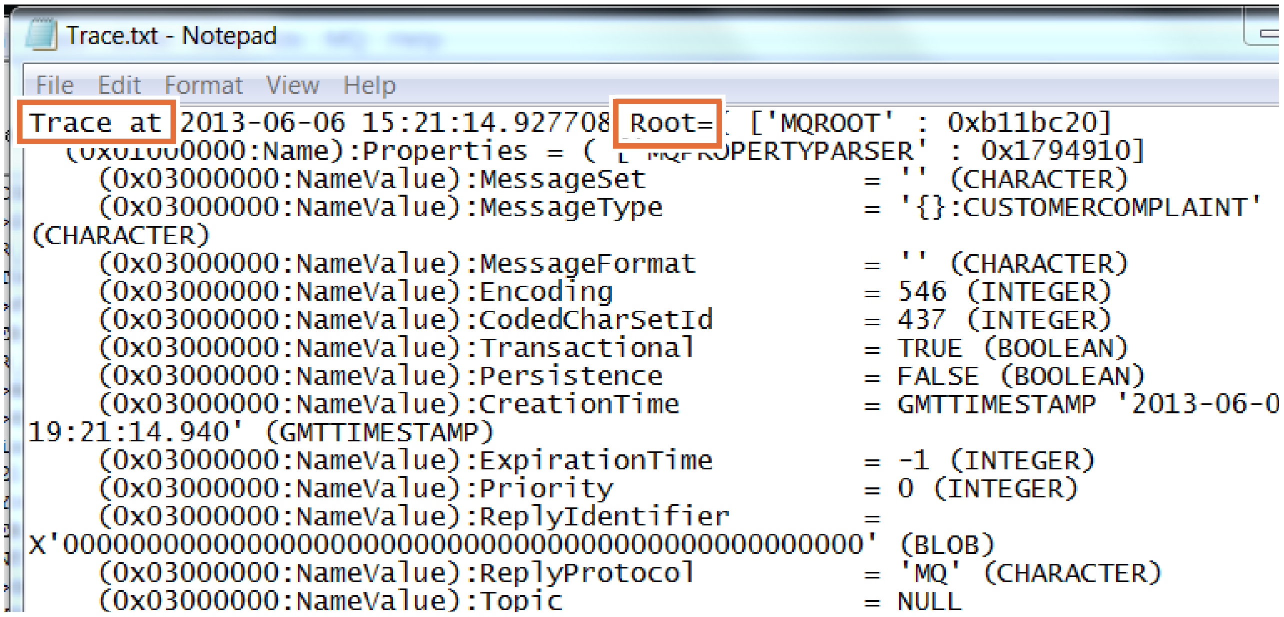
__ 4. Review the output from Trace node.

__ a. Using Windows Explorer, browse to file `C:\Labs\Lab04-ProblemDetermination\Trace.txt`. Double-click the file to open it with the default editor (Notepad).

Troubleshooting

If you cannot find your Trace output, the next steps (User Trace) help you determine why.

__ b. Review the data (a partial screen capture is shown here). You should see the text that you included in the Trace node configuration with the contents of the ESQL variables `CURRENT_TIMESTAMP` and `ROOT`.



```
Trace at 2013-06-06 15:21:14.927708 Root=( ['MQROOT' : 0xb11bc20]
  (0x01000000:Name):Properties = ( ['MQPROPERTYPARSER' : 0x1794910]
    (0x03000000:NameValue):MessageSet              = '' (CHARACTER)
    (0x03000000:NameValue):MessageType             = '{}:CUSTOMERCOMPLAINT'
(CHARACTER)
    (0x03000000:NameValue):MessageFormat           = '' (CHARACTER)
    (0x03000000:NameValue):Encoding                = 546 (INTEGER)
    (0x03000000:NameValue):CodedCharSetId          = 437 (INTEGER)
    (0x03000000:NameValue):Transactional           = TRUE (BOOLEAN)
    (0x03000000:NameValue):Persistence             = FALSE (BOOLEAN)
    (0x03000000:NameValue):CreationTime            = GMTTIMESTAMP '2013-06-0
19:21:14.940' (GMTTIMESTAMP)
    (0x03000000:NameValue):ExpirationTime          = -1 (INTEGER)
    (0x03000000:NameValue):Priority                = 0 (INTEGER)
    (0x03000000:NameValue):ReplyIdentifier         =
X'000000000000000000000000000000000000000000000000' (BLOB)
    (0x03000000:NameValue):ReplyProtocol           = 'MQ' (CHARACTER)
    (0x03000000:NameValue):Topic                   = NULL
```

__ 5. Disable the Trace node in the message flow.

__ a. On the Windows desktop, double-click the **IBM Integration Explorer** shortcut icon.

You can also start IBM Integration Explorer by clicking **Start > All Programs > IBM Integration Bus 9.0.0.0 > IBM Internet Explorer (Installation 1)** from the Windows desktop.

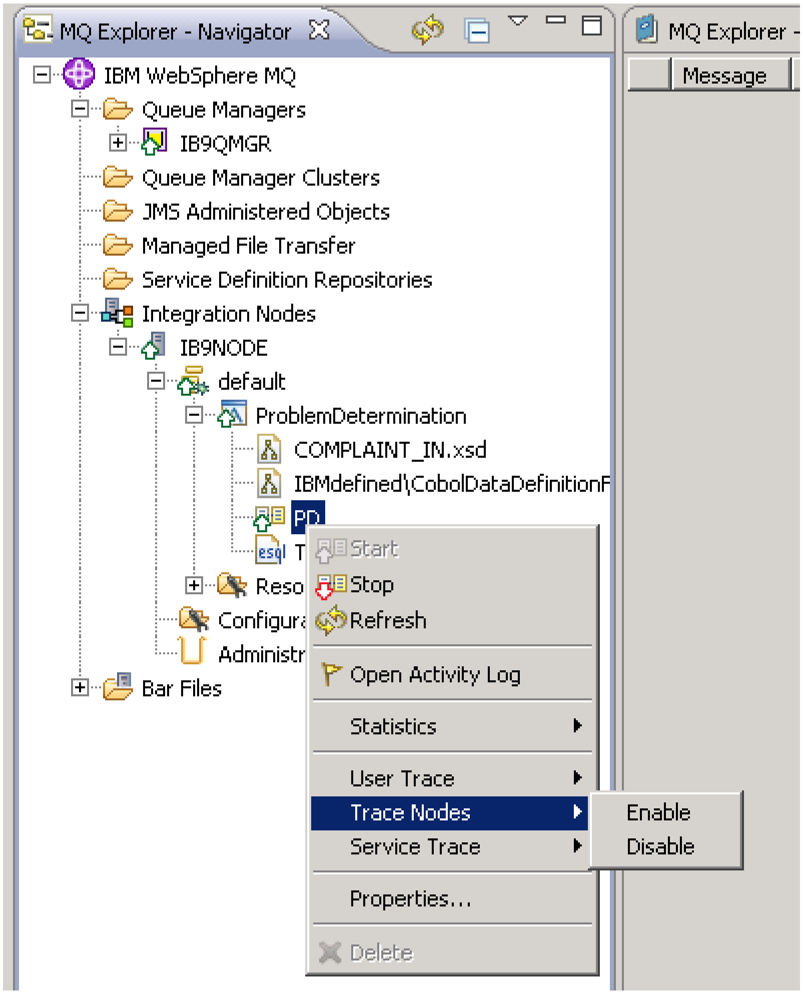
It can take a few minutes for IBM Integration Explorer (WebSphere MQ Explorer) to start.

__ b. In the navigator, expand **Integration Nodes** to list the current integration nodes.

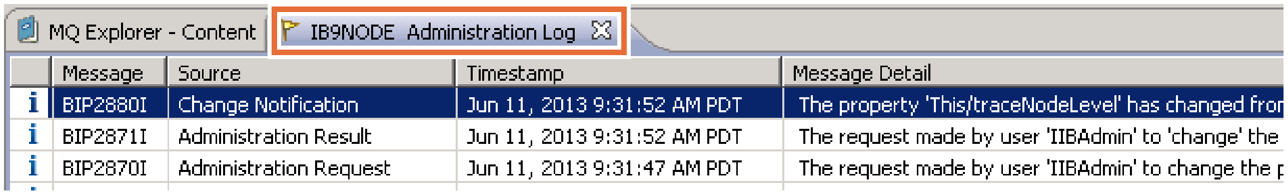
__ c. Expand the **default** integration server under **IB9NODE**.

__ d. Expand the **ProblemDetermation** application under the **default** integration server.

__ e. Right-click the message flow **PD** under **default** and then select **Trace Nodes > Disable**.



__ f. Check the **IB9NODE Administration Log** view in IBM Integration Explorer for a message that confirms that the Trace node is disabled. You must have the integration node that is selected in the navigator.



	Message	Source	Timestamp	Message Detail
i	BIP2880I	Change Notification	Jun 11, 2013 9:31:52 AM PDT	The property 'This/traceNodeLevel' has changed fror
i	BIP2871I	Administration Result	Jun 11, 2013 9:31:52 AM PDT	The request made by user 'IIBAdmin' to 'change' the
i	BIP2870I	Administration Request	Jun 11, 2013 9:31:47 AM PDT	The request made by user 'IIBAdmin' to change the p

__ g. Use RFHUtil to send another test message by returning to the RFHUtil session for **COBOL_IN** and clicking **Write Q** on the **Main** tab.

__ h. Review the contents of the Trace node file `C:\Labs\Lab04-ProblemDetermination\Trace.txt`. Scroll to the end of the file (new trace messages are appended to the end of the file). Check the time stamp to verify that no entry was made in the file from this run of the message flow.

Part 6: Configure the message flow to cause a runtime error

In this part of the exercise, you modify the message flow so that it causes a runtime error. This action allows you to do more problem determination steps in subsequent parts of the exercise.

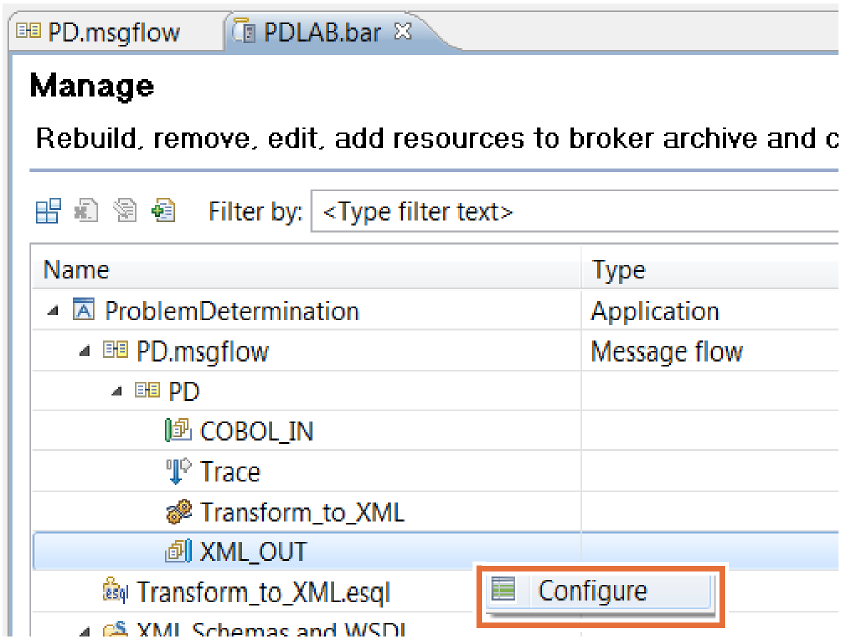
__ 1. Modify the BAR file to specify an output queue that does not exist, which results in a runtime error when the flow runs.

__ a. In the IBM Integration Toolkit, double-click the **PDLAB.bar** file in the Application Development navigator to open the BAR file editor.

__ b. Go to the **Manage** tab.

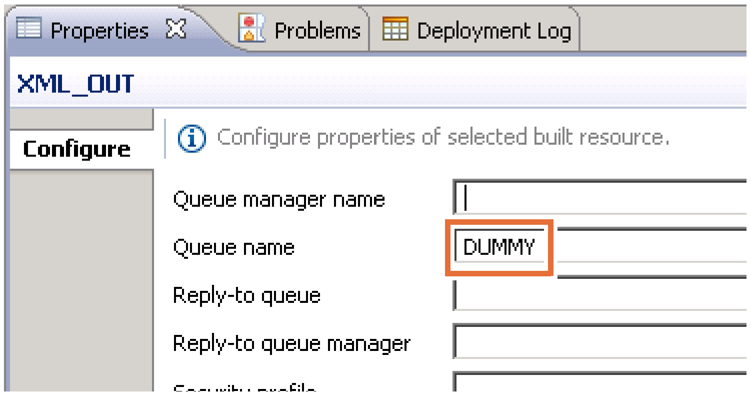
__ c. Expand **ProblemDetermination > PD.msgflow > PD**.

__ d. Right-click **XML_OUT** and select **Configure**.



Name	Type
ProblemDetermination	Application
PD.msgflow	Message flow
PD	
COBOL_IN	
Trace	
Transform_to_XML	
XML_OUT	
Transform_to_XML.esql	

__ e. In the **Properties** tab, enter `DUMMY` for the **Queue name**.



Queue manager name	
Queue name	DUMMY
Reply-to queue	
Reply-to queue manager	

__ f. Save the changes to the BAR file (**Ctrl+S**).

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

Troubleshooting

Remember to redeploy the modified BAR file. Changing the BAR file and then not redeploying it to the integration server is a common cause of unexpected results.

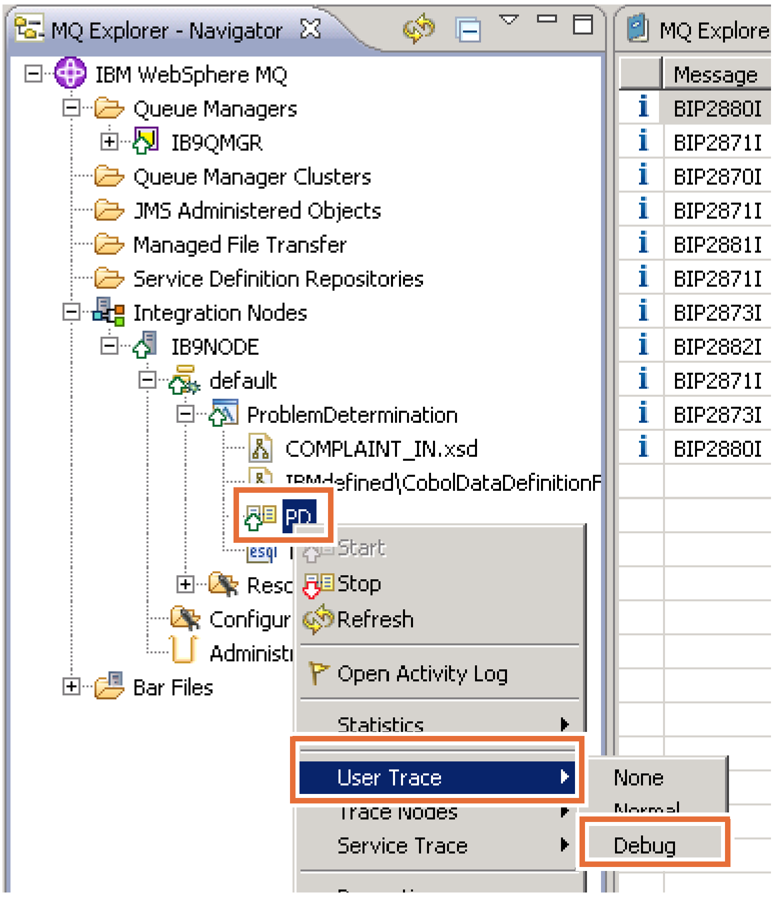
Part 7: Run the flow with User Trace

In this part of the exercise, you run the message flow with the invalid queue name defined for the MQOutput node that is named XML_OUT. This action causes a runtime error, which you diagnose by using a number of tools.

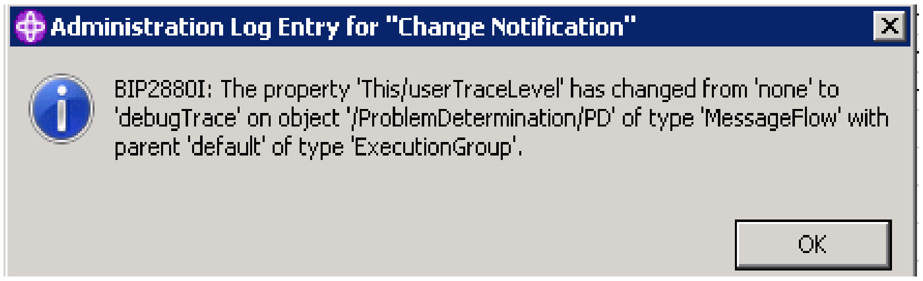
__ 1. In IBM Integration Explorer, enable User Trace for the message flow.

__ a. If it is not already open, open the IBM Integration Explorer.

__ b. Right-click the message flow that is named **PD** in integration server **default,** and then click **User Trace > Debug**.



__ c. Check the **IBB9NODE Administration Log** view in IBM Integration Explorer for a message that confirms that the User Trace mode is changed to debug. You see two *initiate* requests and then a *request complete* message. Double-click the message to view the details.



BIP2880I: The property 'This/userTraceLevel' has changed from 'none' to 'debugTrace' on object '/ProblemDetermination/PD' of type 'MessageFlow' with parent 'default' of type 'ExecutionGroup'.

__ 2. Using the RFHUtil session that is already running for queue COBOL_IN, send a test message by clicking **Write Q**.

Questions

Where do you expect to find the result message?

__ 3. One way to find the message is to check the number of messages on all queues.

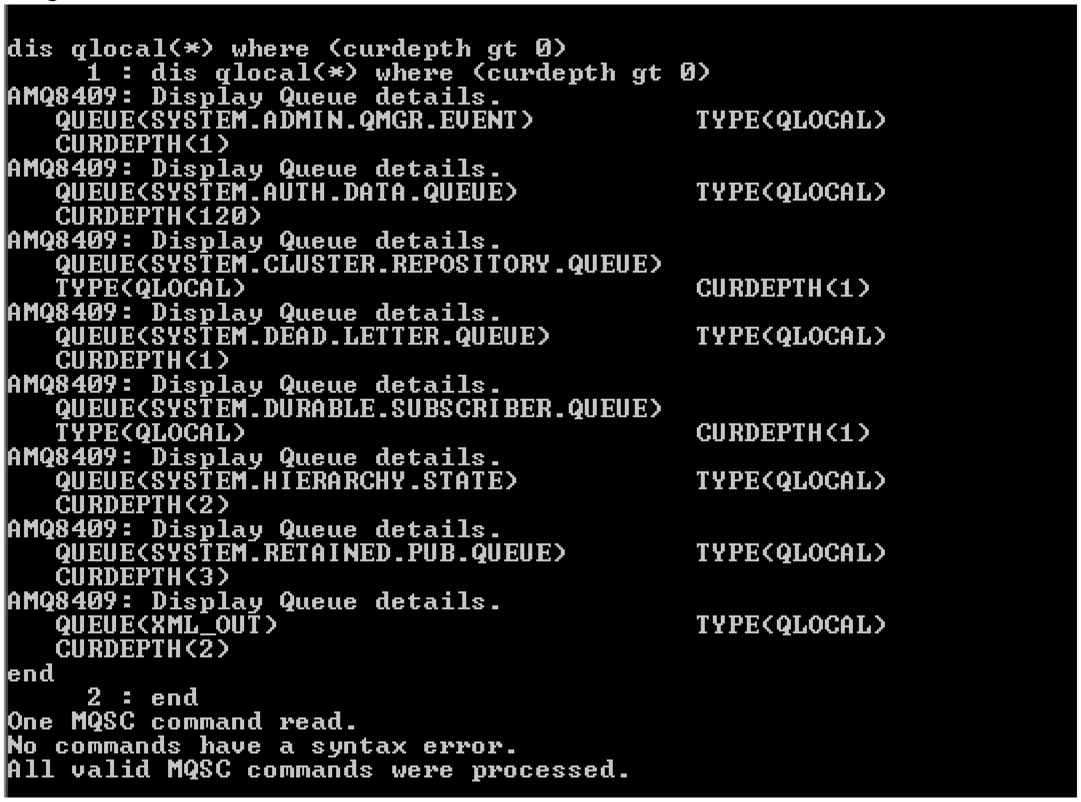
__ a. Open an IBM Integration Console window by clicking the IBM Integration Console shortcut icon on the Windows desktop or by clicking **Start > All Programs > IBM Integration Bus 9.0.0.0 > IBM Integration Console 9.0.0.0**.

__ b. Enter the following commands. The `DIS QLOCAL` command shows all queues that are defined to the default queue manager that contain one or more messages.

```
runmqsc IB9QMGR
dis qlocal(*) where(curdepth gt 0)
end
```

Review the list to see whether you can determine to which queue the message was routed.

Note: Your output might not be identical to the screen capture shown in this guide.

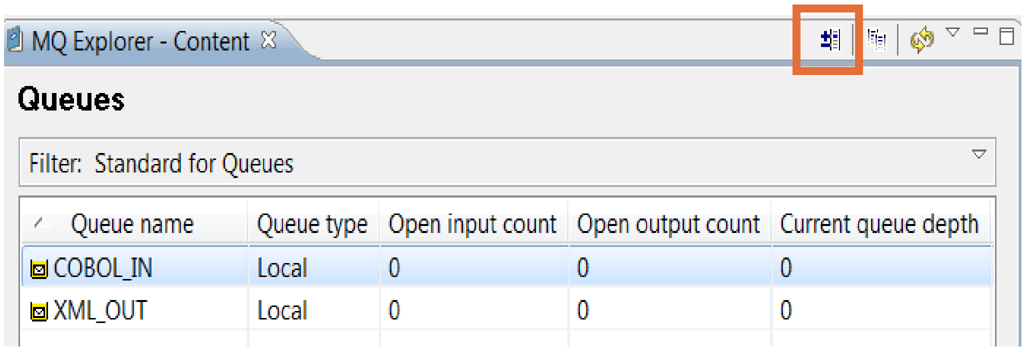


```
dis qlocal(*) where (curdepth gt 0)
    1 : dis qlocal(*) where (curdepth gt 0)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.ADMIN.QMGR.EVENT)          TYPE(QLOCAL)
   CURDEPTH(1)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.AUTH.DATA.QUEUE)           TYPE(QLOCAL)
   CURDEPTH(120)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.CLUSTER.REPOSITORY.QUEUE)
   TYPE(QLOCAL)                            CURDEPTH(1)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.DEAD.LETTER.QUEUE)         TYPE(QLOCAL)
   CURDEPTH(1)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.DURABLE.SUBSCRIBER.QUEUE)
   TYPE(QLOCAL)                            CURDEPTH(1)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.HIERARCHY.STATE)           TYPE(QLOCAL)
   CURDEPTH(2)
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.RETAINED.PUB.QUEUE)        TYPE(QLOCAL)
   CURDEPTH(3)
AMQ8409: Display Queue details.
   QUEUE(XML_OUT)                          TYPE(QLOCAL)
   CURDEPTH(2)
end
    2 : end
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

Questions

Did you find the message? Review the error flow diagram in the lecture materials. You can confirm the location of the message by using the IBM Integration Explorer and reviewing the message on the appropriate queue.

1) In IBM Integration Explorer, expand **IB9QMGR**.
2) Select **Queues** in the Navigator.
3) In the **Queues** content view, click the **Show System Objects** icon to show the user and system queues.



Queues

Filter: Standard for Queues

Queue name	Queue type	Open input count	Open output count	Current queue depth
COBOL_IN	Local	0	0	0
XML_OUT	Local	0	0	0

4) Locate the queue that contains the message in the list of all queues.
5) Right-click the queue name and then select **Browse Messages**.

6) Verify that the **Put application name** is `rfhutil.exe`.

7) When you are done examining the message, click **Close**.

8) Click the **Show System Objects** icon to show only the user queues.

__ 4. Retrieve the User Trace data by using the **Gettrace** script.

__ a. In the IBM Integration Console, browse to `C:\Labs\Tools`.

__ b. Enter: `gettrace`

Note

The script uses integration node **IB9NODE** and integration server **default** as defaults. If the names in your environment are different, you must specify the integration node and integration server in the `gettrace.cmd` by typing:

`C:\Labs\Tools\gettrace` *`IntegrationNodeName IntegrationServerName`*

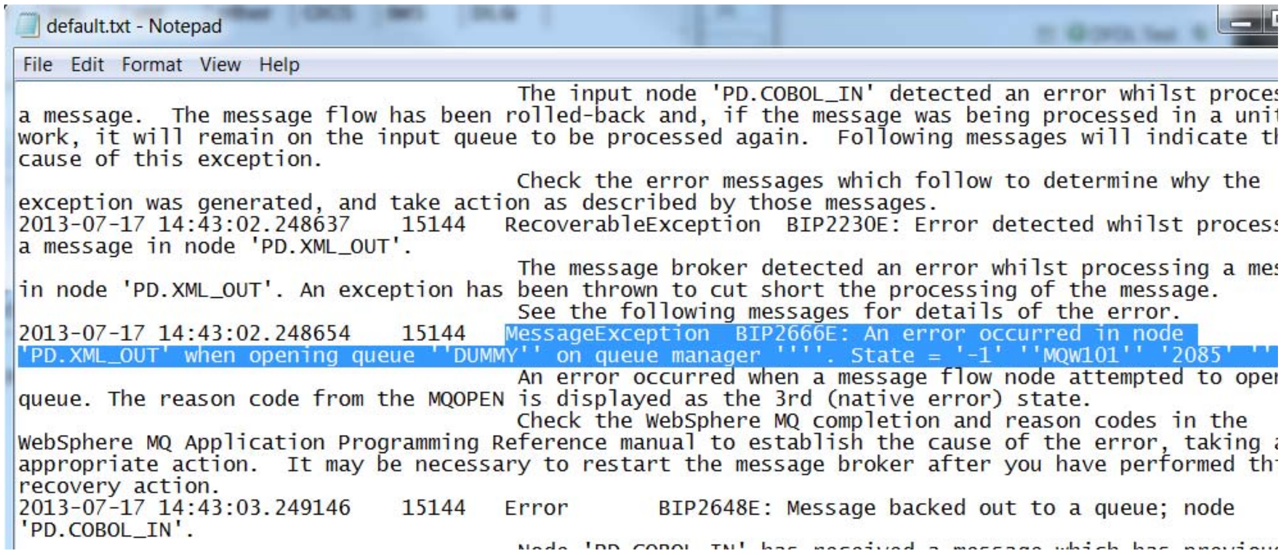
Substitute the appropriate values for *IntegrationNodeName* and *IntegrationServerName*.

The trace results are shown in a Notepad session.

__ c. Review the User Trace output. Runtime events and errors are shown there.

To find the error, search for the string: `MessageException`

This line gives you information and the 4-digit reason code WebSphere MQ reported.



```
                                          The input node 'PD.COBOL_IN' detected an error whilst proces
a message.  The message flow has been rolled-back and, if the message was being processed in a uni
work, it will remain on the input queue to be processed again.  Following messages will indicate t
cause of this exception.
                                          Check the error messages which follow to determine why the
exception was generated, and take action as described by those messages.
2013-07-17 14:43:02.248637    15144   RecoverableException  BIP2230E: Error detected whilst proces
a message in node 'PD.XML_OUT'.
                                          The message broker detected an error whilst processing a mes
in node 'PD.XML_OUT'. An exception has been thrown to cut short the processing of the message.
                                          See the following messages for details of the error.
2013-07-17 14:43:02.248654    15144   MessageException  BIP2666E: An error occurred in node
'PD.XML_OUT' when opening queue ''DUMMY'' on queue manager ''''. State = '-1' ''MQW101'' '2085' ''
                                          An error occurred when a message flow node attempted to ope
queue. The reason code from the MQOPEN is displayed as the 3rd (native error) state.
                                          Check the WebSphere MQ completion and reason codes in the
WebSphere MQ Application Programming Reference manual to establish the cause of the error, taking
appropriate action.  It may be necessary to restart the message broker after you have performed th
recovery action.
2013-07-17 14:43:03.249146    15144   Error        BIP2648E: Message backed out to a queue; node
'PD.COBOL_IN'.
```

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

Questions

Now you know both the failure reason and the current location of the message. What was the reason code reported by WebSphere MQ?

To find out what that reason code means, you can look it up in the information center, or type the following in a DOS command session:

`mqrc` *ReasonCode*

__ 5. Use the Windows Event Viewer to identify the error.

__ a. Right-click **Start** on the Windows desktop, and then select **Administrative Tools > Event Viewer**.

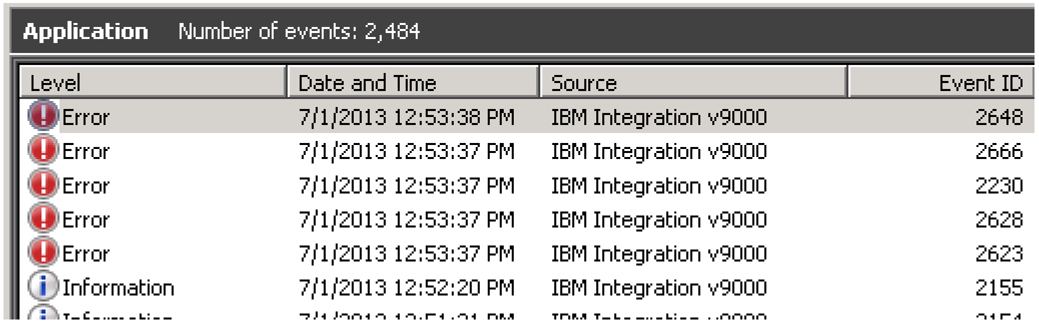
__ b. Expand **Windows Logs**.

Information

You can also access the Event Viewer by starting a DOS command session and entering the `eventvwr` command.

__ c. Click **Application** to open the event log for application programs.

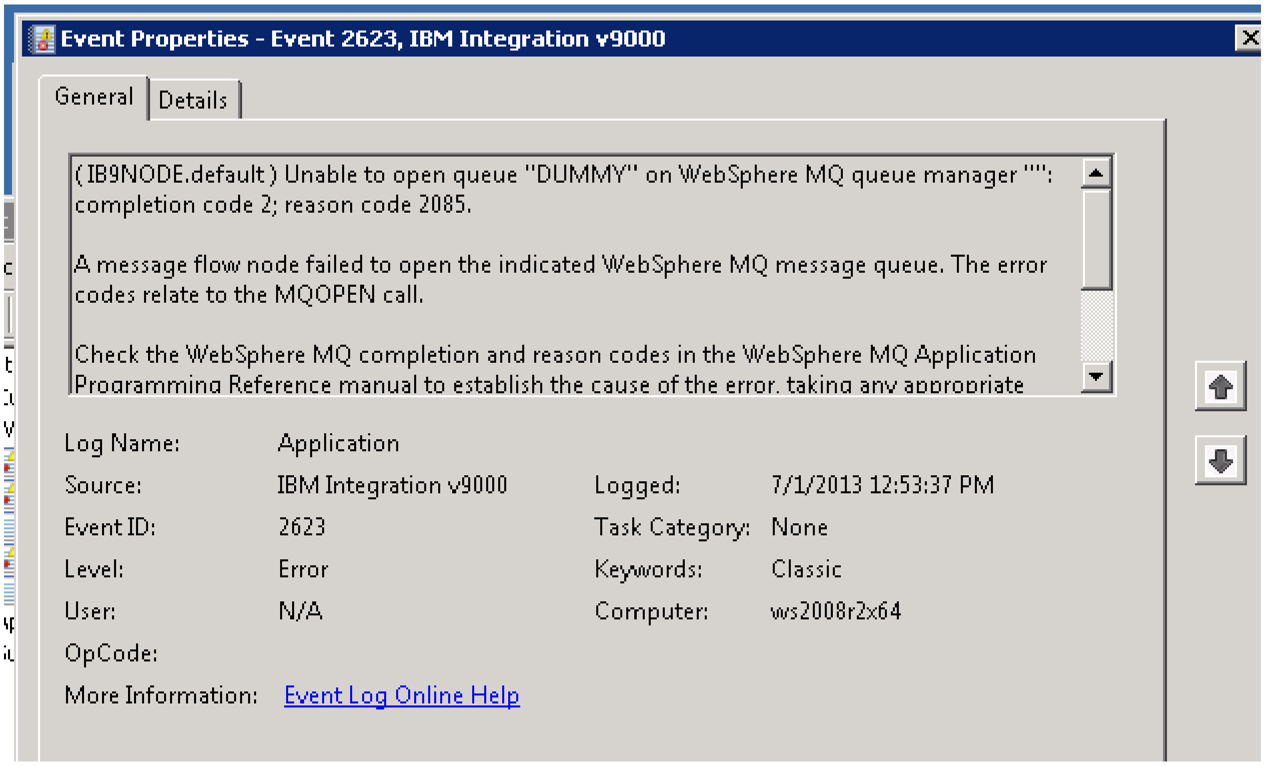
You see a number of entries that are marked with **Error**.



Application Number of events: 2,484

Level	Date and Time	Source	Event ID
Error	7/1/2013 12:53:38 PM	IBM Integration v9000	2648
Error	7/1/2013 12:53:37 PM	IBM Integration v9000	2666
Error	7/1/2013 12:53:37 PM	IBM Integration v9000	2230
Error	7/1/2013 12:53:37 PM	IBM Integration v9000	2628
Error	7/1/2013 12:53:37 PM	IBM Integration v9000	2623
Information	7/1/2013 12:52:20 PM	IBM Integration v9000	2155

__ d. Double-click the error message that has the *earliest* time stamp to see the Event Properties details.



```
( IB9NODE.default ) Unable to open queue "DUMMY" on WebSphere MQ queue manager "":
completion code 2; reason code 2085.

A message flow node failed to open the indicated WebSphere MQ message queue. The error
codes relate to the MQOPEN call.

Check the WebSphere MQ completion and reason codes in the WebSphere MQ Application
Programming Reference manual to establish the cause of the error, taking any appropriate
```

Log Name:	Application		
Source:	IBM Integration v9000	Logged:	7/1/2013 12:53:37 PM
Event ID:	2623	Task Category:	None
Level:	Error	Keywords:	Classic
User:	N/A	Computer:	ws2008r2x64
OpCode:			
More Information:	Event Log Online Help		

__ e. Review the **Description** to determine the cause of the error.

__ f. Subsequent error messages often provide more information about the error, or about processing that took place in response to the error. Review the other messages that are marked as **Error**.

__ g. Close the Event Viewer.

Part 8: Debugger

In this part of the exercise, you configure the debugger in the IBM Integration Toolkit and use it for problem determination.

__ 1. Set the debug port for the integration server in IBM Integration Toolkit.

__ a. Right-click the **default** integration server on the **Integration Nodes** tab in IBM Integration Toolkit and then select **Launch Debugger**.

__ b. You receive a message that indicates that the debug port is not set. Click **Configure**.

__ c. Enter 2311 as the **Flow Debug Port**, and then click **OK**.

The integration server must be restarted before the property change takes effect. The restart happens automatically when you click **OK**.

You might also notice the status icon in front of the **default** installation server change from started (green upward-pointing arrow) to stopped (red downward-pointing arrow) and back to started.

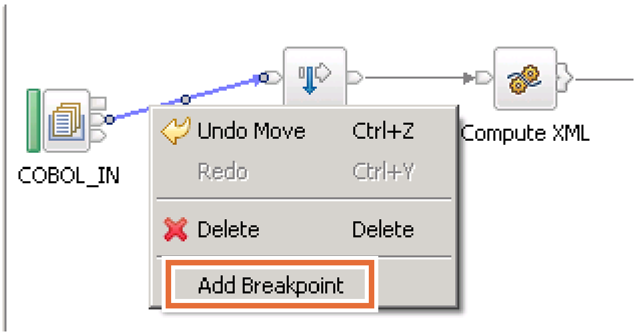
You can also review the change to the integration server in the IB9NODE Administration Log in the IBM Integration Explorer.

__ d. After the installation server restarts, you receive a confirmation message that the debugger will be started on port 2311. Click **OK**.

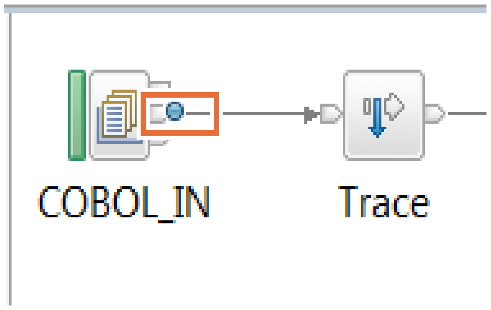
__ 2. Set breakpoints in the message flow and the ESQL program.

__ a. If it is not already open, open **PD.msgflow** in the IBM Integration Toolkit.

__ b. Right-click the connection between the MQInput node (COBOL_IN) and the Trace node, and then select **Add Breakpoint**.



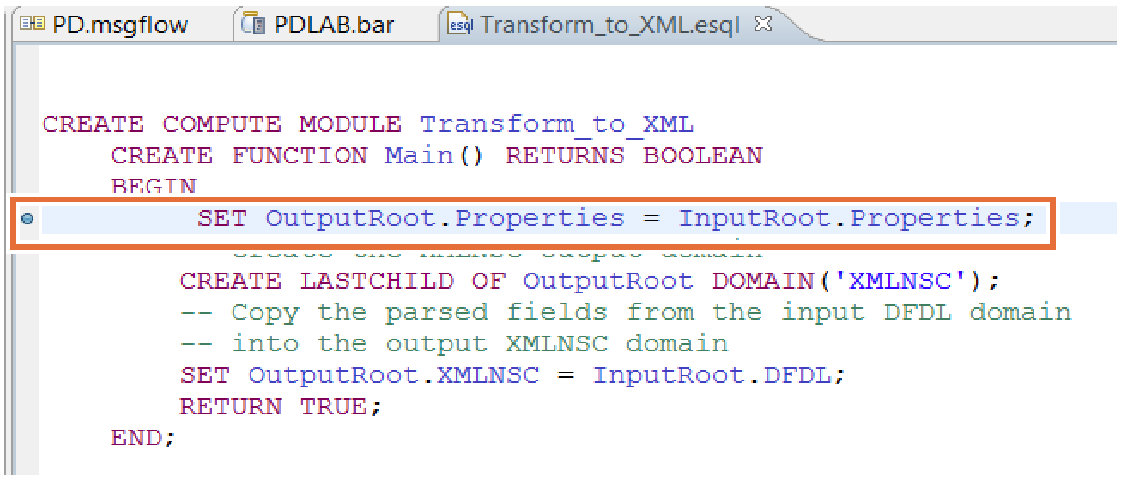
A blue circle is displayed on the wire, indicating that a breakpoint is set.



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

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

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



PD.msgflow | PDLAB.bar | Transform_to_XML.esql

```
CREATE COMPUTE MODULE Transform_to_XML
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
          SET OutputRoot.Properties = InputRoot.Properties;
        -- Create the XMLNSC output domain
        CREATE LASTCHILD OF OutputRoot DOMAIN('XMLNSC');
        -- Copy the parsed fields from the input DFDL domain
        -- into the output XMLNSC domain
        SET OutputRoot.XMLNSC = InputRoot.DFDL;
        RETURN TRUE;
    END;
```

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

___ 3. In the IBM Integration Toolkit, start and configure the debugger, and attach it to the integration server.

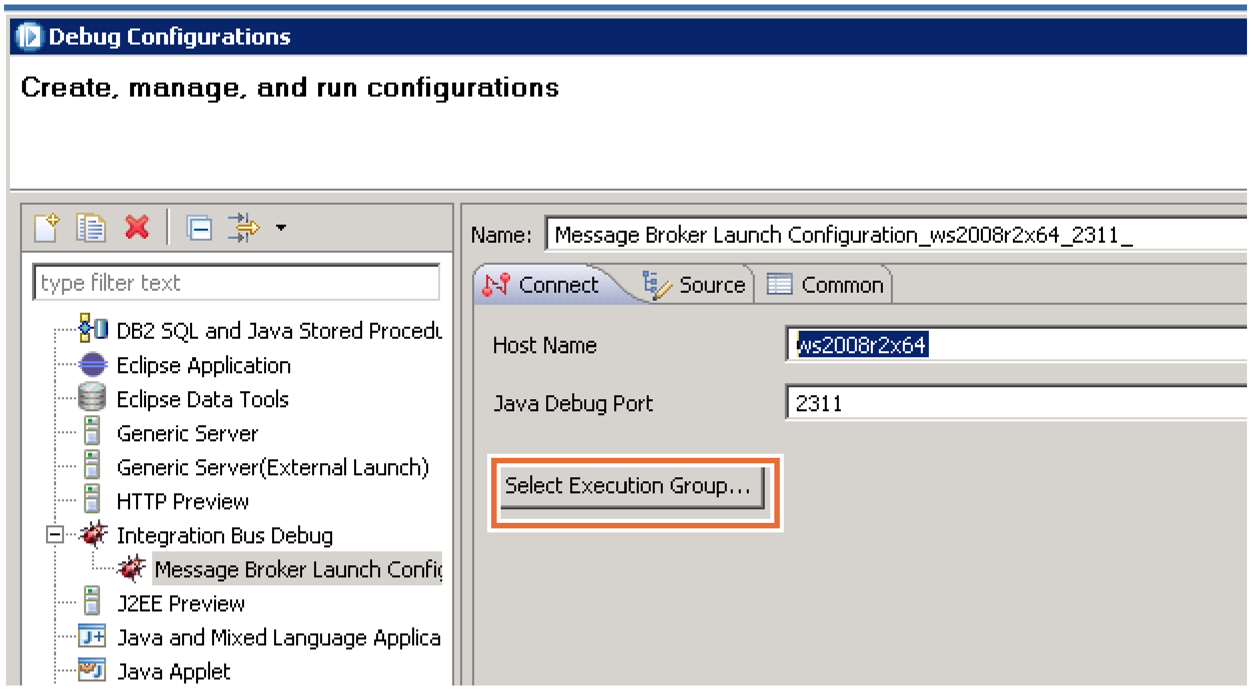
___ a. From the IBM Integration Toolkit toolbar, select **Window > Open Perspective > Debug** and then click **OK**. The debugging perspective opens.

___ b. From the toolbar, click **Run > Debug Configurations**. The Debug Configurations menu is displayed.

Because you are using the Debugger for the first time, you must create an IBM Integration Bus debug configuration.

___ c. Expand **Integration Bus Debug** and then select **Message Broker Launch Configuration**.

__ d. Click **Select Execution Group**.


Debug Configurations

Create, manage, and run configurations

type filter text

- DB2 SQL and Java Stored Procedu
- Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(External Launch)
- HTTP Preview
- Integration Bus Debug
  - Message Broker Launch Confi
- J2EE Preview
- Java and Mixed Language Applica
- Java Applet

Name: Message Broker Launch Configuration_ws2008r2x64_2311_

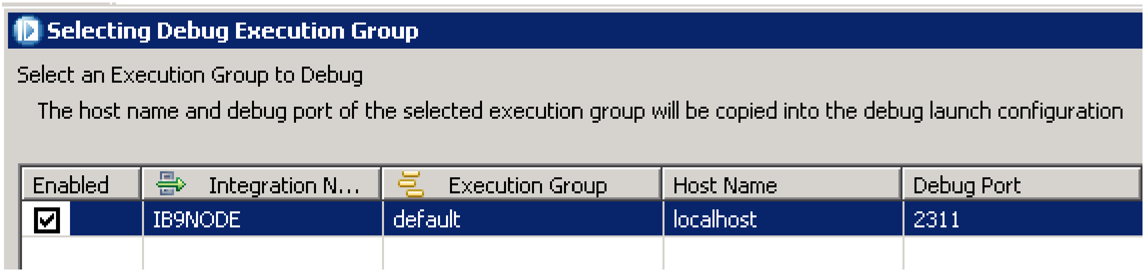
Connect | Source | Common

Host Name: ws2008r2x64

Java Debug Port: 2311

Select Execution Group...

__ e. Verify that **Enabled** is selected for the row in the table for integration node **IB9NODE** with integration server **default**, and then click **Cancel**.


Selecting Debug Execution Group

Select an Execution Group to Debug

The host name and debug port of the selected execution group will be copied into the debug launch configuration

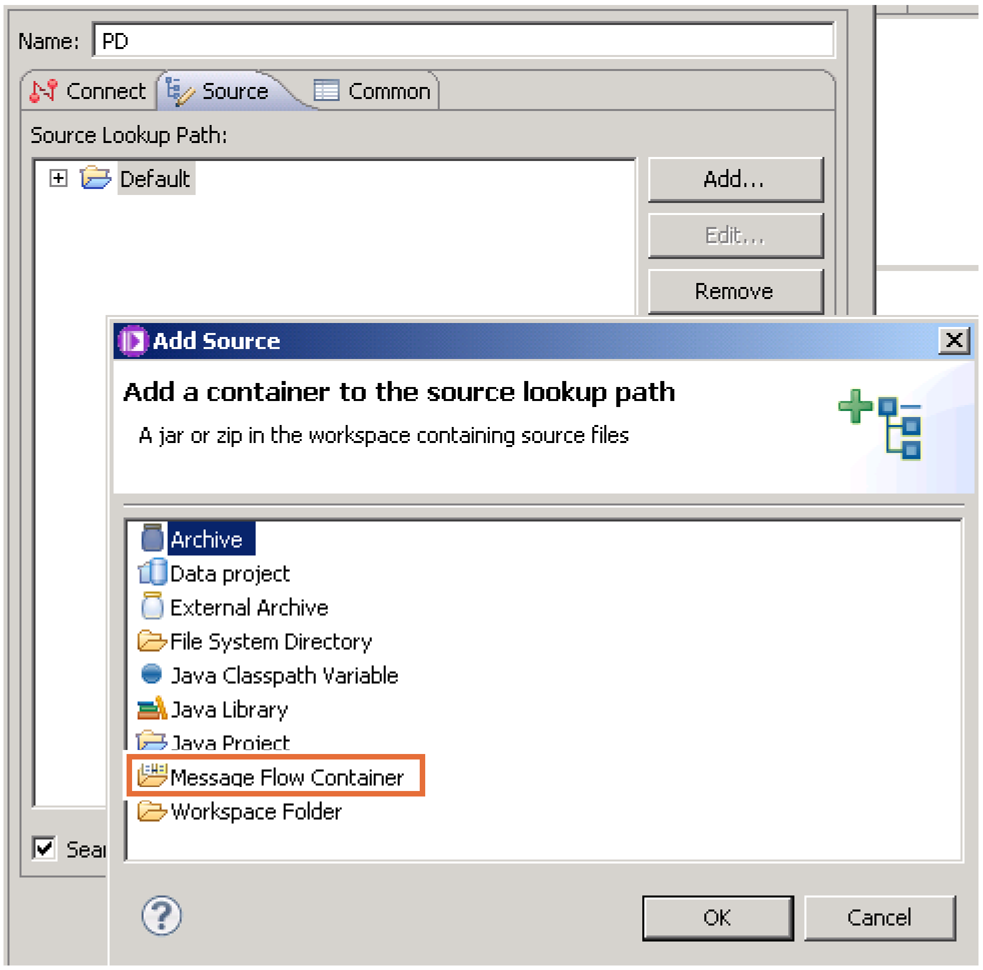
Enabled	Integration N...	Execution Group	Host Name	Debug Port
☑	IB9NODE	default	localhost	2311

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

__ f. Click the **Source** tab.

__ g. Click **Add**.

__ h. Select **Message Flow Container,** and then click **OK**.



Add a container to the source lookup path

A jar or zip in the workspace containing source files

The Project Select menu is displayed.

__ i. Select **ProblemDetermination**, and then click **OK**.

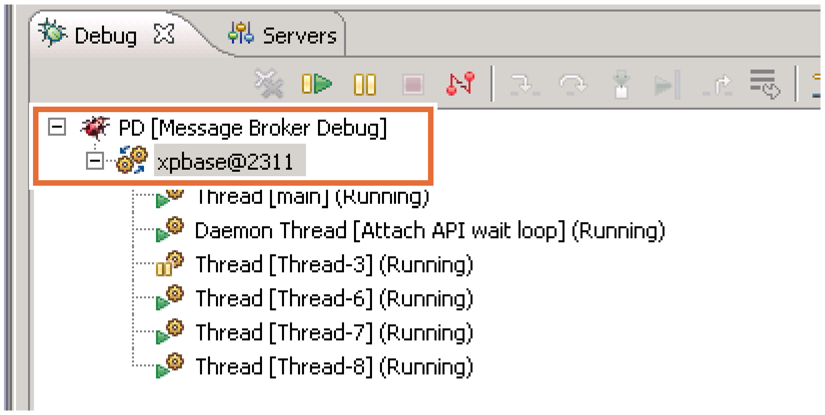
__ j. Click **Apply**.

__ k. Click **Close**.

The debugging session starts and the name of the debug session that is created is shown in the view with the host name and debug port (2311).

Note

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



```
PD [Message Broker Debug]
  xpbase@2311
    Thread [main] (Running)
    Daemon Thread [Attach API wait loop] (Running)
    Thread [Thread-3] (Running)
    Thread [Thread-6] (Running)
    Thread [Thread-7] (Running)
    Thread [Thread-8] (Running)
```

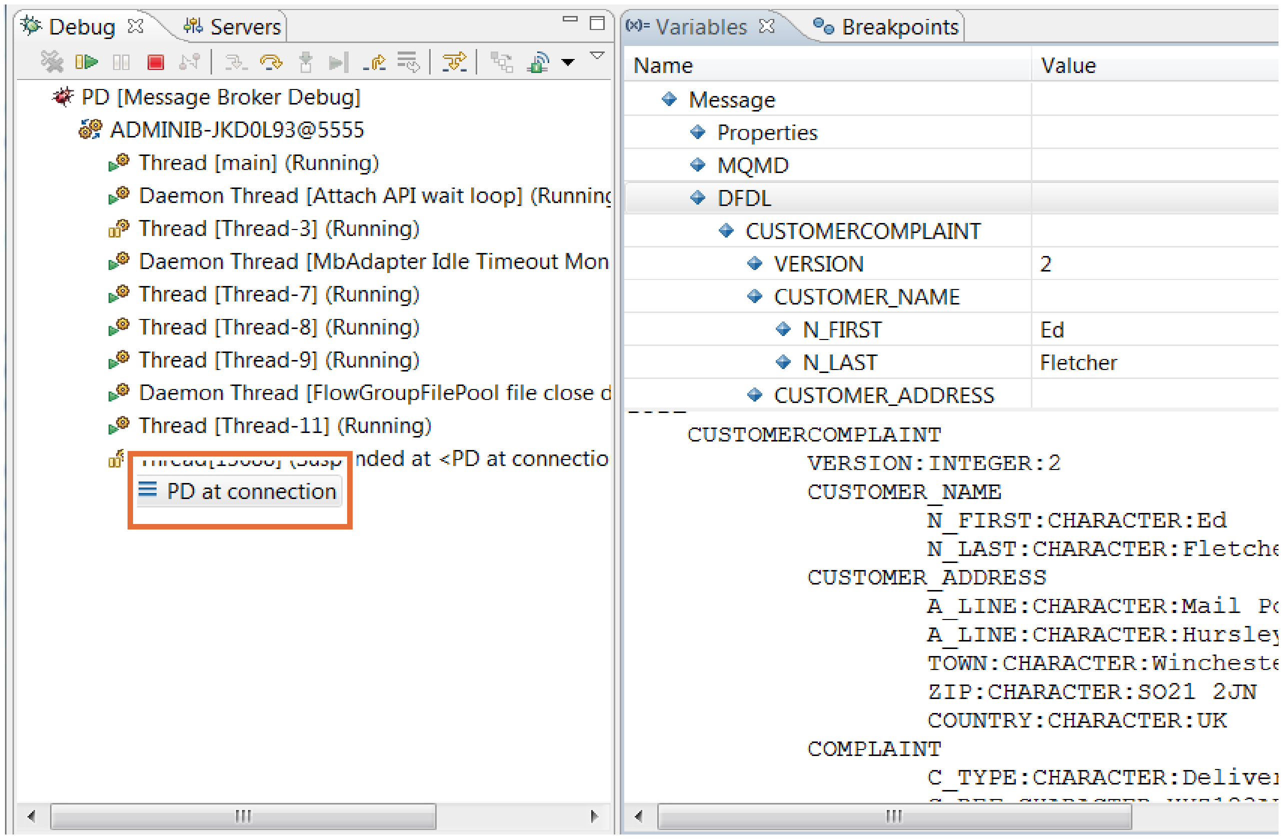
__ 4. Using the RFHUtil session that is already running for queue COBOL_IN, send a test message by clicking **Write Q**.

__ 5. In the IBM Integration Toolkit, the **Debug** view shows that the message flow is suspended.

__ 6. Select **PD at connection** in the **Debug** view.

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

Expand **Message > DFDL** and a few of the subordinate components such as **CUSTOMER_NAME** and **CUSTOMER_ADDRESS**.

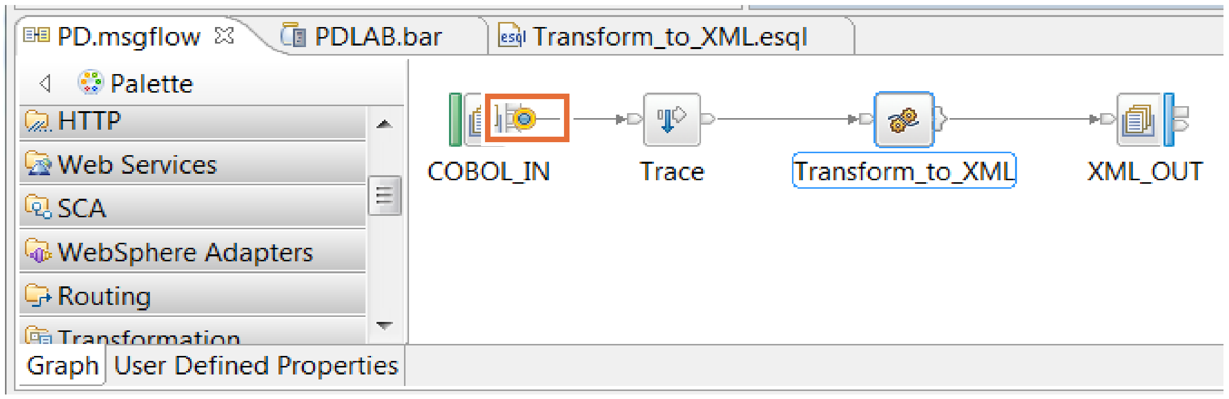


```
PD [Message Broker Debug]
  ADMINIB-JKD0L93@5555
    Thread [main] (Running)
    Daemon Thread [Attach API wait loop] (Running
    Thread [Thread-3] (Running)
    Daemon Thread [MbAdapter Idle Timeout Mon
    Thread [Thread-7] (Running)
    Thread [Thread-8] (Running)
    Thread [Thread-9] (Running)
    Daemon Thread [FlowGroupFilePool file close d
    Thread [Thread-11] (Running)
    Thread [15008] (Susp nded at <PD at connectio
      PD at connection
```

Name	Value
Message	
Properties	
MQMD	
DFDL	
CUSTOMERCOMPLAINT	
VERSION	2
CUSTOMER_NAME	
N_FIRST	Ed
N_LAST	Fletcher
CUSTOMER_ADDRESS	

```
----
    CUSTOMERCOMPLAINT
            VERSION:INTEGER:2
            CUSTOMER_NAME
                    N_FIRST:CHARACTER:Ed
                    N_LAST:CHARACTER:Fletche
            CUSTOMER_ADDRESS
                    A_LINE:CHARACTER:Mail Po
                    A_LINE:CHARACTER:Hursley
                    TOWN:CHARACTER:Wincheste
                    ZIP:CHARACTER:SO21 2JN
                    COUNTRY:CHARACTER:UK
            COMPLAINT
                    C_TYPE:CHARACTER:Delive
```

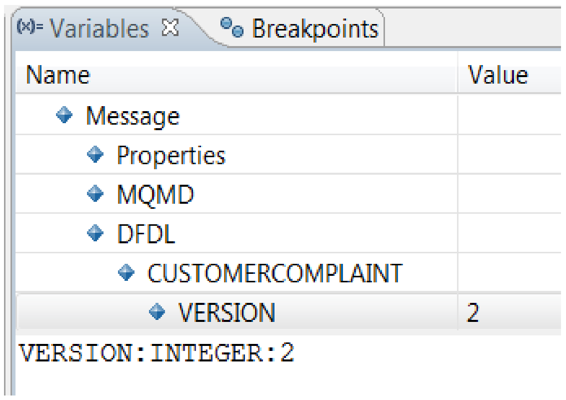
The message flow is shown in the lower half of the screen, with the breakpoint that paused the flow, as indicated by the blue circle on the wire between the MQInput and Trace nodes. The yellow circle around the breakpoint indicator shows that it is the current breakpoint at which flow is halted.



__ 7. Update the message.

__ a. In the **Variables** view on the upper-right side, expand the **Message** tree.

__ b. It is possible to modify the value of a message field while running a message flow in the Debug perspective. For example, right-click **Message.DFDL.VERSION**, type over its current value (2) with another value, and then click **OK**.



Name	Value
Message	
Properties	
MQMD	
DFDL	
CUSTOMERCOMPLAINT	
VERSION	2

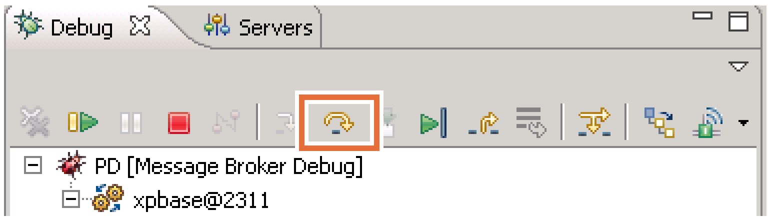
```
VERSION:INTEGER:2
```

Note

In this exercise, this modification does not change the runtime results. It just shows you that you can change a value during debugging when necessary. Later in this course, you might find it useful to change a value while you are debugging a message flow to test a Route node, for example.

__ 8. Step through the flow.

__ a. Use the **Step** icons in the Debugger toolbar or right-click anywhere in the **Flow Debug** view and select **Step over**. You can also press **F6** to step over.



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

__ b. Click **Step over** again (or press **F6**), or click **Resume** (or press **F8**). The message flow resumes until the next breakpoint is encountered.

__ c. The next breakpoint is encountered in the Compute node, where you set the breakpoint in the ESQL code. Examine the **Variables** view.

The incoming message is shown as **WMQI_DebugMessage**, and the new (transformed) output message is shown in OutputRoot.

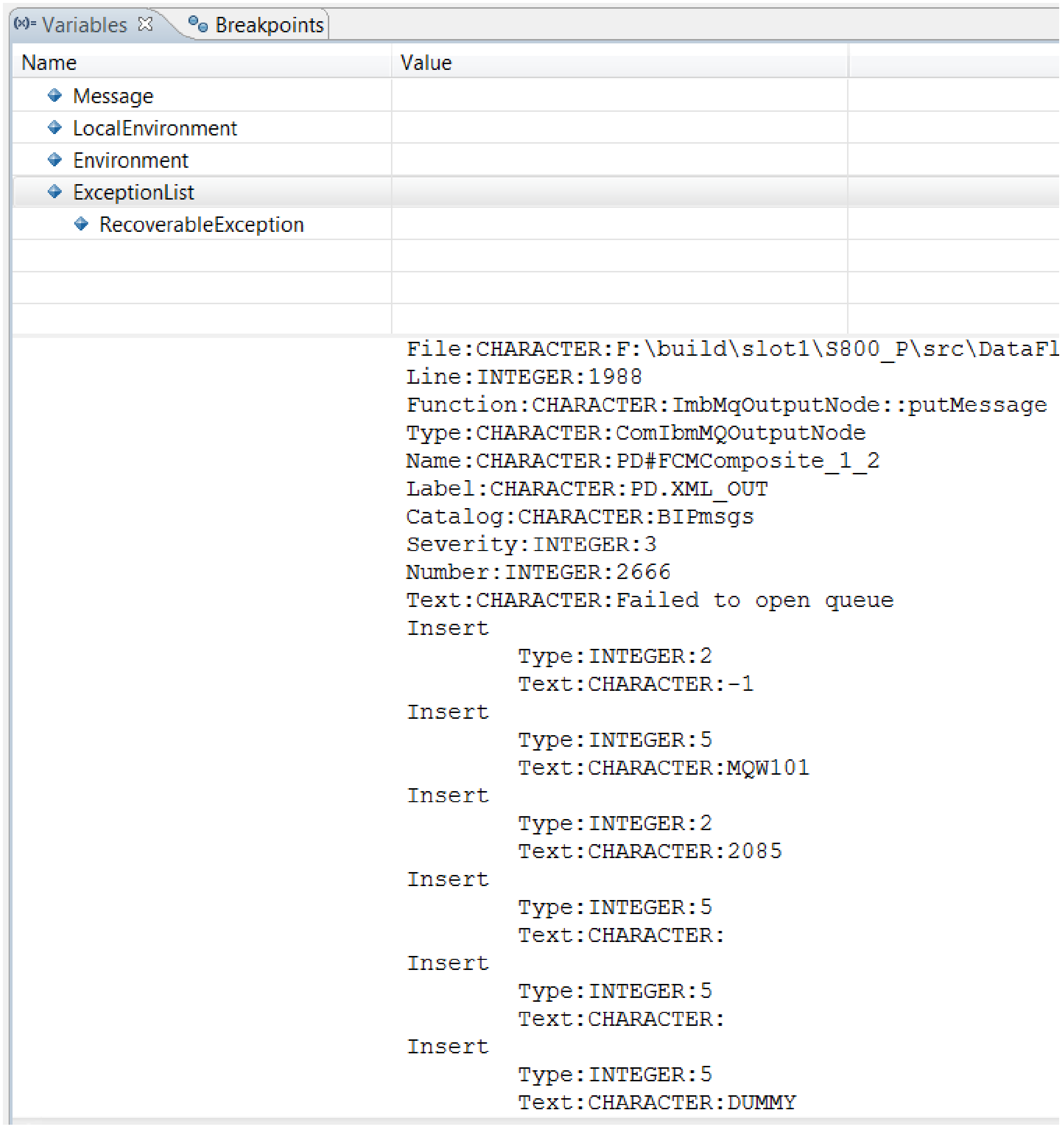
__ d. Click **Resume** again to continue execution.

Questions

Did you observe the runtime error while stepping through the message flow?

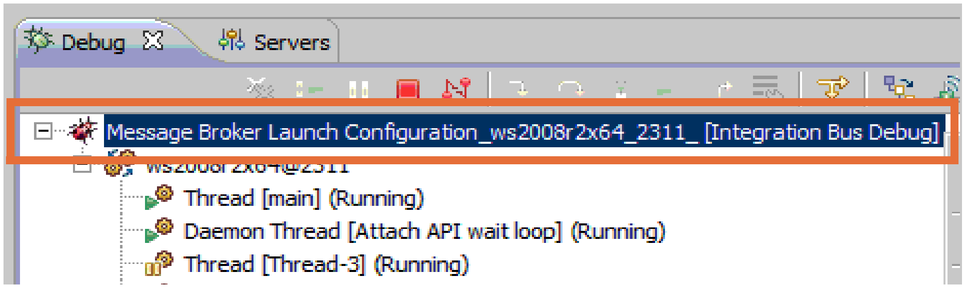
What do you see?

On the **Variables** tab, expand the ExceptionList to find the cause of the exception.



```
File:CHARACTER:F:\build\slot1\S800_P\src\DataFl
Line:INTEGER:1988
Function:CHARACTER:ImbMqOutputNode::putMessage
Type:CHARACTER:ComIbmMQOutputNode
Name:CHARACTER:PD#FCMComposite_1_2
Label:CHARACTER:PD.XML_OUT
Catalog:CHARACTER:BIPmsgs
Severity:INTEGER:3
Number:INTEGER:2666
Text:CHARACTER:Failed to open queue
Insert
        Type:INTEGER:2
        Text:CHARACTER:-1
Insert
        Type:INTEGER:5
        Text:CHARACTER:MQW101
Insert
        Type:INTEGER:2
        Text:CHARACTER:2085
Insert
        Type:INTEGER:5
        Text:CHARACTER:
Insert
        Type:INTEGER:5
        Text:CHARACTER:
Insert
        Type:INTEGER:5
        Text:CHARACTER:DUMMY
```

__ 9. Disconnect from the debugger by right-clicking the first line in the **Debug** view and then selecting **Disconnect**.

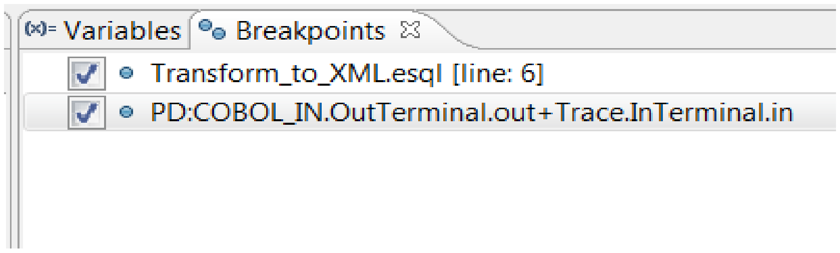

Debug
Servers
Message Broker Launch Configuration_ws2008r2x64_2311_ [Integration Bus Debug]
ws2008r2x64@2311
Thread [main] (Running)
Daemon Thread [Attach API wait loop] (Running)
Thread [Thread-3] (Running)

Optional

You can also select **Terminate and Remove**.

__ 10. Remove all breakpoints.

__ a. Switch to the **Breakpoints** view (to the right of the **Variables** view). The list of current breakpoints is displayed.



Transform_to_XML.esql [line: 6]
PD:COBOL_IN.OutTerminal.out+Trace.InTerminal.in

__ b. Right-click anywhere in the **Breakpoints** view and select **Remove all**.

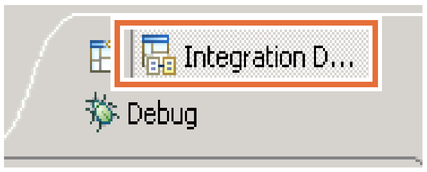
__ c. Click **Yes** to confirm the removal of the breakpoints.

Part 9: Use Component Trace in the Test Client

In this part of the exercise, you use the Component Trace facility of the Test Client.

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

__ a. Switch back to the Integration Development perspective.



__ b. In the Message Flow editor, right-click the MQInput node (COBOL_IN) and then select **Test**.

__ c. Click **OK** to the confirmation message that reminds you that an application owns the message flow.

__ d. The Test Client view opens. Click **Import Source**.

__ e. Browse to `C:\Labs\Lab04-ProblemDetermination\data`, select `Complaint_cwf.txt`, and then click **Open**.

__ 2. Configure the Test Client for Component Trace to use the existing BAR file.

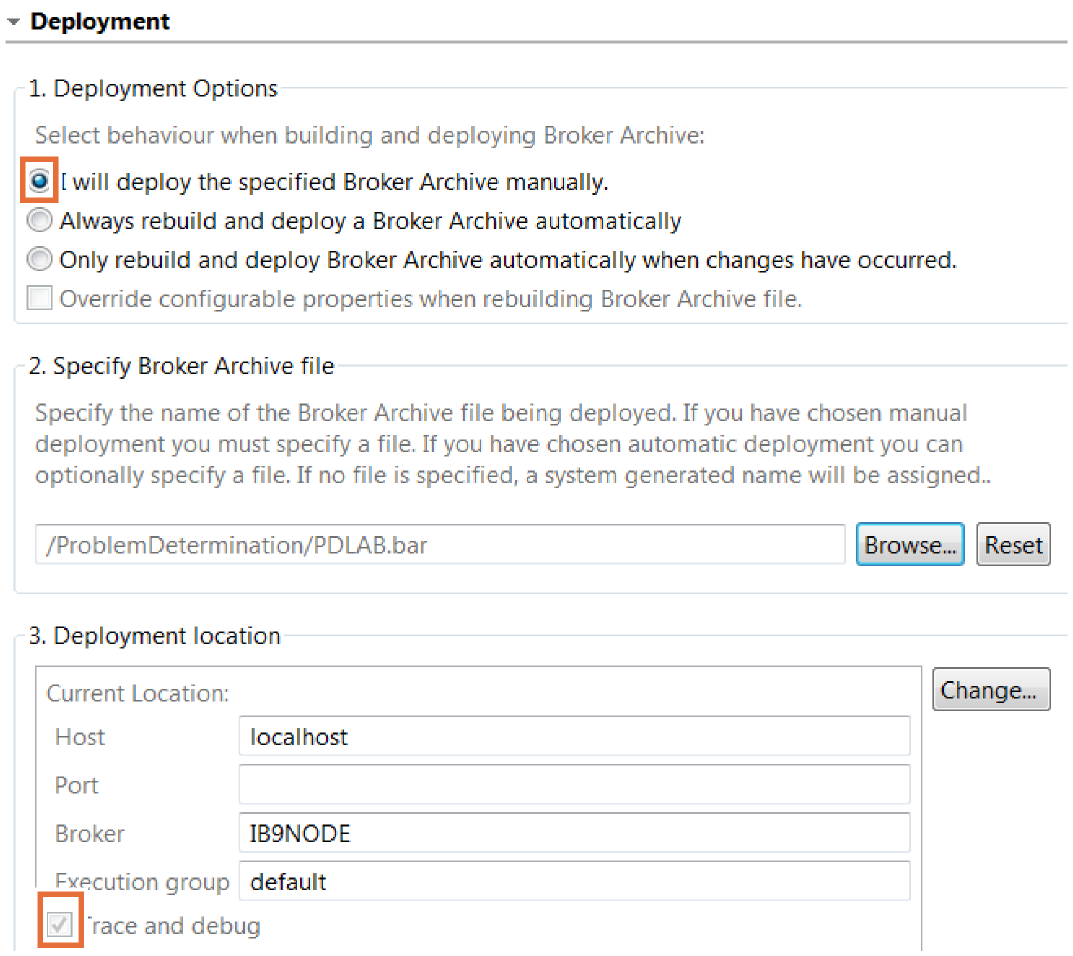
__ a. Click the **Configuration** tab.

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

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

__ d. In the Deployment Location section, click **Change**.

__ e. Select **Trace and debug**, and then click **Next**.



Deployment

1. Deployment Options

Select behaviour when building and deploying Broker Archive:

[✓ I will deploy the specified Broker Archive manually.]
Always rebuild and deploy a Broker Archive automatically
Only rebuild and deploy Broker Archive automatically when changes have occurred.
Override configurable properties when rebuilding Broker Archive file.

2. Specify Broker Archive file

Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

/ProblemDetermination/PDLAB.bar

3. Deployment location

Current Location:

Host	localhost
Port	
Broker	IB9NODE
Execution group	default

[✓ Trace and debug]

__ f. Clear the option to **Create queues of input and output nodes of message flows when hostname is localhost** if it is set.

In this exercise, you want the message flow to fail so that you can use the problem determination tools. If you do not clear this option, the Test Client automatically creates the DUMMY queue and the message flow does not fail.

__ g. Click **Finish**.

__ 3. Send a test message.

__ a. Click the **Events** tab of the Test Client.

__ b. Click **Send Message**.

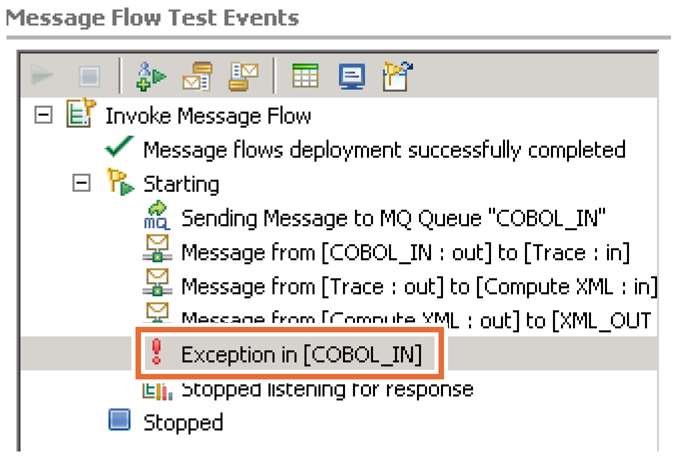
The debugger starts automatically. However, because you removed all of the breakpoints, the flow was not suspended.

__ 4. If you are prompted to switch to the debug perspective, click **No**.

In most cases, you would choose **Yes**, but because no breakpoints are set, switching to the debug perspective is not necessary.

__ 5. Review the Component Trace output.

__ a. Review the Message Flow Test Events in the Test Client **Events** tab. Observe the exception that is shown in the event list.



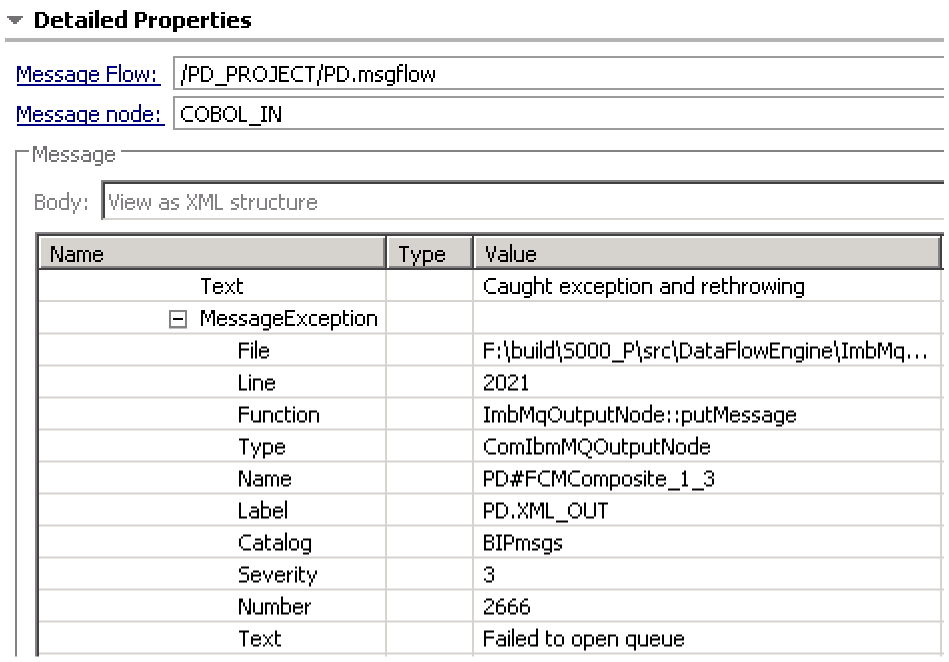
Message Flow Test Events

- Invoke Message Flow
  - Message flows deployment successfully completed
  - Starting
    - Sending Message to MQ Queue "COBOL_IN"
    - Message from [COBOL_IN : out] to [Trace : in]
    - Message from [Trace : out] to [Compute XML : in]
    - Message from [Compute XML : out] to [XML_OUT
    - Exception in [COBOL_IN]
    - Stopped listening for response
  - Stopped

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

__ c. Expand the WMQI_ExceptionList. You see several nested exceptions. Recall that it is generally the innermost exception that describes the error that occurred.

__ d. Review the innermost exception.



Detailed Properties

Message Flow: /PD_PROJECT/PD.msgflow

Message node: COBOL_IN

Message

Body: View as XML structure

Name	Type	Value
Text		Caught exception and rethrowing
MessageException		
File		F:\build\S000_P\src\DataFlowEngine\ImbMq...
Line		2021
Function		ImbMqOutputNode::putMessage
Type		ComIbmMQOutputNode
Name		PD#FCMComposite_1_3
Label		PD.XML_OUT
Catalog		BIPmsgs
Severity		3
Number		2666
Text		Failed to open queue

__ e. Review the **Insert** messages too. One of them shows the 2085 status code, which you reviewed earlier.

Part 10: Correct the problem and retest

After you identified your error with the problem determination tools demonstrated in this exercise, the next step is to correct the source code (or the BAR file configuration, as in this exercise). You can then redeploy and retest it.

- There are multiple ways in which the problem can be solved. It is your choice as to how you solve it.
- Take the appropriate means to redeploy the flow.
- Retest the corrected flow with a problem determination tool of your choice.

Part 11: 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 then click **Delete > All Flows and Resources**.

__ 3. In the **Integration Nodes** view, right-click the **default** integration server and click **Terminate Debugger**.

End of exercise

Exercise 5. Implementing a message flow

What this exercise is about

In this exercise, you create a message flow that uses ESQL and a Compute node or Java and a JavaCompute node to transformation a message.

What you should be able to do

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

- Create a message flow
- Use a Compute node or JavaCompute node in a message flow to transform a message

Introduction

In this exercise, you create a simple messaging framework for the processing of incoming complaint messages. In this application, a user completes a web-based complaint form, which arrives as an incoming XML message. In this exercise, you create a message flow that:

- Reads the XML message from a WebSphere MQ queue that is named COMPLAINT_IN
- Generates a unique complaint ID
- Determines which department is to process the message, which is based on the complaint type
- Writes the XML message to the WebSphere MQ queue that is named COMPLAINT_OUT

Subsequent exercises build on this framework, by adding routing and processing logic to the complaint handling process.

Requirements

- A workstation with the IBM Integration Bus components installed and the default configuration
- The files that are required for this exercise are in `C:\Labs\Lab05-Compute` and `C:\Labs\Tools`

- COMPLAINT_IN and COMPLAINT_OUT local queues on IB9QMGR

Sample input message:

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

Sample output message:

The sample output message includes the input message, followed by an administrative reference number and the responsible department:

```
<CUSTOMERCOMPLAINT>
   <VERSION>1</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <COUNTRY>UK</COUNTRY>
   </CUSTOMER_ADDRESS>
   <COMPLAINT>
      <C_TYPE>Delivery</C_TYPE>
      <C_REF>XYZ123ABC</C_REF>
      <C_TEXT>My order was delivered in time, but the package was torn and
      dirty.</C_TEXT>
   </COMPLAINT>
   <ADMIN>
      <REFERENCE>COMda1b71b2-f7fd-49e5-addb-ac9062f490c2</REFERENCE>
      <DEPT>B01</DEPT>
   </ADMIN>
</CUSTOMERCOMPLAINT>
```

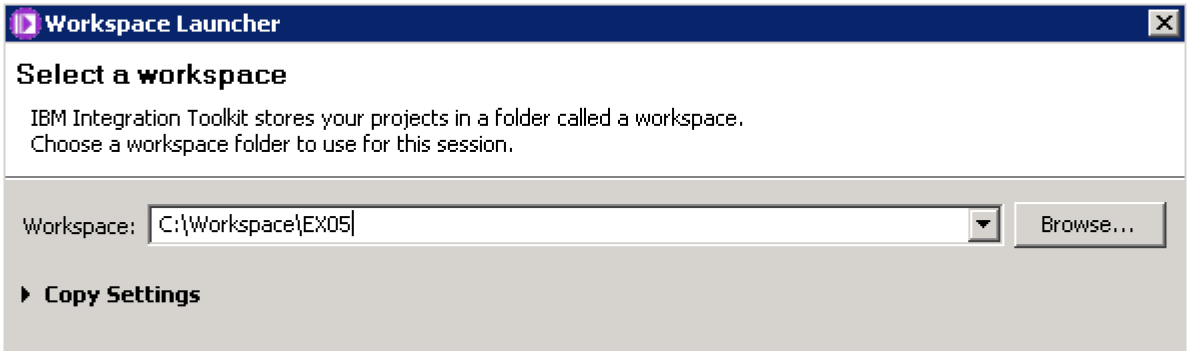
Exercise instructions

Part 1: Start components and clean up the workspace

__ 1. Open a new workspace.

__ a. In the IBM Integration Toolkit, select **File > Switch Workspace > Other** from the menu bar.

__ b. In the Workspace Launcher, enter: `c:\Workspace\Ex05`



IBM Integration Toolkit stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Workspace\EX05

__ c. Click **OK**. It might take a minute to open IBM Integration Toolkit with the new workspace.

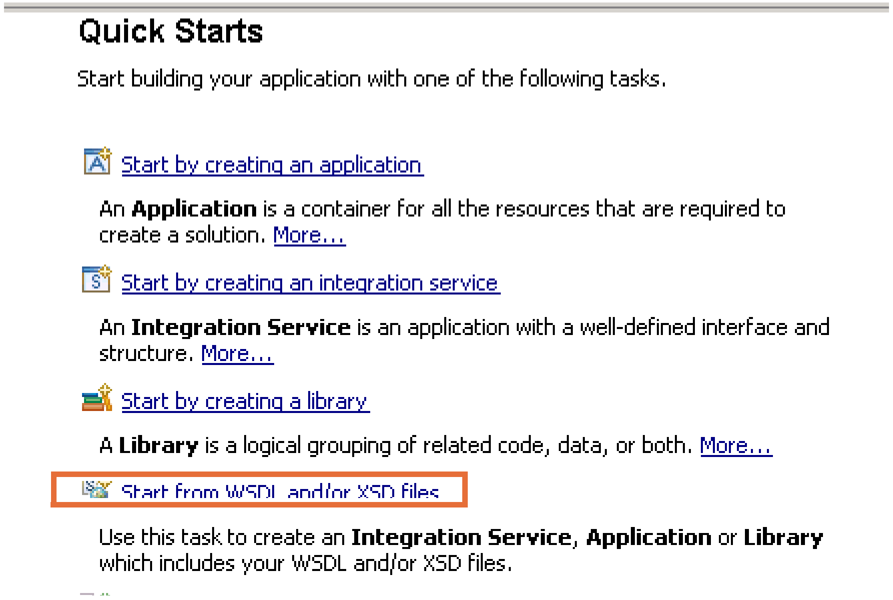
__ d. Close the Welcome pane when it is displayed (or click **Go to Integration Toolkit**) to open the Integration Development perspective.

Part 2: Build a message flow

In this part of the exercise, you import an XML Schema Definition (XSD) file as the basis for the IBM Integration Bus application. You also add processing nodes to the message flow.

__ 1. Create an IBM Integration Bus application.

__ a. Under the **Quick Starts** section, click **Start from WSDL and/or XSD files**.



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 an integration service

An **Integration Service** is an application with a well-defined interface and structure. More...

Start by creating a library

A **Library** is a logical grouping of related code, data, or both. More...

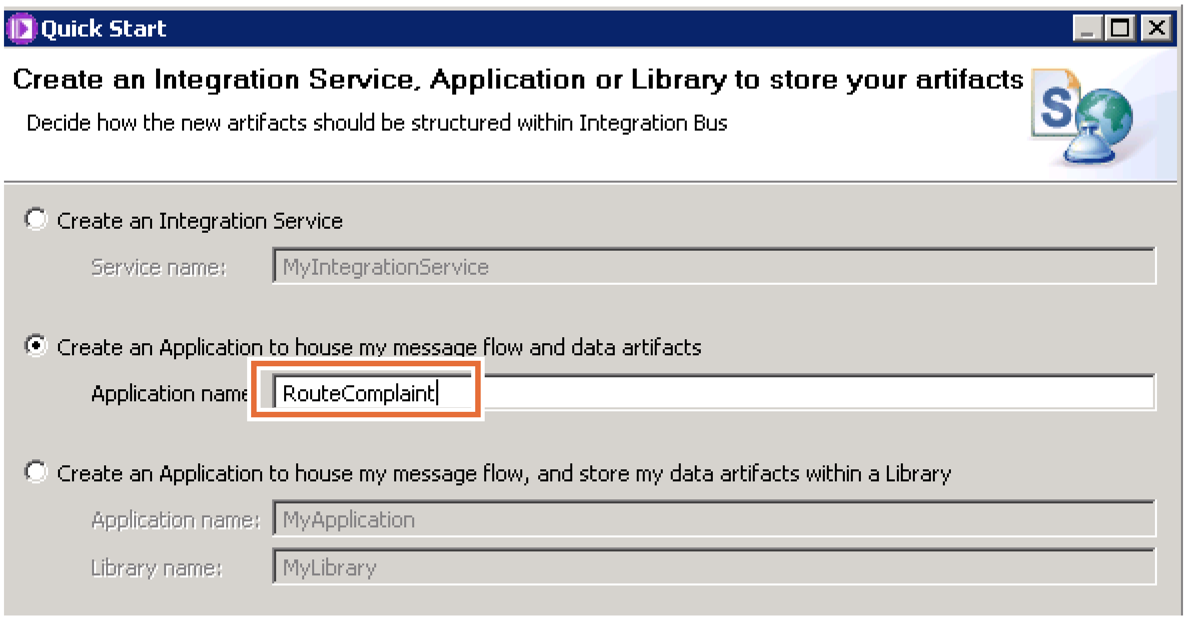
Start from WSDL and/or XSD files

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

The **Create an Application or Library to store your artifacts** menu opens.

__ b. Select **Create an Application to house my message flow and data artifacts**.

__ c. For **Application name**, enter `RouteComplaint` and then click **Next**.



Create an Integration Service, Application or Library to store your artifacts

Decide how the new artifacts should be structured within Integration Bus

○ Create an Integration Service

Service name: MyIntegrationService

● Create an Application to house my message flow and data artifacts

Application name: RouteComplaint

○ Create an Application to house my message flow, and store my data artifacts within a Library

Application name: MyApplication

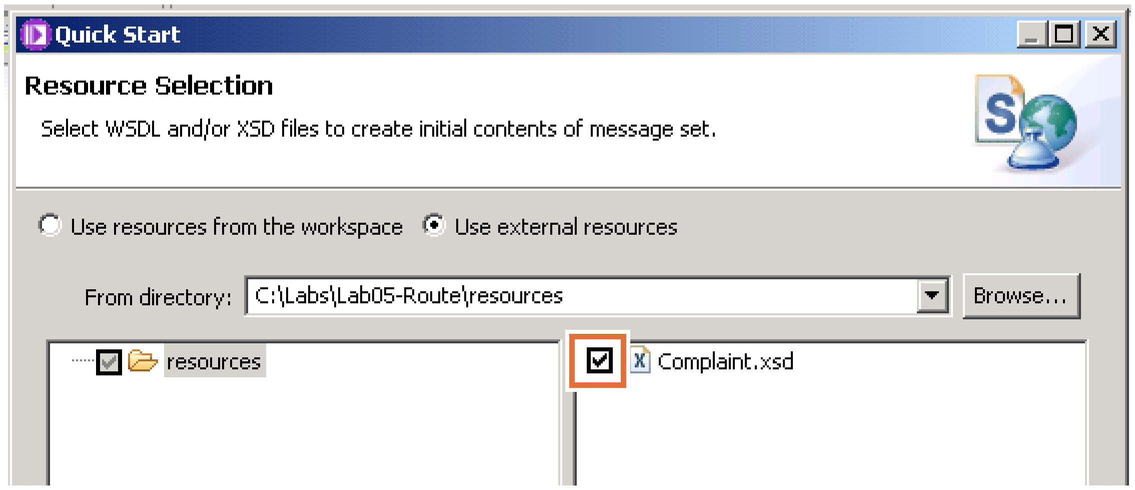
Library name: MyLibrary

The **Resource Selection** window opens.

__ d. Select **Use external resources**.

___ e. Click **Browse** to the right of **From directory**; then browse to `C:\Labs\Lab05-Compute\resources`, and then click **OK**.

___ f. Select **Complaint.xsd**.



Quick Start

Resource Selection

Select WSDL and/or XSD files to create initial contents of message set.

Use resources from the workspace [✓ Use external resources]

From directory: C:\Labs\Lab05-Route\resources

___ g. Click **Finish**. The schema definition is imported, and the application is created in the Application Development view.

___ 2. Add the following nodes to **Myflow.msgflow**: one **MQInput** node, one **Compute** or **JavaCompute** node, and one **MQOutput** node.

Note

Select either the Compute node or the JavaCompute node. You do not need to select both. If time is available, you can implement the other node after you complete the exercise.

- The **MQInput** and **MQOutput** nodes are in the **WebSphere MQ** drawer.
- The **Compute** and **JavaCompute** nodes are in the **Transformation** drawer.

To add a node to the canvas, select it from the appropriate drawer and drag it to the canvas, or select it from drawer and then click in the canvas. Do not be concerned about the positioning of the nodes; you fix that in a subsequent step.

___ 3. Rename the nodes. To rename a node, select it, and then switch to the **Properties** tab. Change the **Node name** field on the **Description** tab.

___ a. Rename the MQInput node to: `COMPLAINT_IN`

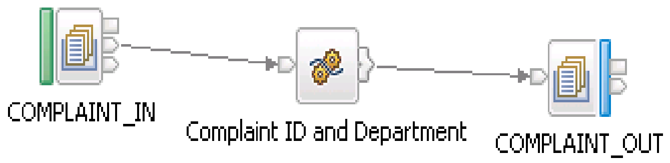
___ b. Rename the MQOutput node to: `COMPLAINT_OUT`

___ c. Rename the Compute or JavaCompute node to: `Complaint ID and Department`

__ 4. Wire the nodes as follows.

  __ a. Wire `COMPLAINT_IN` **Out** terminal to `Complaint ID and Department` **In** terminal.

  __ b. Wire `Complaint ID and Department` **Out** terminal to `COMPLAINT_OUT` **In** terminal.



__ 5. Configure the properties for MQInput node **COMPLAINT_IN**:

  __ a. On the **Basic** tab, set **Queue name** to: `COMPLAINT_IN`

  __ b. On the **Input Message Parsing** tab, set the **Message Domain** to: `XMLNSC`

__ 6. Configure the properties for MQOutput node **COMPLAINT_OUT**.

On the **Basic** tab, set the **Queue name** to: `COMPLAINT_OUT`

Next, you configure the **Complaint ID and Department** processing. You implement this configuration by using either a Compute node or a JavaCompute node. Perform one or the other of the two steps that follow, but not both.

Optional

If time remains after you implement and test the message flow with one of the compute nodes, remove the node and substitute the other.

Option 1: Write a transformation by using a Compute node and ESQL

If you are using a Compute node in your message flow, the next step is to create the ESQL to transform the message.

__ 1. On the Compute node **Complaint ID and Department**, set the **ESQL module** to `Compute_Complaint_ID_and_Department` on the **Basic** tab.

__ 2. Write ESQL for the **Complaint ID and Department** Compute node to copy the entire message.

  __ a. Double-click the **Complaint ID and Department** Compute node. The ESQL editor opens a new module.

The logic that you are adding:

- Copies the input message to the output message.
- Creates an element in the output message, `OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE`. The value for this element is computed by concatenating the string “`COM`” with a unique identifier that the ESQL function `UUIDASCHAR` generates. This function returns universally unique identifiers (UUIDs) as CHARACTER values. It acts as a type of random generator of character strings.
- Creates another element in the output message, `OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT`. The value for this element is based on the content of `InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE`, as follows:
  - If the complaint is about an order (C_TYPE = “Order”), then the department B01 is assigned to handle the complaint.
  - If the complaint is about a delivery (C_TYPE = “Delivery”), then the department C01 is assigned to handle the complaint.
  - All other complaint types are assigned to department E01.

Note

The middle character in the department (`E01`) is a zero, ***not*** the letter O.

___ b. Copy the ESQL from `C:\Labs\Lab05-Route\resources\Cut&Paste.txt`.

The file contains ESQL code for the Compute node and a Java method for the JavaCompute node.

Copy only the ESQL portion of code and paste it over the entire existing ESQL module (completely replacing it). The contents of the ESQL module should match the code that is shown here.

```
CREATE COMPUTE MODULE Compute_Complaint_ID_and_Department
  CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN

      SET OutputRoot = InputRoot;

      /* create the complaint reference ID by using the UUIDASCHAR function */
      SET OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE =
      'COM' || UUIDASCHAR;

      /* assign the department to handle the complaint based on
      complaint type */
      SET OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT =
        CASE InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE
           WHEN 'Order' THEN 'B01'
           WHEN 'Delivery' THEN 'C01'
           ELSE 'E01'
        END;  /* case */

      RETURN TRUE;  /* propagate message to Out terminal */

    END;  /* create function */
END MODULE;
```

___ c. Save the ESQL file by typing **Ctrl + S**.

___ d. Close the ESQL editor.

___ e. Save the message flow by typing **Ctrl + S**.

Option 2: Write a transformation in the JavaCompute node

__ 1. In the JavaCompute node **Complaint ID and Department,** set the **Java class** to `ComputeComplaintIDandDepartment` on the **Basic** tab.

__ 2. Create a JavaCompute node project where you store the classes that are used in the JavaCompute nodes.

__ a. Double-click the **JavaCompute** node. The **New Java Compute Node Class** wizard is displayed. This wizard helps you create a Java project. By default, the project is named **RouteComplaintJava**.

__ b. Click **Next**. The Java Compute Node Class Template opens.

__ c. In this exercise you modify the message in this JavaCompute node, select **Modifying message class,** and then click **Finish**. The Java editor opens.

The logic that you are adding:

- Copies the input message to the output message.
- Creates an element in the output message, `OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE`. The value for this element is computed by concatenating the string “`COM`” with a unique identifier that the function `randomUUID` generates. This function returns universally unique identifiers (UUIDs) as CHARACTER values. It acts as a type of random generator of character strings.
- Creates another element in the output message, `OutputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT`. The value for this element is based on the content of `InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE`, as follows:
  - If the complaint is about an order (C_TYPE = “Order”), then the department B01 is assigned to handle the complaint.
  - If the complaint is about a delivery (C_TYPE = “Delivery”), then the department C01 is assigned to handle the complaint.
  - All other complaint types are assigned to department E01.

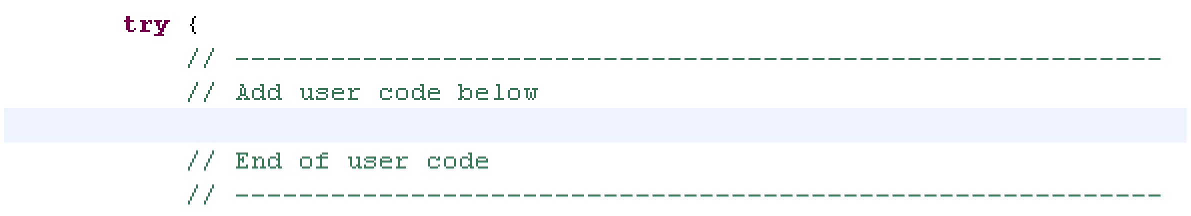
Note

The middle character in the department (E01) is a zero, ***not*** the letter O.

__ d. Copy the Java from `C:\Labs\Lab05-Route\resources\Cut&Paste.txt`.

The file contains ESQL code for the Compute node and a Java method for the JavaCompute node. Be sure to select only the Java code.

Insert the code that you copy from the file between the lines `// Add user code below` and `// End of user code` in the `try` block.

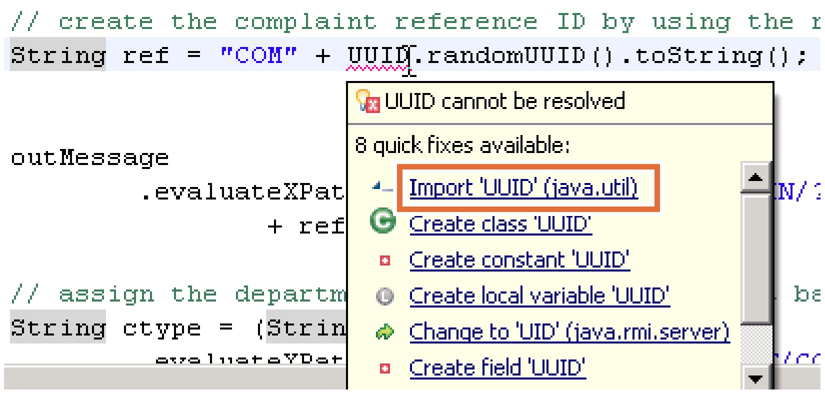


```
try {
    // ----------------------------------------------------------
    // Add user code below

    // End of user code
    // ----------------------------------------------------------
```

```
  // Add user code below

  String ref = "COM" + UUID.randomUUID().toString();

  outMessage
  .evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?REFERENCE
   [set-value('"+ ref + "')]");

  String ctype = (String) outMessage
  .evaluateXPath("string(CUSTOMERCOMPLAINT/COMPLAINT/C_TYPE)");

  if (ctype.equalsIgnoreCase("Order"))
  outMessage
  .evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?DEPT[set-value('B01')]");

  else if (ctype.equalsIgnoreCase("Delivery"))
  outMessage
  .evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?DEPT[set-value('C01')]");

  else
  outMessage
  .evaluateXPath("?CUSTOMERCOMPLAINT/?ADMIN/?DEPT[set-value('E01')]");

  // End of user code
```

__ e. When you paste the Java code into the editor, a red error decorator is shown in the left margin of the editor. If you hover the mouse cursor over the decorator, you see the message `UUID cannot be resolved`.

To fix this problem, click the UUID string in the code. A quick fix menu opens. Click **import 'UUID' (java.util)**.



```
// create the complaint reference ID by using the r
String ref = "COM" + UUID.randomUUID().toString();

outMessage
        .evaluateXPat
                + ref

// assign the departm                                  be
String ctype = (Strin
        evaluateXPat
```

UUID cannot be resolved

8 quick fixes available:

- Import 'UUID' (java.util)
- Create class 'UUID'
- Create constant 'UUID'
- Create local variable 'UUID'
- Change to 'UID' (java.rmi.server)
- Create field 'UUID'

The error decorator disappears.

__ f. Save the changes in the Java editor by typing **Ctrl + S**.

__ g. Close the Java editor.

__ h. Save the message flow by typing **Ctrl + S**.

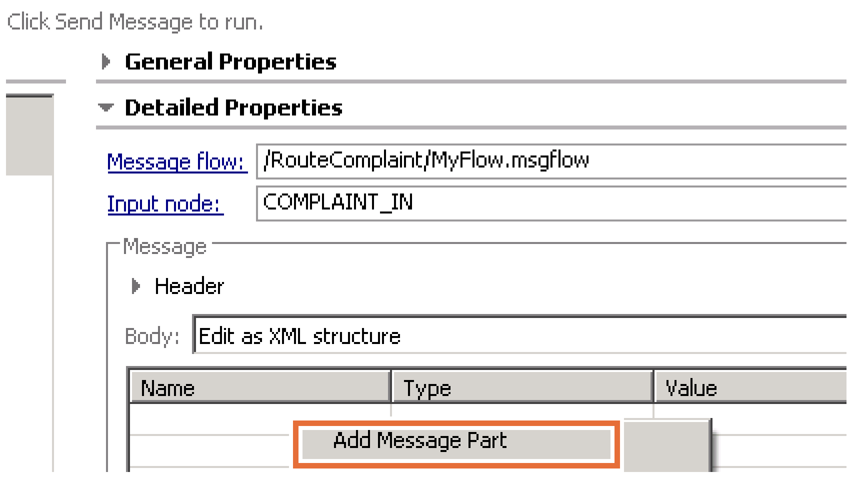
Part 3: Test with the Test Client

__ 1. Start the Test Client for the application.

In the Application Development view, right-click **RouteComplaint** and then select **Test**. The Test Client starts.

__ 2. The input node expects an XML message, and the message flow is associated with a message definition. Thus, the **Edit as XML structure** editor is available. Populate the CUSTOMERCOMPLAINT type with test data.

__ a. On the **Events** tab of the Test Client, right-click anywhere in the **Message** table (in the Name, Type, Value area), and then select **Add Message Part** from the menu.


Click Send Message to run.

General Properties

Detailed Properties

Message flow: /RouteComplaint/MyFlow.msgflow

Input node: COMPLAINT_IN

Message

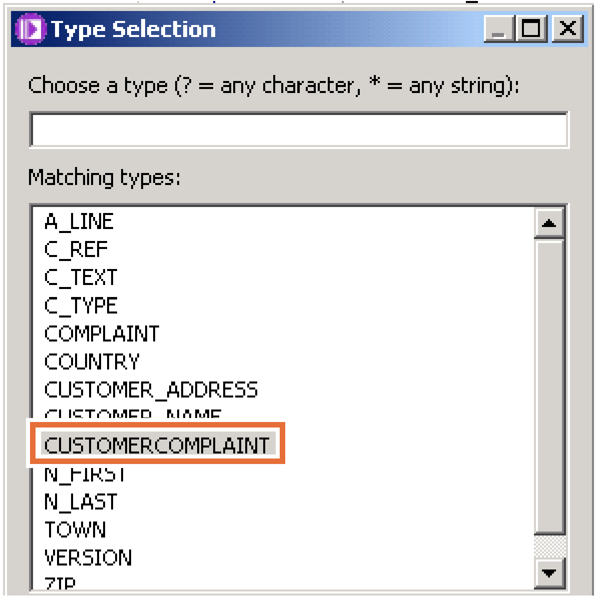
Header

Body: Edit as XML structure

Name	Type	Value

The **Type Selection** menu opens.

___ b. Select **CUSTOMERCOMPLAINT,** and then click **OK**.



Type Selection

Choose a type (? = any character, * = any string):

Matching types:

A_LINE
C_REF
C_TEXT
C_TYPE
COMPLAINT
COUNTRY
CUSTOMER_ADDRESS
CUSTOMER_NAME
CUSTOMERCOMPLAINT
N_FIRST
N_LAST
TOWN
VERSION
ZIP

The Message table is populated with the message definition that you imported by using the XSD file at the start of the exercise.

___ 3. The elements are filled with default values. You can use most of those values, but the message flow needs specific values for the **VERSION** (`2` or another value) and **C_TYPE** (`Order`, `Delivery`, `Other`) elements.

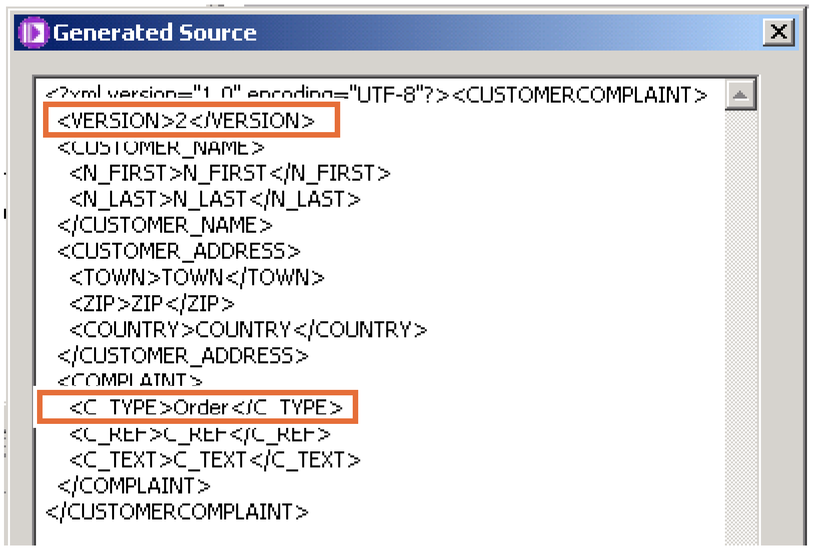
___ a. Right-click the **VERSION** element, and then select **Set Value** from the menu.

___ b. Enter `2` and click **OK**.

___ c. Right-click the **C_TYPE** element and select **Set Value**.

___ d. Type `Order` and click **OK**.

___ e. Click **Show Generated Source** below the Message table, and verify that `Version=2` and `C_TYPE=Order` are in the generated XML message.



Generated Source

```
<?xml version="1.0" encoding="UTF-8"?><CUSTOMERCOMPLAINT>
 <VERSION>2</VERSION>
 <CUSTOMER_NAME>
  <N_FIRST>N_FIRST</N_FIRST>
  <N_LAST>N_LAST</N_LAST>
 </CUSTOMER_NAME>
 <CUSTOMER_ADDRESS>
  <TOWN>TOWN</TOWN>
  <ZIP>ZIP</ZIP>
  <COUNTRY>COUNTRY</COUNTRY>
 </CUSTOMER_ADDRESS>
 <COMPLAINT>
  <C_TYPE>Order</C_TYPE>
  <C_REF>C_REF</C_REF>
  <C_TEXT>C_TEXT</C_TEXT>
 </COMPLAINT>
</CUSTOMERCOMPLAINT>
```

__ f. Click **OK**.

Optional

If you want to save this test data for later reuse, you can store it in the workspace data pool as follows:

__ 4. Save these values to the pool.

__ a. Right-click the top-level **CUSTOMERCOMPLAINT** element, and then select **Add Value to Pool** from the menu. The **Value Name** menu opens.

__ b. Enter any name for this test data, for example: `Route_Version2_Order`

__ c. Click **OK**.

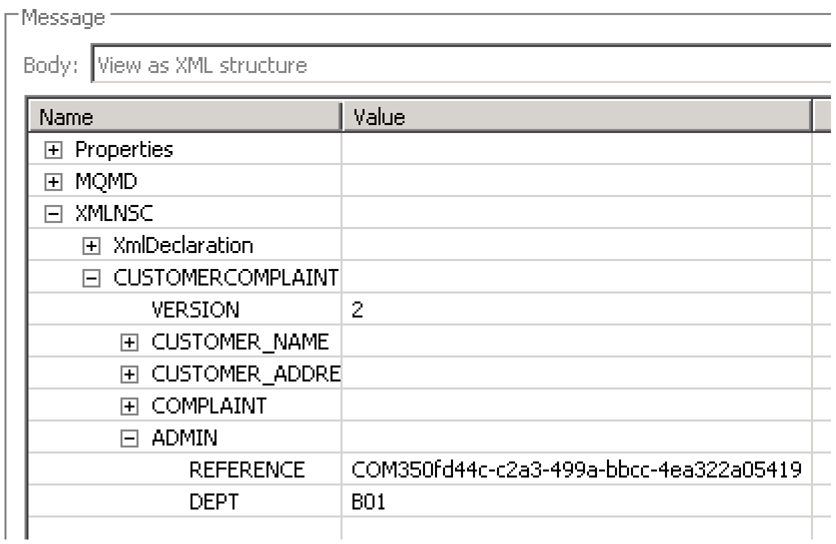
__ 5. In the Test Client, send a test message.

__ a. In the Test Client, click **Send Message**. The **Select Deployment Location** menu opens.

__ b. Select the **default** integration server.

__ c. Click **Finish**. The BAR file is built and deployed, and the test starts.

__ d. The flow should complete successfully. Verify that the output message has a new `ADMIN` structure at the end of the message, which includes a `REFERENCE` and `DEPT=B01`.



Name	Value
⊞ Properties	
⊞ MQMD	
⊟ XMLNSC	
⊞ XmlDeclaration	
⊟ CUSTOMERCOMPLAINT	
VERSION	2
⊞ CUSTOMER_NAME	
⊞ CUSTOMER_ADDRE	
⊞ COMPLAINT	
⊟ ADMIN	
REFERENCE	COM350fd44c-c2a3-499a-bbcc-4ea322a05419
DEPT	B01

Your `REFERENCE` value is expected to be different, since the `UUIDASCHAR` function generates a unique value.

Troubleshooting

If the test results are not what you expected, try to determine what went wrong. The Test Client Component Trace gives you valuable hints. Use other IBM Integration Bus debugging tools as appropriate.

If you cannot find a message on the COMPLAINT_OUT queue, try to find the reason by running the message flow with **UserTrace=Normal**. Retrieve user trace data by using the following script: `C:\Labs\Tools\GETTRACE.cmd`

You can also look for `BIPxxxx` messages in the Windows Event Viewer Application Log.

If the message is on the COMPLAINT_OUT queue but its content is not what you expect, run the message flow with **UserTrace=Debug**. Or, use the debugger to find out why the message was not transformed correctly.

Part 4: Optional: Finding ESQL errors

If you are not having any problems with deployment so far in this course, take the opportunity to become familiar with how IBM Integration Bus handles errors during deployment.

One of the reasons a deployment can fail is incorrect ESQL. Not all errors are signaled in the task list.

For example, a statement like

```
DECLARE V INT CARDINALITY(InputBody.CUSTOMERCOMPLAINT);
```

cannot deploy because `CARDINALITY` needs a list (expressed by `[]`) as its argument.

__ 1. Enter some erroneous ESQL into one of your message flows and deploy.

__ 2. Check the messages in the Deployment log to find out:

- That the deployment failed
- For integration server E
- Because of an error in message flow M
- In processing node P
- Exactly in ESQL line L, column C

You also find this information in the system log (Event Viewer, application log).

Part 5: Clean up the environment

__ 1. Close all open editors and the Test Client. 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

Exercise 6. Adding flow control to a message flow

What this exercise is about

In this exercise, you implement a Route node to a message flow to add flow control and wire the Failure and Catch terminals to capture exceptions.

What you should be able to do

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

- Use the Route node to control message processing
- Wire a Failure terminal and Catch terminals to handle exceptions

Introduction

In this exercise, you modify a message flow to route messages as determined by the version number in the message. When this exercise is complete, the message flow in this exercise:

- Reads the IBM WebSphere MQ message from a queue that is named COMPLAINT_IN
- Checks the version number in the message data and routes the message as determined by the version number
- Generates a unique complaint ID
- Determines which department is to process the message, which is based on the complaint type:
    - If the complaint is about an order (C_TYPE = “Order”), then the department B01 is assigned to handle the complaint.
    - If the complaint is about a delivery (C_TYPE = “Delivery”), then the department C01 is assigned to handle the complaint.
    - All other complaint types are assigned to department E01.
- Outputs the message to COMPLAINT_OUT or COMPLAINT_FALSE, depending on the result of the previous filter operation
- Enables debug port 2311 on the **default** integration server

Requirements

- A workstation with the IBM Integration Bus components installed
- Completed message flow from Lab Exercise 5 or an imported project interchange from the `C:\Labs\Lab05-Compute\Solutions` directory
- The files that are required for this exercise are in `C:\Labs\Lab06-Route` and `C:\Labs\Tools`
- The local queues COMPLAINT_IN, COMPLAINT, COMPLAINT_FAILURE, and COMPLAINT_FALSE on the IB9QMGR

Sample input message:

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

Sample output message:

The sample output message includes the input message, followed by an administrative reference number and the responsible department:

```
<CUSTOMERCOMPLAINT>
   <VERSION>1</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <COUNTRY>UK</COUNTRY>
   </CUSTOMER_ADDRESS>
   <COMPLAINT>
      <C_TYPE>Delivery</C_TYPE>
      <C_REF>XYZ123ABC</C_REF>
      <C_TEXT>My order was delivered in time, but the package was torn and
      dirty.</C_TEXT>
   </COMPLAINT>
   <ADMIN>
      <REFERENCE>COMda1b71b2-f7fd-49e5-addb-ac9062f490c2</REFERENCE>
      <DEPT>B01</DEPT>
   </ADMIN>
</CUSTOMERCOMPLAINT>
```

Exercise instructions

This exercise builds on the message flow that was created in Exercise 5, “Implementing a message flow.”

If you successfully completed Exercise 5, proceed to **Part 1, Modify the message flow**.

If you did not complete Exercise 5, start a new workspace and import one of the project interchange files in the `C:\Labs\Lab05-Compute\Solution` directory.

There are two solution project interchange files:

- `RouteComplaint_WithCompute_PI.zip` contains a message flow with a Compute node and ESQL code
- `RouteComplaint_WithJavaCompute_PI.zip` contains a message flow with a JavaCompute node and Java code.

Import only one of the project interchange files before you start this exercise.

Part 1: Modify the message flow

The starting message flow:

- Reads a WebSphere MQ message from a queue that is named COMPLAINT_IN
- Generates a unique complaint ID
- Determines which department is to process the message, which is based on the complaint type

The starting message flow contains three nodes:

- An MQInput node that is named COMPLAINT_IN that reads from a local queue that is named COMPLAINT_IN
- A Compute node or JavaCompute node that is named Complaint ID and Department. The Compute or JavaCompute node generates a unique complaint ID and determines which department is to process the message, which is based on the complaint type.
- An MQOutput node that is named COMPLAINT_OUT, which writes to a local queue that is named COMPLAINT_OUT

An imported XSD defines the message data.

In this part of the exercise, you add a Route node to the message flow to check the version number in the message data and route the message by version number. You also add two MQOutput nodes to the flow to handle message exceptions.

__ 1. Open the message flow, and add one Route node and two MQOutput nodes.

- The **MQOutput** node is in the **WebSphere MQ** drawer.
- The **Route** node is in the **Routing** drawer.

To add a node to the canvas, select it from the appropriate drawer and drag it to the canvas, or select it from drawer and then click in the canvas. Do not be concerned about the positioning of the nodes; you fix that in a subsequent step.

__ 2. Rename the nodes as follows. To rename a node, select it, and then switch to the **Properties** tab. Change the **Node name** field on the **Description** tab.

__ a. Rename the MQOutput nodes to: `COMPLAINT_FALSE` and `COMPLAINT_FAILURE`

__ b. Rename the Route node to: `Version=2?`

__ 3. Delete the wire between the **COMPLAINT_IN MQInput** node and the node that is named **Complaint ID and Department**.

__ 4. Wire the nodes as follows.

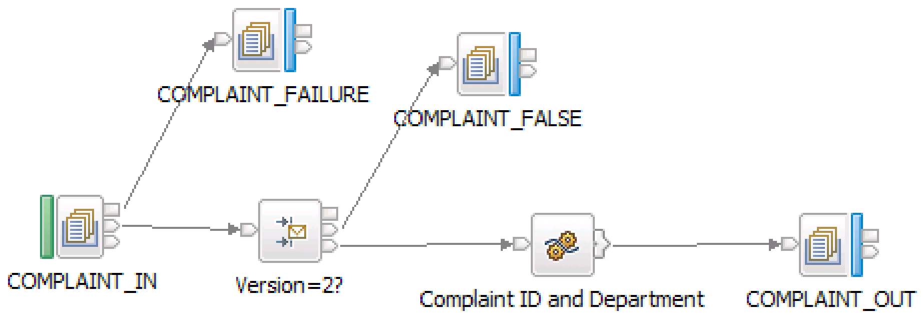
__ a. `COMPLAINT_IN` **Out** terminal to `Version=2?` **In** terminal.

__ b. `COMPLAINT_IN` **Failure** terminal to `COMPLAINT_FAILURE` **In** terminal.

__ c. `Version=2?` **Default** terminal to `COMPLAINT_FALSE` **In** terminal.

__ d. `Version=2?` **Match** terminal to `Complaint ID and Department` **In** terminal.

__ e. `Complaint ID and Department` **Out** terminal to `COMPLAINT_OUT` **In** terminal.



__ 5. Configure the properties for MQOutput node **COMPLAINT_FALSE**:

On the **Basic** tab, set **Queue name** to: `COMPLAINT_FALSE`

__ 6. Configure the properties for MQOutput node **COMPLAINT_FAILURE**:

On the **Basic** tab, set **Queue name** to: `COMPLAINT_FAILURE`

Important

Queue names are case-sensitive. Verify that the queue names that you entered in the node properties exactly match the queue names.

Part 2: Configure an XPath expression in the Route node

In this step, you configure the Route node so that any messages with `Version = 2` are sent to the **Match** terminal.

__ 1. Configure the Route node `Version=2?` with an XPath match expression.

__ a. In the Message Flow editor, select the **Route** node.

__ b. In the **Properties** view, select the **Basic** tab. The Filter Table is shown.

__ c. Click **Add**. The **Add Filter table entry** menu opens.

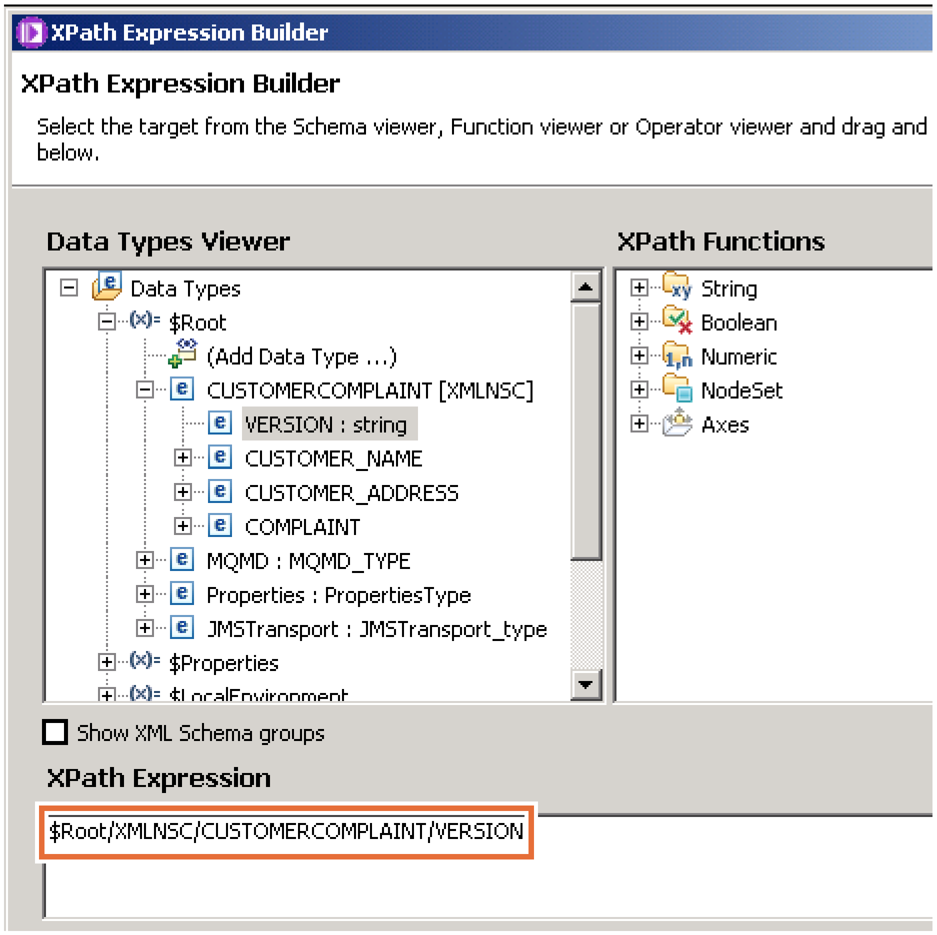
__ d. To the right of **Filter pattern**, click **Edit**. The XPath expression builder is shown.

__ e. In the **Data Types Viewer** pane, expand **$Root**, and then click **Add Data Type**. The **Type Selection Menu** is displayed.

__ f. Select **CUSTOMERCOMPLAINT**, and then click **OK**. CUSTOMERCOMPLAINT is added to the hierarchy.

__ g. Expand **CUSTOMERCOMPLAINT** under **$Root**.

__ h. Double-click **VERSION**. The hierarchical expression is copied to the XPath Expression section.



XPath Expression Builder

XPath Expression Builder

Select the target from the Schema viewer, Function viewer or Operator viewer and drag and below.

Data Types Viewer

- Data Types
  - $Root
    - (Add Data Type ...)
    - CUSTOMERCOMPLAINT [XMLNSC]
      - VERSION : string
      - CUSTOMER_NAME
      - CUSTOMER_ADDRESS
      - COMPLAINT
    - MQMD : MQMD_TYPE
    - Properties : PropertiesType
    - JMSTransport : JMSTransport_type
  - $Properties
  - $LocalEnvironment

XPath Functions

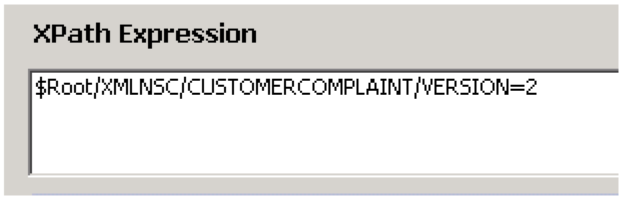
- String
- Boolean
- Numeric
- NodeSet
- Axes

☐ Show XML Schema groups

XPath Expression

```
$Root/XMLNSC/CUSTOMERCOMPLAINT/VERSION
```

__ i. Complete the XPath expression by typing =2 at the end of the string in the **XPath Expression** field.



XPath Expression

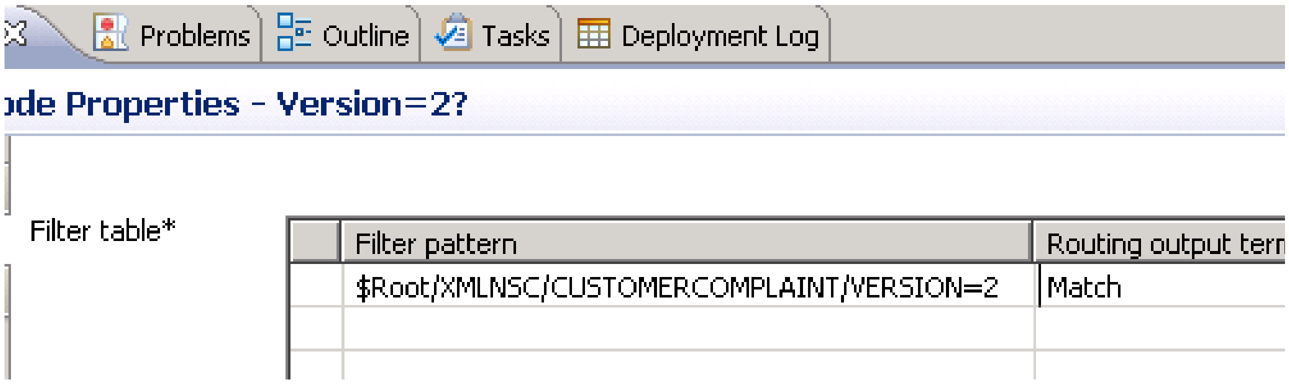
```
$Root/XMLNSC/CUSTOMERCOMPLAINT/VERSION=2
```

__ j. Click **Finish**. The **Add Filter table entry** menu is displayed.

__ k. Leave the **Routing output terminal** set to **Match**.

When a message passes through this Route node and the Version field equals 2, the message assembly is routed through the Match terminal of the node.

__ l. Click **OK** to add the row to the filter table.



Filter table*

	Filter pattern	Routing output term
	$Root/XMLNSC/CUSTOMERCOMPLAINT/VERSION=2	Match

__ 2. Save the message flow.

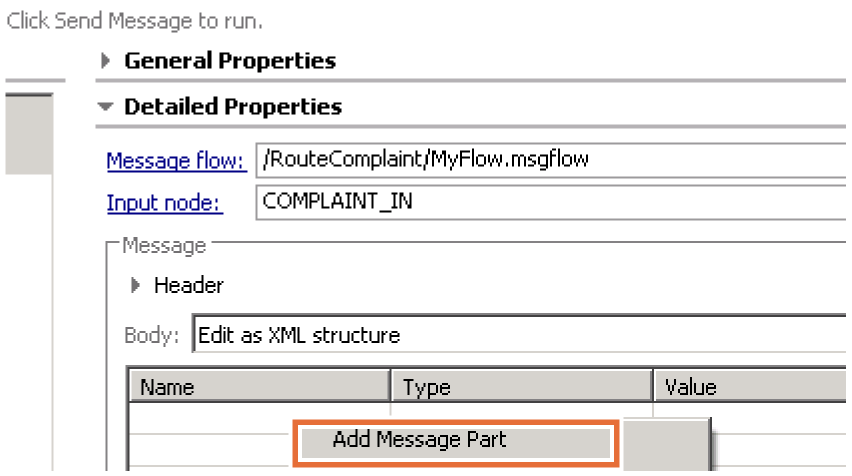
Part 3: Test with the Test Client

__ 1. Start the Test Client for the application.

In the Application Development view, right-click **RouteComplaint** and then select **Test**. The Test Client starts.

__ 2. The input node expects an XML message, and the message flow is associated with a message definition. Thus, the **Edit as XML structure** editor is available. Populate the CUSTOMERCOMPLAINT type with test data.

__ a. On the **Events** tab of the Test Client, right-click anywhere in the **Message** table (in the Name, Type, Value area) and then select **Add Message Part** from the menu.


Click Send Message to run.

General Properties

Detailed Properties

Message flow: /RouteComplaint/MyFlow.msgflow

Input node: COMPLAINT_IN

Message

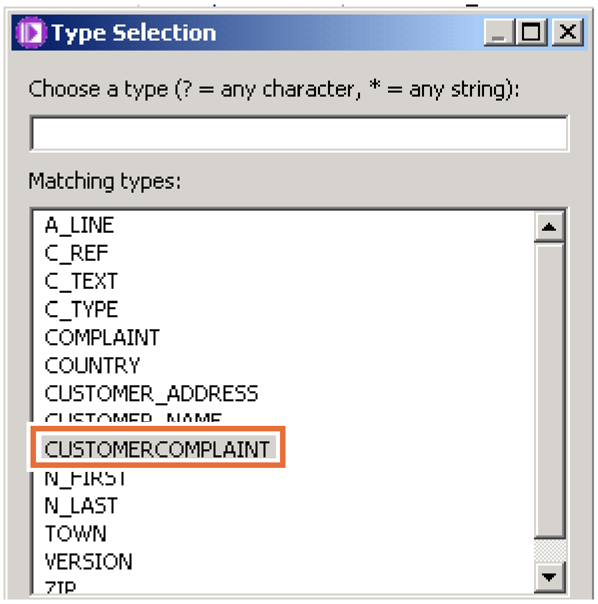
Header

Body: Edit as XML structure

Name	Type	Value

The **Type Selection** menu opens.

__ b. Select **CUSTOMERCOMPLAINT**, and then click **OK**.



The Message table is populated with the message definition that you imported by using the XSD file at the start of the exercise.

__ 3. The elements are filled with default values. You can use most of those values, but the message flow needs specific values for the **VERSION** (`2` or another value) and **C_TYPE** (`Order`, `Delivery`, `Other`) elements.

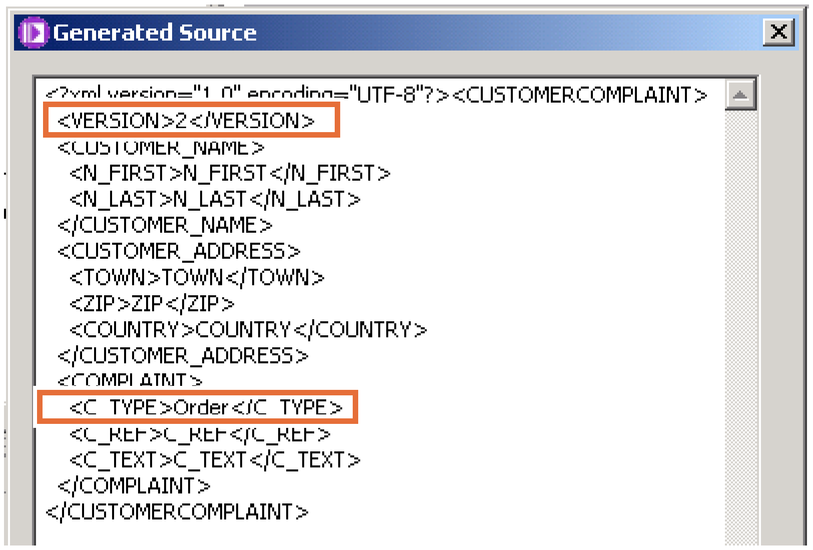
__ a. Right-click the **VERSION** element, and then select **Set Value** from the menu.

__ b. Enter `2` and click **OK**.

__ c. Right-click the **C_TYPE** element and select **Set Value**.

__ d. Type `Order` and then click **OK**.

___ e. Click **Show Generated Source** below the Message table, and verify that `Version=2` and `C_TYPE=Order` in the generated XML message.



```
<?xml version="1.0" encoding="UTF-8"?><CUSTOMERCOMPLAINT>
 <VERSION>2</VERSION>
 <CUSTOMER_NAME>
  <N_FIRST>N_FIRST</N_FIRST>
  <N_LAST>N_LAST</N_LAST>
 </CUSTOMER_NAME>
 <CUSTOMER_ADDRESS>
  <TOWN>TOWN</TOWN>
  <ZIP>ZIP</ZIP>
  <COUNTRY>COUNTRY</COUNTRY>
 </CUSTOMER_ADDRESS>
 <COMPLAINT>
  <C_TYPE>Order</C_TYPE>
  <C_REF>C_REF</C_REF>
  <C_TEXT>C_TEXT</C_TEXT>
 </COMPLAINT>
</CUSTOMERCOMPLAINT>
```

___ f. Click **OK**.

Optional

If you want to save this test data for later reuse, you can store it in the workspace data pool as follows:

___ 4. Save these values to the pool.

___ a. Right-click the top-level **CUSTOMERCOMPLAINT** element, and then select **Add Value to Pool** from the menu. The **Value Name** menu opens.

___ b. Enter any name for this test data, for example, `Route_Version2_Order`.

___ c. Click **OK**.

___ 5. Set a breakpoint on the message flow between the Route node and the Compute node.

___ a. Switch to the Message Flow editor.

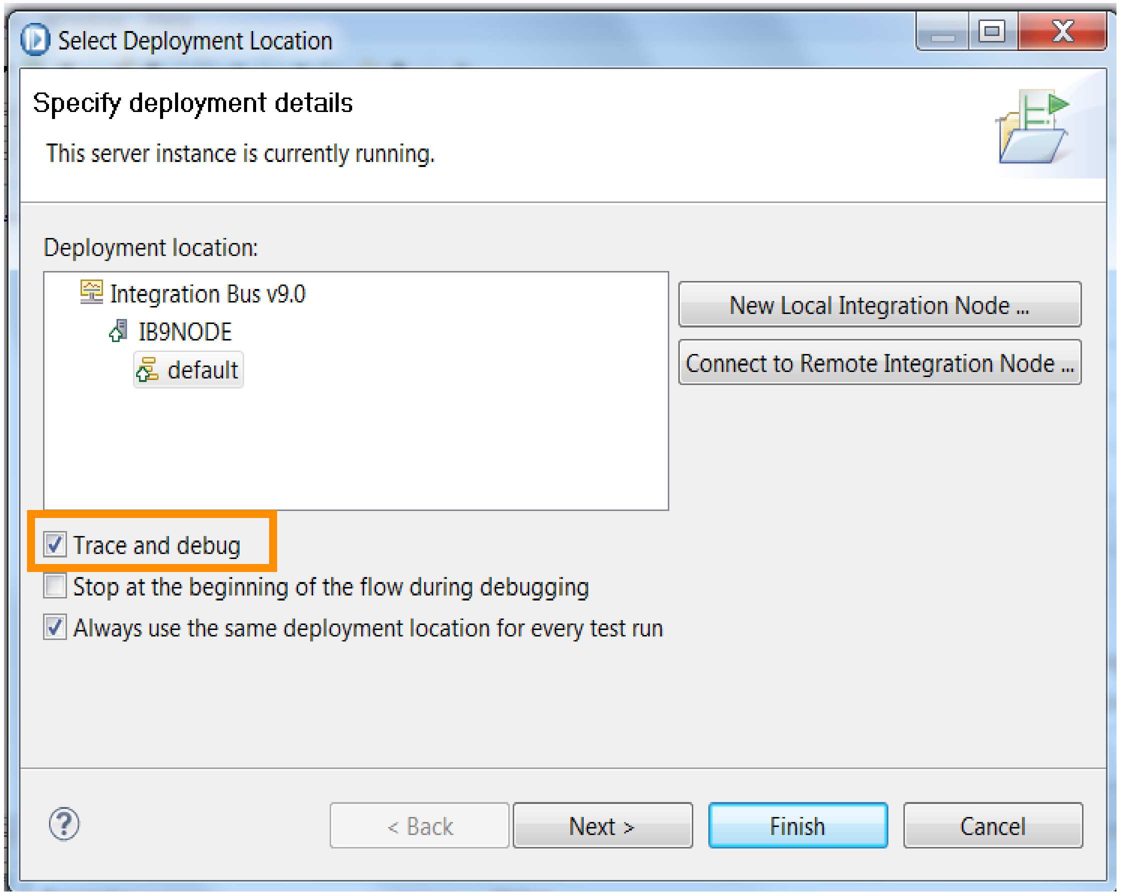
___ b. Right-click the wire between the Route node (**Version=2?**) and the Compute node (**Complaint ID and Department**), and then click **Add Breakpoint** from the menu. The breakpoint indicator is displayed on the wire.

___ 6. In the Test Client, send a test message with the tracing and debugging option enabled.

___ a. In the Test Client, click **Send Message**. The **Select Deployment Location** menu opens.

___ b. Select the **default** integration server.

___ c. Select the **Trace and debug** check box.



Select Deployment Location

Specify deployment details

This server instance is currently running.

Deployment location:

- Integration Bus v9.0
  - IB9NODE
    - default

☑ Trace and debug

☐ Stop at the beginning of the flow during debugging

☑ Always use the same deployment location for every test run

New Local Integration Node ...
Connect to Remote Integration Node ...
< Back
Next >
Finish
Cancel

___ d. Click **Finish**. The test starts.

Important

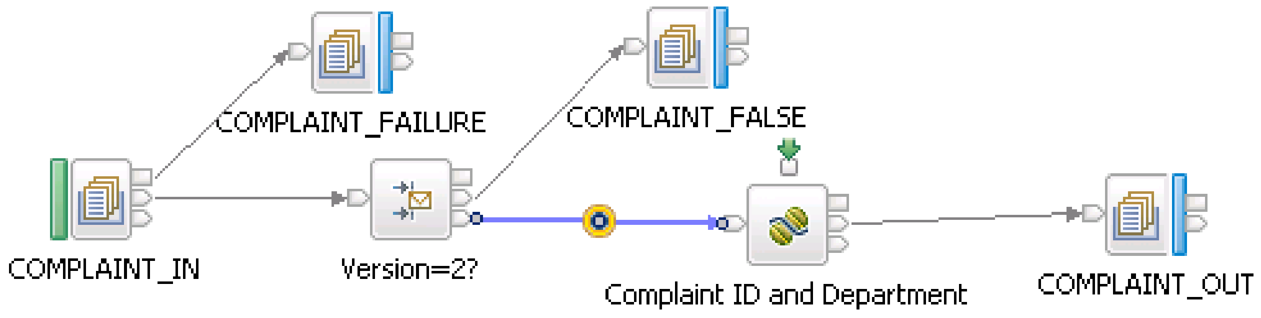
In Exercise 4, “Using Problem Determination Tools” you configured the debug port. If you did not complete Exercise 4 and set the debug port, deployment fails.

Open IBM Integration Explorer and verify that the debug port is set to 2311 on the **default** integration server and that the flow debug port is enabled. If it is not, configure the debug port and enable it.

___ e. If you did not specify a breakpoint, the flow completes and leaves you in the Integrated Test Client view.

If you did specify a breakpoint, you are prompted to switch to the Debug perspective. Click **Yes**.

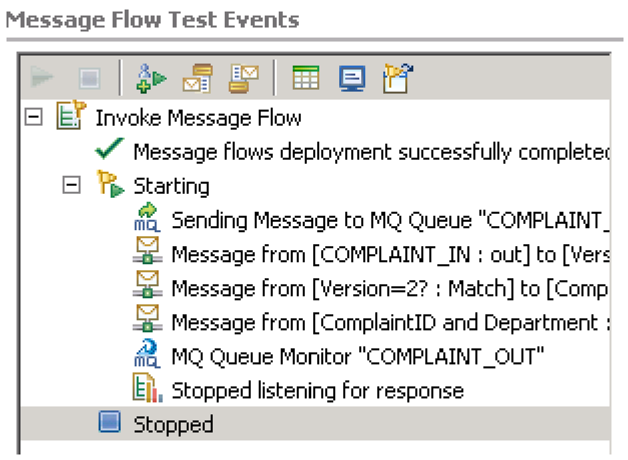
Verify that the message passed through the **Match** terminal of the Route node (**Version=2?**).



Click **Resume** (or press **F6**) to continue the flow. The message flow runs to completion.

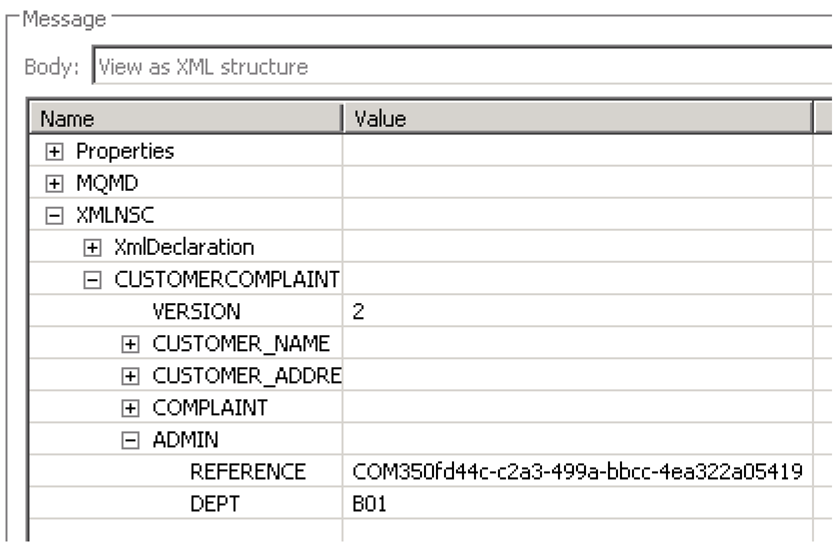
__ f. Switch back to the Integration Development perspective and the Test Client view.

__ g. Click the items in the Test Events list to inspect the flow of logical message and the output from queue monitors.



```
Invoke Message Flow
  Message flows deployment successfully completed
  Starting
    Sending Message to MQ Queue "COMPLAINT_
    Message from [COMPLAINT_IN : out] to [Vers
    Message from [Version=2? : Match] to [Comp
    Message from [ComplaintID and Department :
    MQ Queue Monitor "COMPLAINT_OUT"
    Stopped listening for response
  Stopped
```

For a message where `VERSION=2` and `C_TYPE=Order`, the message is routed to queue COMPLAINT_OUT with a new `ADMIN` structure at the end of the message, which includes a `REFERENCE` and `DEPT=B01`.


Message

Body: View as XML structure

Name	Value
⊞ Properties	
⊞ MQMD	
⊟ XMLNSC	
⊞ XmlDeclaration	
⊟ CUSTOMERCOMPLAINT	
VERSION	2
⊞ CUSTOMER_NAME	
⊞ CUSTOMER_ADDRE	
⊞ COMPLAINT	
⊟ ADMIN	
REFERENCE	COM350fd44c-c2a3-499a-bbcc-4ea322a05419
DEPT	B01

Your `REFERENCE` value is expected to be different because the `UUIDASCHAR` function generates a unique value.

Troubleshooting

If the test results are not what you expected, try to determine what went wrong. The Test Client Component Trace gives you valuable hints. Use other IBM Integration Bus debugging tools as appropriate.

If you cannot find a message on the COMPLAINT_OUT queue, try to find the reason by running the message flow with **UserTrace=Normal**. Retrieve user trace data by using the following script: `C:\Labs\Tools\GETTRACE.cmd`

You can also look for `BIPxxxx` messages in the Windows Event Viewer Application Log.

If the message is on the COMPLAINT_OUT queue but its content is not what you expect, run the message flow with UserTrace=Debug. Or, use the debugger to find out why the transformation was not correct.

__ 7. Test your flow with different values for `VERSION` and `C_TYPE` to find out whether the Route node configuration and the department distribution are correct.

Test files are supplied in `C:\Labs\Lab06-Route\data` as follows:

Test File	Effect
`Complaint_0.xml`	• No **VERSION** field • Message is routed unchanged to **COMPLAINT_FALSE**
`Complaint_1.xml`	• **VERSION** = 1 • Message is routed unchanged to **COMPLAINT_FALSE**
`Complaint_2d.xml`	• **VERSION** = 2, **C_TYPE** = Delivery • Message is routed to **COMPLAINT_OUT** with **DEPT** = C01
`Complaint_2m.xml`	• **VERSION** = 2, **C_TYPE** = Misc • Message is routed to **COMPLAINT_OUT** with **DEPT** = E01
`Complaint_2o.xml`	• **VERSION** = 2, **C_TYPE** = Order • Message is routed to **COMPLAINT_OUT** with **DEPT** = B01

Note

The next steps describe how to use the Test Client for testing in “Run” mode. You can optionally use RFHUtil to send test messages by using the test files.

__ a. In the Test Client, go to the **Configuration** tab.

__ b. Under **Deployment**, in the **Deployment location section**, click **Change**.

__ c. Clear the **Trace and debug** check box.

__ d. Click **Finish**.

__ e. Switch to the Test Client **Events** tab.

__ f. Right-click **Invoke Message Flow** on the Message Flow Events list and select **Duplicate**.

__ g. In Message Viewer**,** change the value of **C_TYPE** to `Delivery`.

Optional

If you want to reuse this test data, you can store it in the workspace data pool as follows:

__ h. Right-click the top-level CUSTOMERCOMPLAINT element and select **Add Value to Pool**.

__ i. Enter any name for this test data, for example: `LAB07_Version2_Delivery`

__ j. Click **OK**.

__ k. Click **Send Message**.

__ l. Inspect the outcome of this test. The message is routed to queue COMPLAINT_OUT with DEPT=C01.

__ m. Using the same procedure as describe in steps **f** through **k**, test the other combinations that are listed in the table.

Part 4: Clean up the environment

__ 1. Close all open editors and the Test Client. 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 **Terminate Debugger**.

__ 3. In the **Integration Nodes** view, right-click the **default** integration server and click **Delete > All Flows and Resources**.

End of exercise

Exercise 7. Implementing explicit error handling

What this exercise is about

In this exercise, you implement message processing nodes that allow you to control the paths that messages take in a message flow. You also write a general-purpose subflow to handle errors that occur during message processing.

What you should be able to do

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

- Implement a generic error handling routine in the form of a subflow
- Use a ResetContentDescriptor node to force the message to be reparsed according to the parser domain that is specified in the node properties
- Use the Debug perspective to examine the ExceptionList

Introduction

In a typical IBM Integration Bus environment, it is useful to have a standard way to handle runtime errors that occur. You learned that runtime errors can be handled locally (at the node where they occur) for many types of message processing nodes. However, in many instances an organization might choose to implement “standard” code that is used for displaying and reporting runtime errors.

In this exercise, you write a general-purpose error handler, in the form of a subflow. You might find this subflow useful in your own implementations.

Requirements

- A workstation with the IBM Integration Bus components installed and the default configuration
- The files are in `C:\Labs\Lab07-ErrorHandler` and `C:\Labs\Tools`
- The following local queues on IB9QMGR: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_FALSE, COMPLAINT_OUT, and ERROR_OUT

Exercise instructions

Part 1: Start components and import resources

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

__ 2. Make sure that the **IB9NODE** integration node and **default** integration server are running.

__ 3. If you successfully completed Exercise 6, “Adding flow control to the message flow,” proceed to **Part 2, Configure and complete ErrorHandler subflow**.

If you did not complete Exercise 6, start a new workspace and import one of the project interchange files in the `C:\Labs\Lab06-Route\Solution` directory.

There are two solution project interchange files:

- `RouteComplaint_WithCompute_PI.zip` contains a message flow with a Compute node and ESQL code.
- `RouteComplaint_WithJavaCompute_PI.zip` contains a message flow with a JavaCompute node and Java code.

Import only one of the project interchange files before you start this exercise.

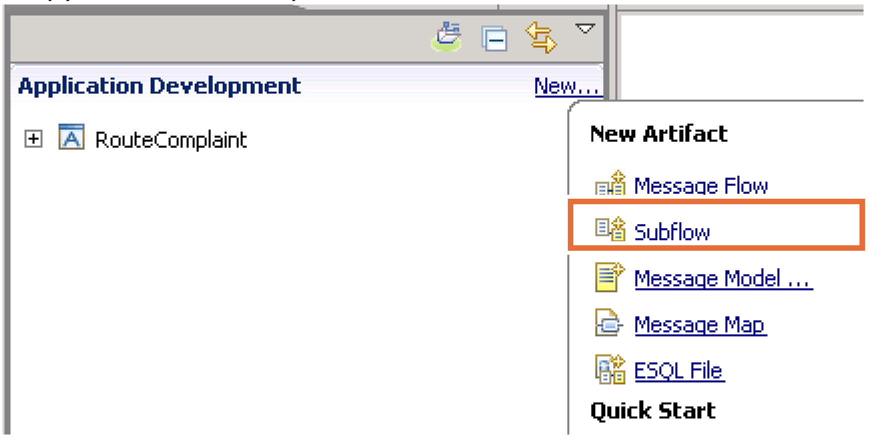
Part 2: Configure and complete ErrorHandler subflow

In this part of the exercise, you construct a subflow to be used as an error handler. It uses the ExceptionList tree, the Environment tree, integration node variables, and shared variables. It also uses pointer references.

Additionally, some of the code is taken directly from the information center. It is a good practice to use code such as this, as it was written by the developers and teams that support IBM Integration Bus.

__ 1. Create the message flow container.

__ a. In the Application Development view click **New > Subflow**.



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

__ c. For **Subflow name**, enter: `ErrorHandler_Subflow`

__ d. Click **Finish**. The message flow editor opens, and an Input and an Output node are already shown on the canvas.

In the next several steps, you add and configure the message processing nodes for the subflow.

__ 2. Rename the **Input** node to: `Catch`

__ 3. Add a ResetContentDescriptor node.

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

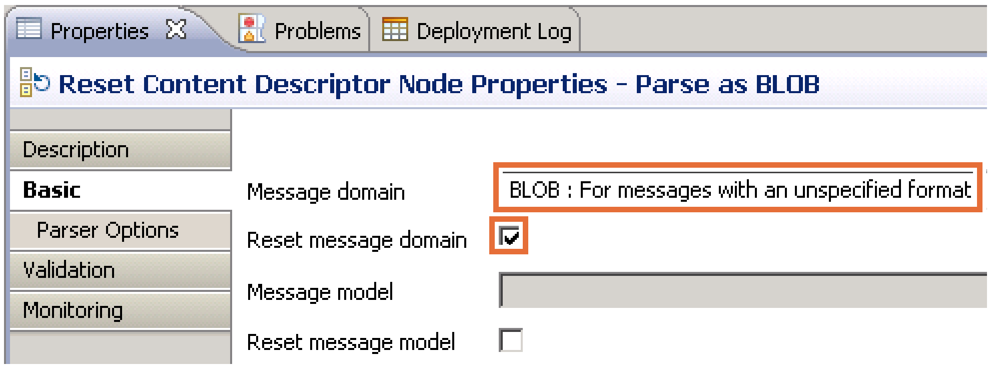
__ a. From the **Construction** drawer, add a **ResetContentDescriptor** node to the canvas.

__ b. Rename the **ResetContentDescriptor** node to: `Parse as BLOB`

__ c. Select the **Properties** view for the ResetContentDescriptor node.

__ d. On the **Basic** tab, for **Message domain**, select **BLOB**.

__ e. Select **Reset message domain**.



Reset Content Descriptor Node Properties - Parse as BLOB

Message domain: BLOB : For messages with an unspecified format

Reset message domain: ✓

Message model:

Reset message model: ☐

__ 4. Add a Compute node.

__ a. From the **Transformation** drawer, add a **Compute** node to the canvas.

__ b. Rename the **Compute** node to: `Capture Error`

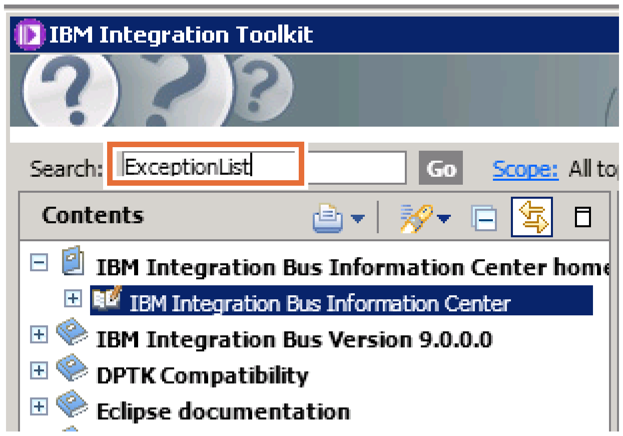
__ c. Select the **Properties** view for the Compute node.

__ d. On the **Basic** tab, rename **ESQL Module** to: `Capture_ExceptionList`

__ e. In the canvas, double-click the **Compute** node. The ESQL editor opens.

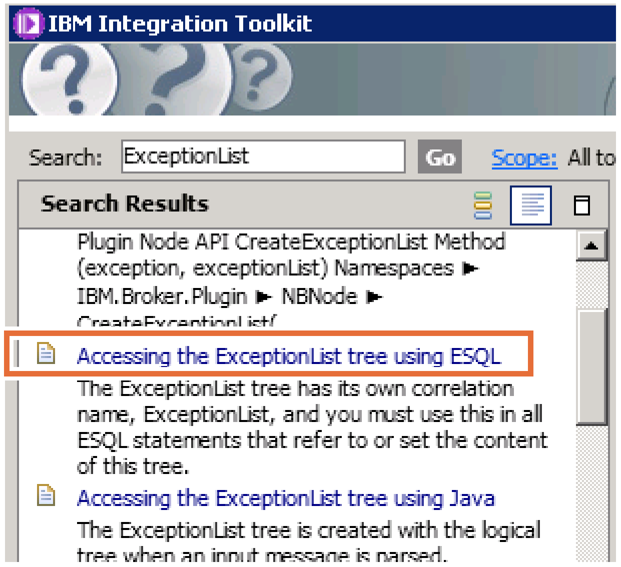
__ f. The ESQL code for the node is based partially on help information that is available in the information center. To access it, click **Help > Help Contents** from the IBM Integration Toolkit menu bar.

__ g. In **Search**, type `ExceptionList` and then click **Go**.


IBM Integration Toolkit
Search: ExceptionList
Go
Scope: All to
Contents
IBM Integration Bus Information Center home
IBM Integration Bus Information Center
IBM Integration Bus Version 9.0.0.0
DPTK Compatibility
Eclipse documentation

This search causes an index to be built for future searches, so this search can take several minutes. The search results are displayed after the search is done.

__ h. Click the link for the topic named **Accessing the ExceptionList tree using ESQL**.



Plugin Node API CreateExceptionList Method (exception, exceptionList) Namespaces ► IBM.Broker.Plugin ► NBNode ► CreateExceptionList(

Accessing the ExceptionList tree using ESQL
The ExceptionList tree has its own correlation name, ExceptionList, and you must use this in all ESQL statements that refer to or set the content of this tree.

Accessing the ExceptionList tree using Java
The ExceptionList tree is created with the logical tree when an input message is parsed.

The full contents of the help item are displayed.

Scroll through the help text until you find a **CREATE PROCEDURE** for **getLastExceptionDetail**. This procedure is one of the procedures you include in Compute node.

The complete ESQL for the module is in `C:\Labs\Lab07-ErrorHandler\resources\Cut&Paste.txt`.

__ i. Copy the entire contents of the `Cut&Paste.txt` file, and replace the entire existing contents of the ESQL module with it.

```
DECLARE s_Counter SHARED INT 0;

CREATE COMPUTE MODULE Capture_ExceptionList

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

  DECLARE ID CHAR;
  DECLARE mNumber integer;
  DECLARE mText char;

  CALL CopyMessageHeaders();

  SET OutputRoot.Properties.Transactional = false;

  SET ID = BrokerName || '/' ||
           ExecutionGroupLabel || '/' ||
           MessageFlowLabel;

  SET Environment.Variables.ID = ID;

  -- Temporarily store ExceptionList in OutputRoot.XMLNSC
  -- to profit from implicit casting of XML parser into CHAR
  -- which is required for RFH2.usr

  -- Pass the ExceptionList to return back the inner most child
  -- error number and Error Reason
  CALL getLastExceptionDetail(InputExceptionList, mNumber, mText);

  -- Track the number of messages that error and add this counter
  -- to fields tracked in RFH2 USR folder
  BEGIN ATOMIC
    SET s_Counter = s_Counter + 1;
    set OutputRoot.XMLNSC.ExceptionDump.ErrCounter = s_Counter;
    SET OutputRoot.XMLNSC.ExceptionDump.ErrList = InputExceptionList;
    set Environment.Variables.ErrCounter = s_Counter;
    Set Environment.Variables.errorNum = mNumber;
    Set Environment.Variables.errorReason = mText;
  END;


  SET OutputRoot.MQRFH2.usr=OutputRoot.XMLNSC;
  SET OutputRoot.MQRFH2.usr.ErrorHandler.ID = ID;
  SET OutputRoot.MQRFH2.usr.Counter = s_Counter;
  SET OutputRoot.MQRFH2.usr.ErrorNumber = mNumber;
  SET OutputRoot.MQRFH2.usr.ErrorReason = mText;

  -- Delete the temporary XML body
  DELETE FIELD OutputRoot.XMLNSC;

  -- copy original message body
  SET OutputRoot.BLOB = InputBody;
  RETURN TRUE;

END;  /* main */



  CREATE PROCEDURE getLastExceptionDetail(IN  InputTree reference,
                                OUT messageNumber integer,
                                        OUT messageText char)

/****************************************************************************
 * A procedure that will get the details of the last exception from a message
 * IN InputTree:  The incoming exception list
 * IN messageNumber:  The last message numberr.
 * IN messageText: The last message text.
 *****************************************************************************/
BEGIN

-- Create a reference to the first child of the exception list
declare ptrException reference to InputTree.*[1];

-- keep looping while the moves to the child of exception list work
WHILE lastmove(ptrException) DO

  -- store the current values for the error number and text
  IF ptrException.Number is not null THEN
            SET messageNumber = ptrException.Number;
            SET messageText = ptrException.Text;
  END IF;

  -- now move to the last child which should be the next exceptionlist
  move ptrException lastchild;

END WHILE;
END; /* getLastException */

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

END MODULE;
```

___ j. Save the ESQL by typing **Ctrl + S.**

__ k. Review the ESQL code.

This code is called only if a runtime error occurs. The code retrieves the information about the error from the ExceptionList, formats it, and places it in the `OutputRoot.MQRFH2.usr` folder. It also converts the original incoming message into a BLOB so that it can be sent to an WebSphere MQ output queue later in the flow.

__ l. Close the ESQL editor

__ 5. Add a TryCatch node.

From the **Construction** drawer, add a **TryCatch** node to the canvas. You do not need to rename it.

__ 6. Add an MQOutput node to be used as one of the destinations for the original message and the error information.

__ a. Rename the MQOutput node to: `ERROR_OUT`

__ b. On the **Basic** tab, for **Queue name**, enter: `ERROR_OUT`

__ 7. Add a FileOutput node.

__ a. From the **File** drawer, add a **FileOutput** node to the canvas.

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

__ c. In the **Basic** tab, for **Directory**, enter: `C:\errorout`

__ d. For **File name or pattern**, enter: `errorout.txt`

__ e. For **File action / mode for writing to file**, select **Write directly to the output file (append if file exists)**.

__ f. In the **Request** tab, for **Data location**, enter: `$Root`

__ 8. Delete the **Output** node.

__ 9. Wire the nodes.

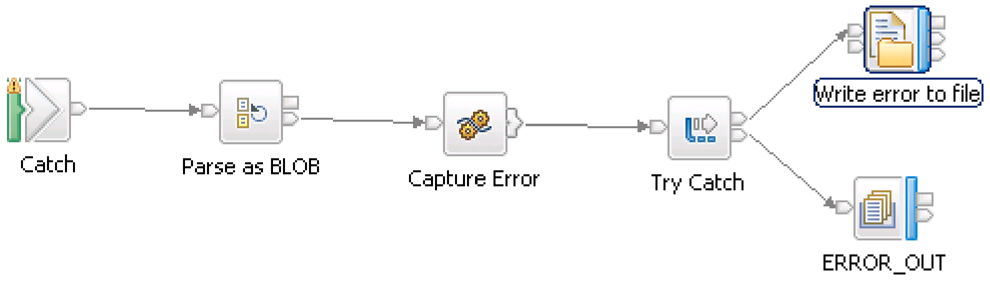
__ a. Wire the Input node (**Catch**) **Out** terminal to the ResetContentDescriptor node (**Parse as Blob**) **In** terminal.

__ b. Wire the ResetContentDescriptor node (**Parse as Blob**) **Out** terminal to the Compute node (**Capture Error**) **In** terminal.

__ c. Wire the Compute node (**Capture Error**) **Out** terminal to the **TryCatch** node **In** terminal.

__ d. Wire the **TryCatch** node **Try** terminal to the MQOutput node (**ERROR_OUT**) **In** terminal.

__ e. Wire the **TryCatch** node **Catch** terminal to the FileOutput node (**Write error to file**) **In** terminal.



__ 10. Save the message flow by typing **Ctrl + S**.

__ 11. Close the message flow editor for the subflow.

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

__ 12. Associate the subflow with the main flow.

__ a. If the main flow **Myflow.msgflow** (under **RouteComplaint > Flows**) is not open in the Message Flow editor, double-click it in the Application Development view.

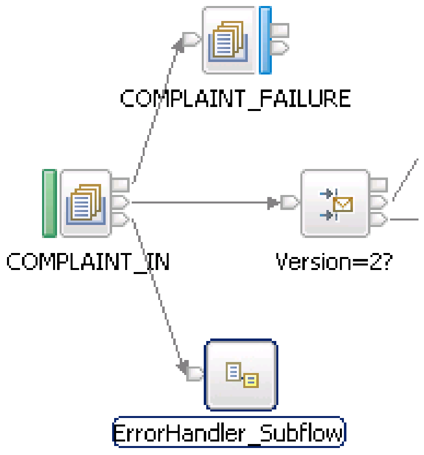
__ b. Right-click in the canvas, and select **Add Subflow** from the menu. The Add Subflow menu is displayed.

__ c. Expand **RouteComplaint > Subflows**, click **ErrorHandler_Subflow.subflow**, and then click **OK**.

The error handler subflow is shown on the desktop, represented by a single node.

__ d. Position the subflow node under and to the right of the MQInput node (**COMPLAINT_IN**).

__ e. Wire the MQInput node (**COMPLAINT_IN**) **Catch** terminal to the **Input** terminal of the **ErrorHandler** subflow.



__ 13. Save the workspace (**Ctrl + S**).

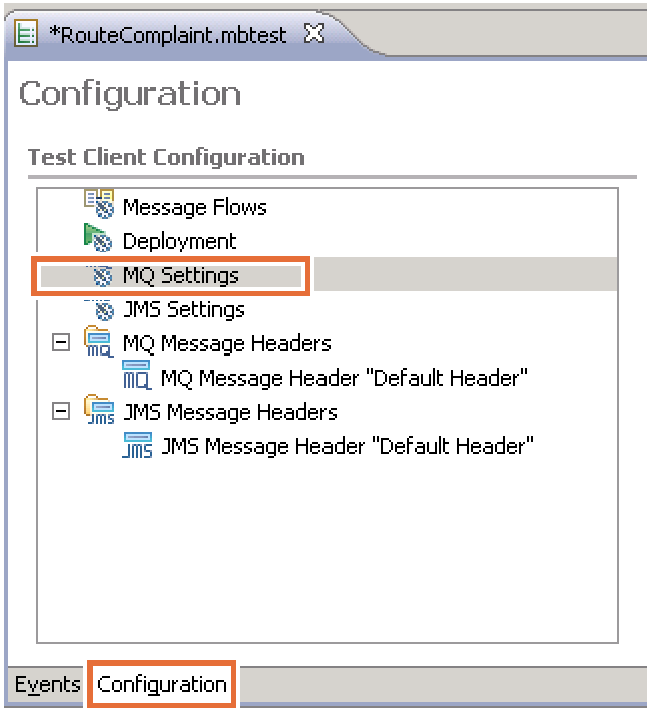
Part 3: Test the application

You use the Test Client, with the IBM Integration Explorer and RFHUtil, to test the exercise.

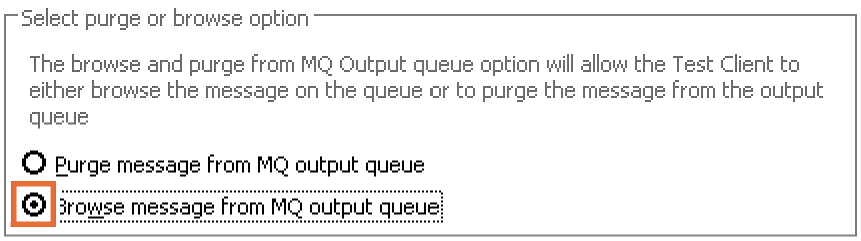
Test 1: Normal message flow processing

__ 1. In the Application Development view, right-click the **RouteComplaint** application and then select **Test** from menu to start the Test Client.

__ 2. On the Test Client **Configuration** tab, click **MQ Settings**.



__ 3. On the right side of the view, under **Select purge or browse option**, select **Browse message from MQ output queue**.


Select purge or browse option

The browse and purge from MQ Output queue option will allow the Test Client to either browse the message on the queue or to purge the message from the output queue

○ Purge message from MQ output queue
[✓ Browse message from MQ output queue]

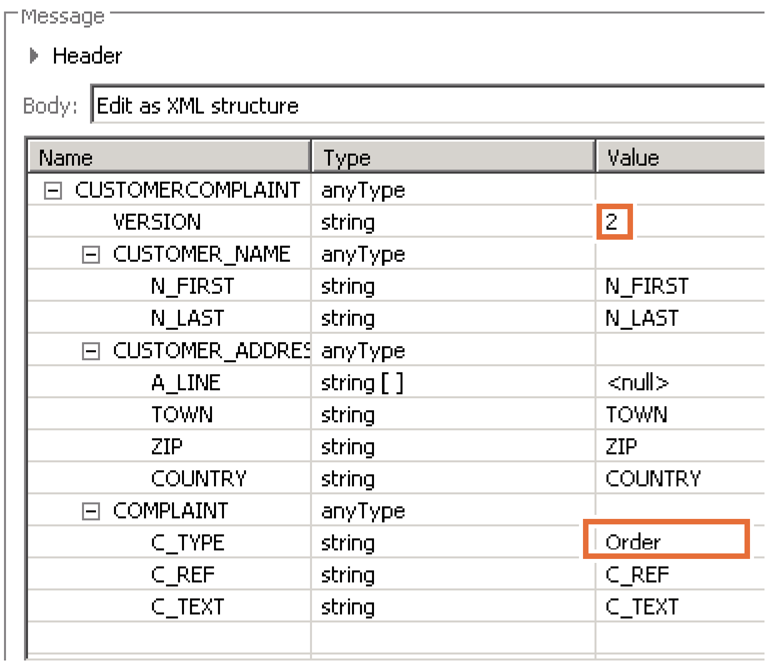
The default setting is to purge messages at the end of the Test Client run. If you use the default setting, you cannot view the messages on the output queues by using WebSphere MQ Explorer or RFHUtil because the Test Client purges the output queue.

__ 4. Enter the test data on the Test Client **Events** tab.

__ a. Under **Detailed Properties**, in the **Message** section, right-click in the **Name-Type-Value** grid, and then select **Add Message Part** from the menu. The Type Selection menu is displayed.

__ b. Select **CUSTOMERCOMPLAINT**, and then click **OK**. The message table is populated.

__ c. Change **VERSION** to `2`, and change **C_TYPE** (under **COMPLAINT**) to `Order`



Name	Type	Value
⊟ CUSTOMERCOMPLAINT	anyType	
VERSION	string	2
⊟ CUSTOMER_NAME	anyType	
N_FIRST	string	N_FIRST
N_LAST	string	N_LAST
⊟ CUSTOMER_ADDRES	anyType	
A_LINE	string [ ]	<null>
TOWN	string	TOWN
ZIP	string	ZIP
COUNTRY	string	COUNTRY
⊟ COMPLAINT	anyType	
C_TYPE	string	Order
C_REF	string	C_REF
C_TEXT	string	C_TEXT

__ 5. Send the test message.

__ a. Click **Send Message**. The Select Deployment Location menu is displayed.

__ b. Click **Finish**. The message flow is deployed and the test runs.

__ 6. Review the results. This test is similar to the one you did in the previous exercise. Because no runtime error occurred, the error handling subflow was not called and the message should be on the COMPLAINT_OUT queue.

__ 7. Repeat the test with other valid data, if you want, similar to the previous exercise. To repeat a test:

__ a. Right-click **Invoke Message Flow** in the **Message Flow Test Events** list, and then click **Duplicate**.

__ b. Change the test message as needed.

__ c. Click **Send Message** to submit the message.

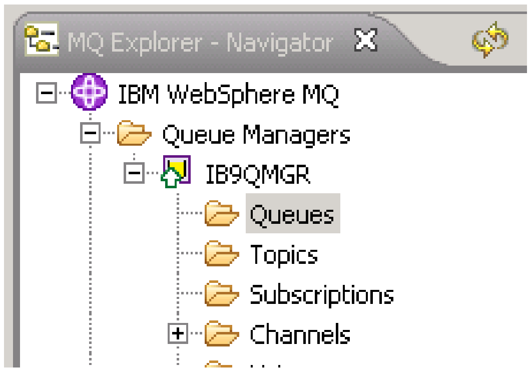
Test 2: Calling the error handler

In this part of the testing, you configure the test environment to force a runtime error to occur.

__ 1. To cause a runtime error, use the initial test data (VERSION = `2` and COMPLAINT.C_TYPE = `Order`), but before clicking Send Message, set the COMPLAINT_OUT queue to "put inhibited" status.

__ a. Start WebSphere MQ Explorer.

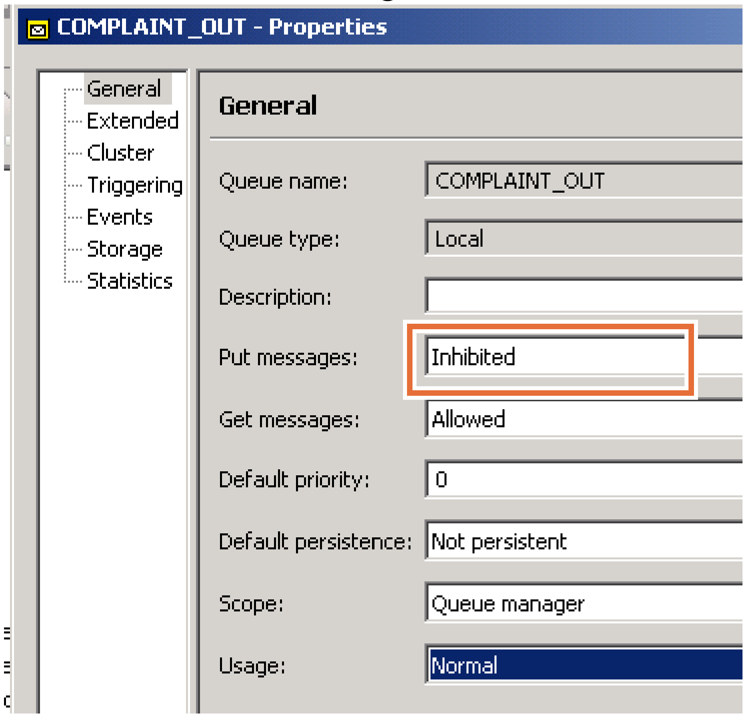
__ b. In the WebSphere MQ Explorer Navigator, expand **IBM WebSphere MQ > Queue Managers > IB9QMGR**.



__ c. Click **Queues**. The list of queues that are defined to the queue manager is shown in the right pane.

__ d. Right-click **COMPLAINT_OUT**, and then click **Properties** from the menu. The queue properties menu is displayed.

__ e. On the **General** tab, for **Put messages**, select **Inhibited** and then click **Apply**.



General

Queue name: COMPLAINT_OUT
Queue type: Local
Description:
Put messages: Inhibited
Get messages: Allowed
Default priority: 0
Default persistence: Not persistent
Scope: Queue manager
Usage: Normal

__ f. In the list of queues, for **COMPLAINT_OUT**, verify that the **Put messages** option is now set to **Inhibited**.

__ g. Click **OK** to close the queue properties window.

__ 2. Add a breakpoint to the message flows.

__ a. Open the **ErrorHandler_Subflow** subflow in the Message Flow editor, if it is not already open.

__ b. Double-click the Compute node (**Capture Error**) to open the ESQL editor.

__ c. Set a breakpoint on the **RETURN TRUE** statement near the end of the main procedure, by clicking in the left margin of the editor on that statement, and then selecting **Add Breakpoint** from the menu.

__ 3. In the **Integration Nodes** view, start the debugger for the **default** integration server.

__ 4. On the Test Client **Configuration** tab, click **Change** under the **Deployment location** section.

Select the **Trace and debug** option and then click **Finish**.

__ 5. Send a test message.

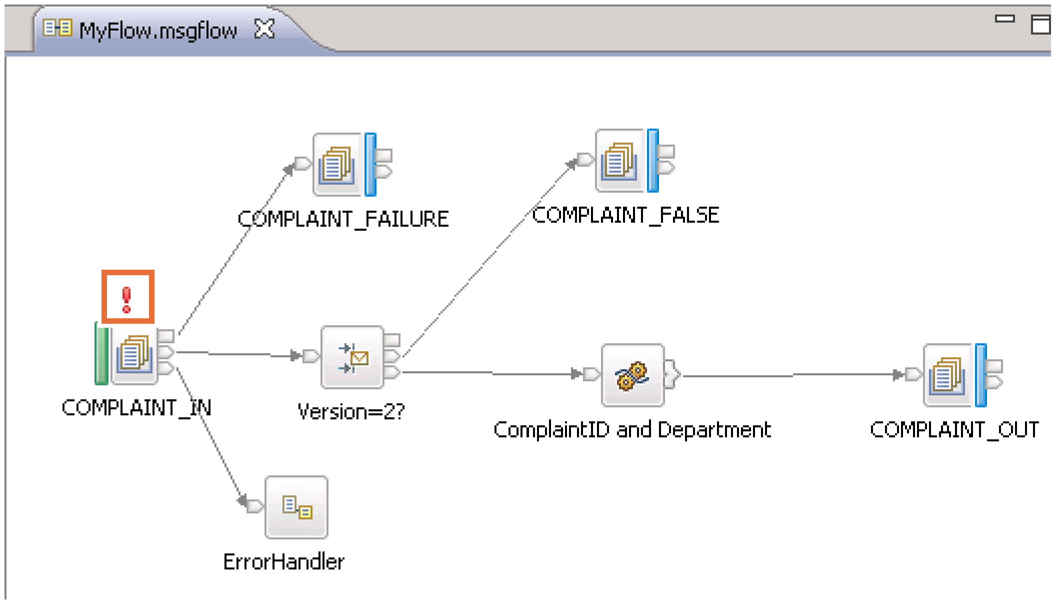
__ a. In the Test Client, again create a test message that attempts to write a record to the COMPLAINT_OUT queue (that is, be sure VERSION = 2).

__ b. Submit the test message. The message flow runs until it attempts to enqueue the message to COMPLAINT_OUT.

__ 6. Review the error handling that occurs.

__ a. When the runtime exception occurs, the debugger is started automatically. When you are asked to confirm the switch to the debugger perspective, click **Yes**.

__ b. In the Message Flow editor pane, observe the exclamation point over the MQInput node. This symbol indicates that a runtime exception occurred, and control is returned to the MQInput node.

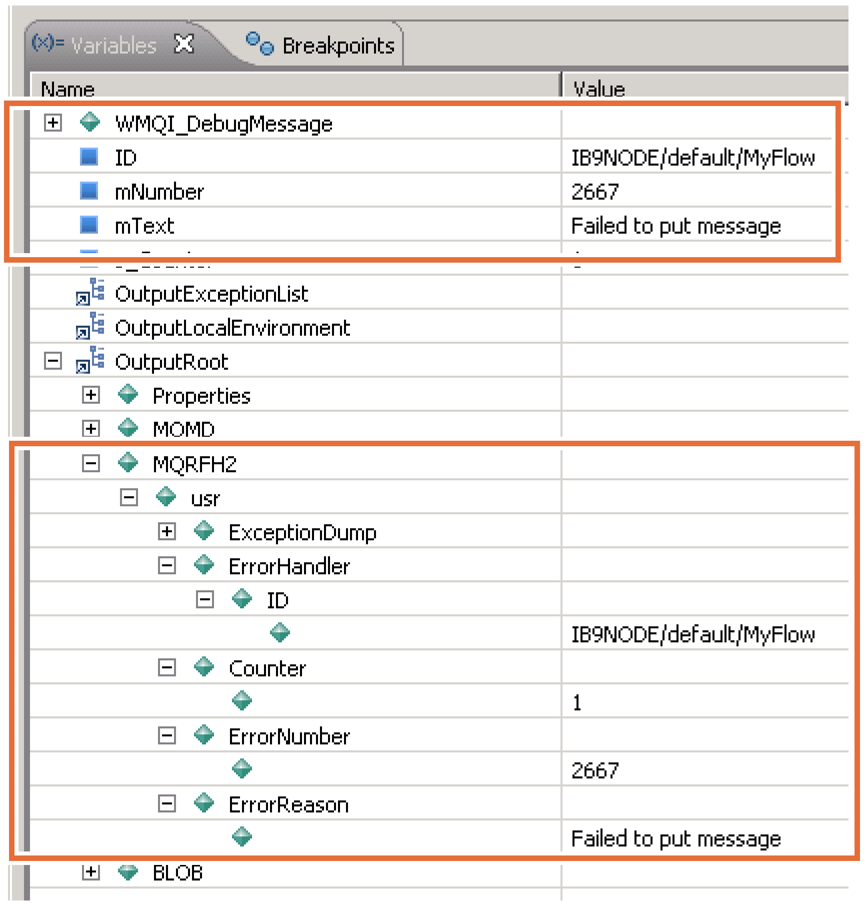


Questions

Why does control return to the MQInput node? See the material on error handling if you are not certain.

__ c. In the **Debug** view, click **Resume** (or press **F8**). Execution continues in the subflow, and then pauses when the breakpoint in the Compute node is encountered.

__ d. In the **Variables** view, examine the four variables that are declared in the Compute module ESQL. They are preceded by a blue square symbol on the **Variables** pane. Also, note the contents of the `OutputRoot.MQRFH2.usr` folder, corresponding to the SET statements in the ESQL.



```
Name                                Value
[+] WMQI_DebugMessage
    ID                              IB9NODE/default/MyFlow
    mNumber                         2667
    mText                           Failed to put message
    [illegible]
    OutputExceptionList
    OutputLocalEnvironment
[-] OutputRoot
    [+] Properties
    [+] MQMD
    [-] MQRFH2
        [-] usr
            [+] ExceptionDump
            [-] ErrorHandler
                [-] ID
                                    IB9NODE/default/MyFlow
            [-] Counter
                                    1
            [-] ErrorNumber
                                    2667
            [-] ErrorReason
                                    Failed to put message
    [+] BLOB
```

__ e. Click **Run to Completion**. The message flow finishes.

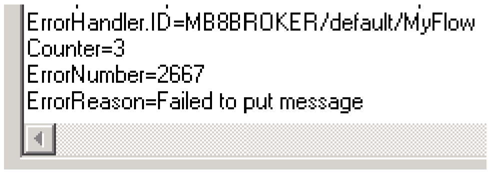
Questions

Where was the message sent, based on the flow within the error handler?

__ 7. Use WebSphere MQ Explorer or RFHUtil to verify the location and content of the message.

WebSphere MQ Explorer is useful to show all the queues and the queue depths (number of messages) on each. RFHUtil allows you to review the details of the messages. WebSphere MQ Explorer can show the basic content of messages as well, but does not give you the ability to look at headers or optional message assembly components, such as the `usr` folder.

Here are the partial contents of the `usr` folder as shown in RFHUtil for the ERROR_OUT queue.



```
ErrorHandler.ID=MB8BROKER/default/MyFlow
Counter=3
ErrorNumber=2667
ErrorReason=Failed to put message
```

Test 3: Calling the error handler with multiple runtime errors

In this part of the testing, you configure the test environment to force multiple runtime errors to occur. This action shows how the error handler responds to nested errors.

In the previous part of the exercise, the error handler was started because the message flow attempted to write to the COMPLAINT_OUT queue, but was unable to do so. In this part of the test, you cause a second runtime error to occur while the error handler is running.

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

__ a. In the WebSphere MQ Explorer navigator, again expand **IBM WebSphere MQ > Queue Managers > IB9QMGR**.

__ b. Click **Queues**. The list of queues that are defined to the queue manager is shown in the right pane.

__ c. Right-click **ERROR_OUT**, and then click **Properties** from the menu. The queue properties menu is displayed.

__ d. On the **General** tab, for **Put messages**, select **Inhibited**. Click **Apply** and then click **OK**.

__ e. In the list of queues, for both **ERROR_OUT** and **COMPLAINT_OUT**, the **Put messages** option is now **Inhibited**.

__ 2. Send another test message by using VERSION = `2` so that the message flow attempts to write to the COMPLAINT_OUT queue.

This time, when the TryCatch node in the error handling subflow attempts to write the error message to ERROR_OUT, it fails.

Use the debugging steps that were outlined previously to examine the output.

Questions

What happens in the error handling subflow?

In the error handling subflow, the Try path of the TryCatch node results in a runtime error because of the input-inhibited queue. The Catch path is then run, which starts the FileOutput node.

Where does the message go?

The message is written to the output file `C:\errorout\errorout.txt`.

Does the message flow continue to run afterward?

No, the flow does not continue processing because there are no further nodes to be processed in the message flow. When the catch terminal of the first MQInput node is wired, IBM Integration Bus assumes that the nodes that are wired to the Catch terminal are responsible for all error handling.

Part 4: Clean up the environment

__ 1. Close all open editors and the Test Client. 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 **Terminate Debugger**.

__ 3. In the **Integration Nodes** view, right-click the **default** integration server and select **Delete > All Flows and Resources**.

__ 4. In the WebSphere MQ Explorer, change the **Put messages** property on the COMPLAINT_OUT and ERROR_OUT queues to: `Allowed`

End of exercise

Exercise 8. Implementing a message model

What this exercise is about

In this exercise, you create a DFDL message model schema file that defines a reply message. You then modify an existing message flow to send the reply message to the originator of the input message.

What you should be able to do

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

- Create a DFDL message definition schema file
- Configure the logical and physical properties for the message model elements
- Test a DFDL schema by parsing test input
- Test a DFDL schema by serializing test output data
- Write ESQL or Java transformations to create a reply message
- Add an MQReply node to a message flow to send a response to the originator of the input message
- Configure the Test Client for WebSphere MQ headers and multiple output messages

Introduction

In this exercise, you extend the complaint flow from the previous exercise. You use a DFDL message definition schema file to define the format of the reply message and then modify the message flow to send the reply.

Requirements

- A workstation with the IBM Integration Toolkit components installed and the default configuration
- The files in `C:\Labs\Lab08-Modeling` and `C:\Labs\Tools`
- The following local queues on IB9QMGR: COMPLAINT_IN, COMPLAINT_FAILURE, COMPLAINT_FALSE, COMPLAINT_OUT, COMPLAINT_REPLY, and ERROR_OUT

Input message

The input message is the same as in the previous exercise. The message flow starts when the input message is put on the COMPLAINT_IN queue.

```
<CUSTOMERCOMPLAINT>
   <VERSION>2</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <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>
```

Sample output message

The sample output message includes the input message, followed by an administrative reference number and the responsible department. The message flow puts the output message on the COMPLAINT_OUT queue.

```
<CUSTOMERCOMPLAINT>
   <VERSION>2</VERSION>
   <CUSTOMER_NAME>
      <N_FIRST>Ed</N_FIRST>
      <N_LAST>Fletcher</N_LAST>
   </CUSTOMER_NAME>
   <CUSTOMER_ADDRESS>
      <A_LINE>Mail Point 135</A_LINE>
      <A_LINE>Hursley Park</A_LINE>
      <TOWN>Winchester</TOWN>
      <COUNTRY>UK</COUNTRY>
   </CUSTOMER_ADDRESS>
   <COMPLAINT>
      <C_TYPE>Delivery</C_TYPE>
      <C_REF>XYZ123ABC</C_REF>
      <C_TEXT>My order was delivered in time, but the package was torn and dirty.</C_TEXT>
   </COMPLAINT>
   <ADMIN>
      <REFERENCE>COMda1b71b2-f7fd-49e5-addb-ac9062f490c2</REFERENCE>
      <DEPT>B01</DEPT>
   </ADMIN>
</CUSTOMERCOMPLAINT>
```

Reply originator of input message

This message is sent to the originator of the input message. The same content is shown in two different formats: XML and plain text. This exercise creates the reply with the Text data.

XML:

```
<Complaint_Reply>
   <YourComplaint>
      <Type>Delivery</Type>
      <Reference>XYZ123ABC</Reference>
      <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>
```

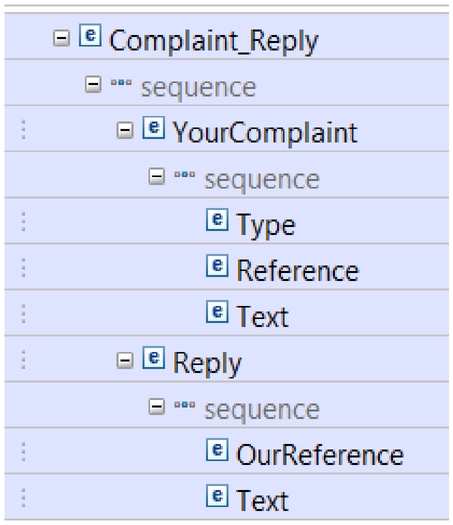
Text

This file consists of two records that are delimited by a “|”. In each record, the string “+++” delimits each field in the record.

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

Structure of message Complaint_Reply

This figure shows the structure of the Complaint_Reply message.



```
Complaint_Reply
  sequence
    YourComplaint
      sequence
        Type
        Reference
        Text
    Reply
      sequence
        OurReference
        Text
```

Exercise instructions

Part 1: Start components and import resources

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

__ 2. Make sure that all local IBM Integration Bus components are running.

__ 3. You can either continue to use the existing workspace that you created for the previous exercise, or you can create a workspace and import the solution file from Exercise 7.

If you successfully completed Exercise 7, “Implementing explicit error handling,” proceed to **Part 2, Model message and set properties for physical formats**.

If you did not complete Exercise 7, start a new workspace and import the `RouteComplaint_ WithErrorHandlerSubflow_PI.zip` project interchange file from the `C:\Labs\Lab07-ErrorHandler\Solution` directory.

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

In this part of the exercise, you create the message model that defines the reply message that matches the text file format that is described in the exercise introduction. You also parse test input data and serialize test output data to ensure that the model accurately defines the data.

Before starting this part of the exercise, review the file structure and contents to ensure that you understand the structure of the data. A sample data file that is named `SampleComplaintReply.txt` is provided in the `C:\Labs\Lab08-Modeling\data` directory.

__ 1. Create a Library to store the DFDL message model for the Complaint_Reply message.

__ a. Select **File > New > Library**.

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

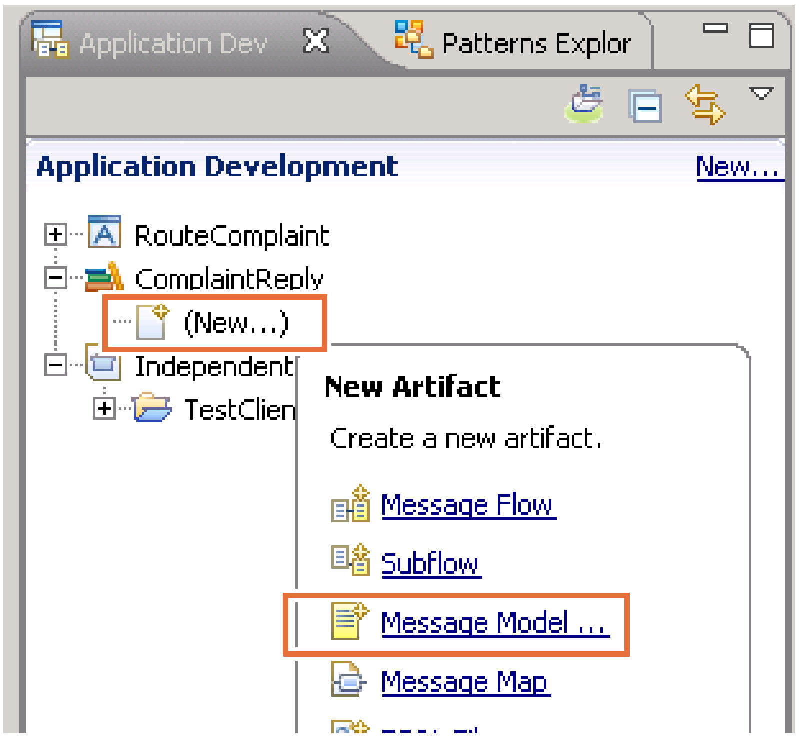
__ c. Click **Finish**.

__ 2. Use the **New Message Model** wizard to create a DFDL schema definition file for the Complaint_Reply message.

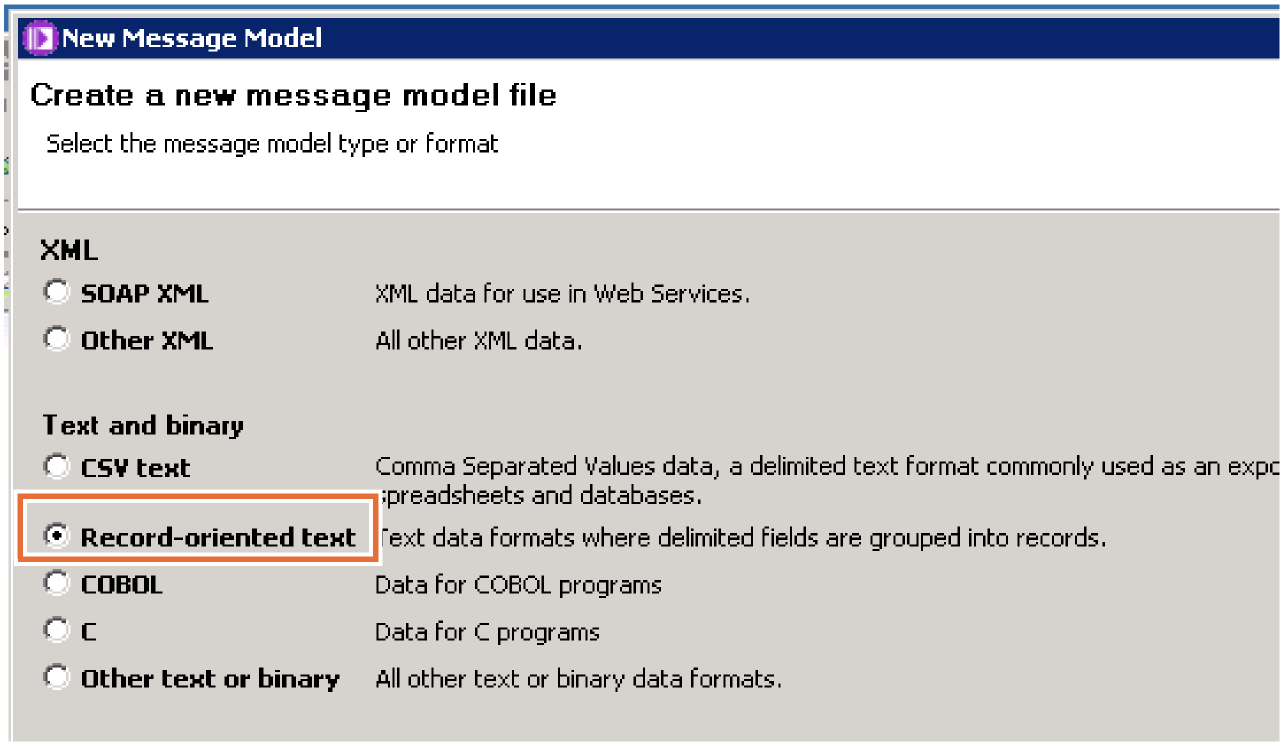
This Complaint_Reply message is a text message that consists of two records that are delimited by a "|". The fields in each record are delimited by "+++".

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

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



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



Create a new message model file

Select the message model type or format

XML

- ○ SOAP XML — XML data for use in Web Services.
- ○ Other XML — All other XML data.

Text and binary

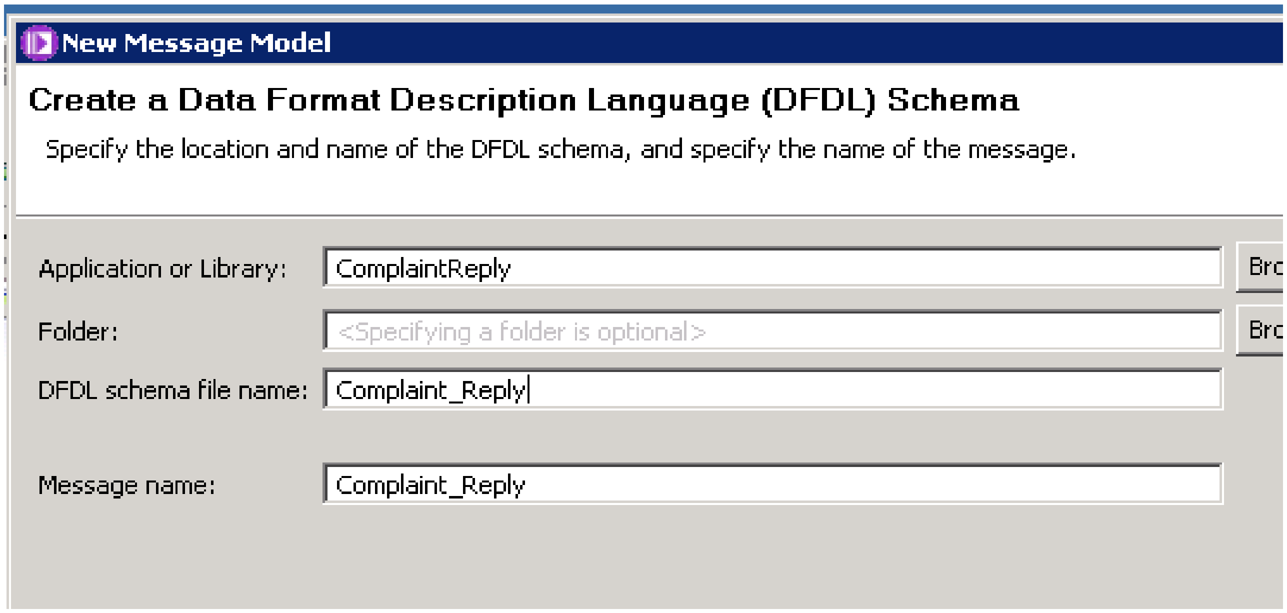
- ○ CSV text — Comma Separated Values data, a delimited text format commonly used as an expo spreadsheets and databases.
- ● Record-oriented text — Text data formats where delimited fields are grouped into records.
- ○ COBOL — Data for COBOL programs
- ○ C — Data for C programs
- ○ Other text or binary — All other text or binary data formats.

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

__ d. On the **Data Format Description Language (DFDL) Schema** page of the wizard, enter `Complaint_Reply` for the **DFDL schema file name**; the **Message name** field automatically uses the same name.

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

Click **Next**.



New Message Model

Create a Data Format Description Language (DFDL) Schema

Specify the location and name of the DFDL schema, and specify the name of the message.

Application or Library: ComplaintReply

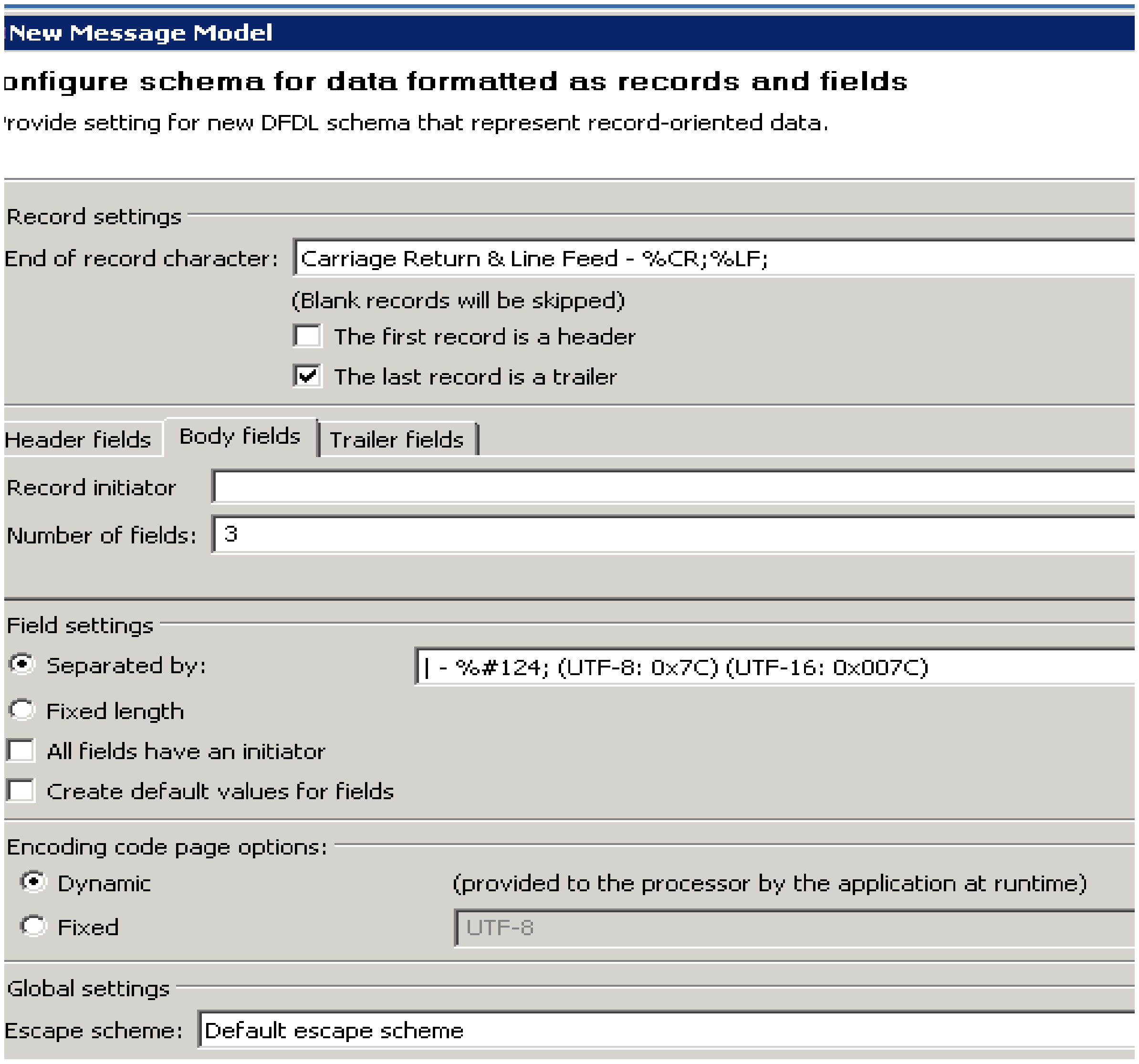
Folder: <Specifying a folder is optional>

DFDL schema file name: Complaint_Reply

Message name: Complaint_Reply

__ e. The Complaint_Reply message has two records that are delimited with a "|" character. On the **Configure schema for data formatted as records and field** page:

- Clear **The first record is a header** under **Record settings**.
- On the **Body fields** tab, clear the **Record initiator** field and set **Number of fields** to: 3
- On the **Trailer fields** tab, clear the **Record initiator** field and set **Number of fields** to: 2
- Under **Field settings**, select **| - %#124; (UTF-8: 0x7c)(UTF-16: 0x007C)** for the **Separated by** field and clear the **All fields have an initiator** option.


New Message Model

onfigure schema for data formatted as records and fields

'rovide setting for new DFDL schema that represent record-oriented data.

Record settings

End of record character: Carriage Return & Line Feed - %CR;%LF;

(Blank records will be skipped)

☐ The first record is a header

☑ The last record is a trailer

Header fields | Body fields | Trailer fields

Record initiator:

Number of fields: 3

Field settings

● Separated by: | - %#124; (UTF-8: 0x7C) (UTF-16: 0x007C)

○ Fixed length

☐ All fields have an initiator

☐ Create default values for fields

Encoding code page options:

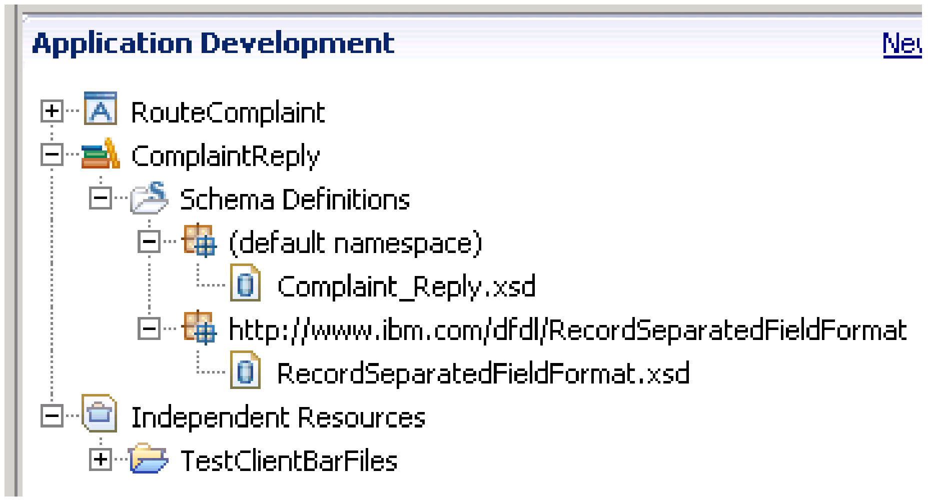
● Dynamic (provided to the processor by the application at runtime)

○ Fixed UTF-8

Global settings

Escape scheme: Default escape scheme

__ f. Click **Finish**.
The wizard creates a DFDL schema definition file that is named `Complaint_Reply.xsd` and opens the DFDL schema editor.
The wizard also includes the “Helper” schema `RecordSeparatedFieldFormat.xsd` in the library.

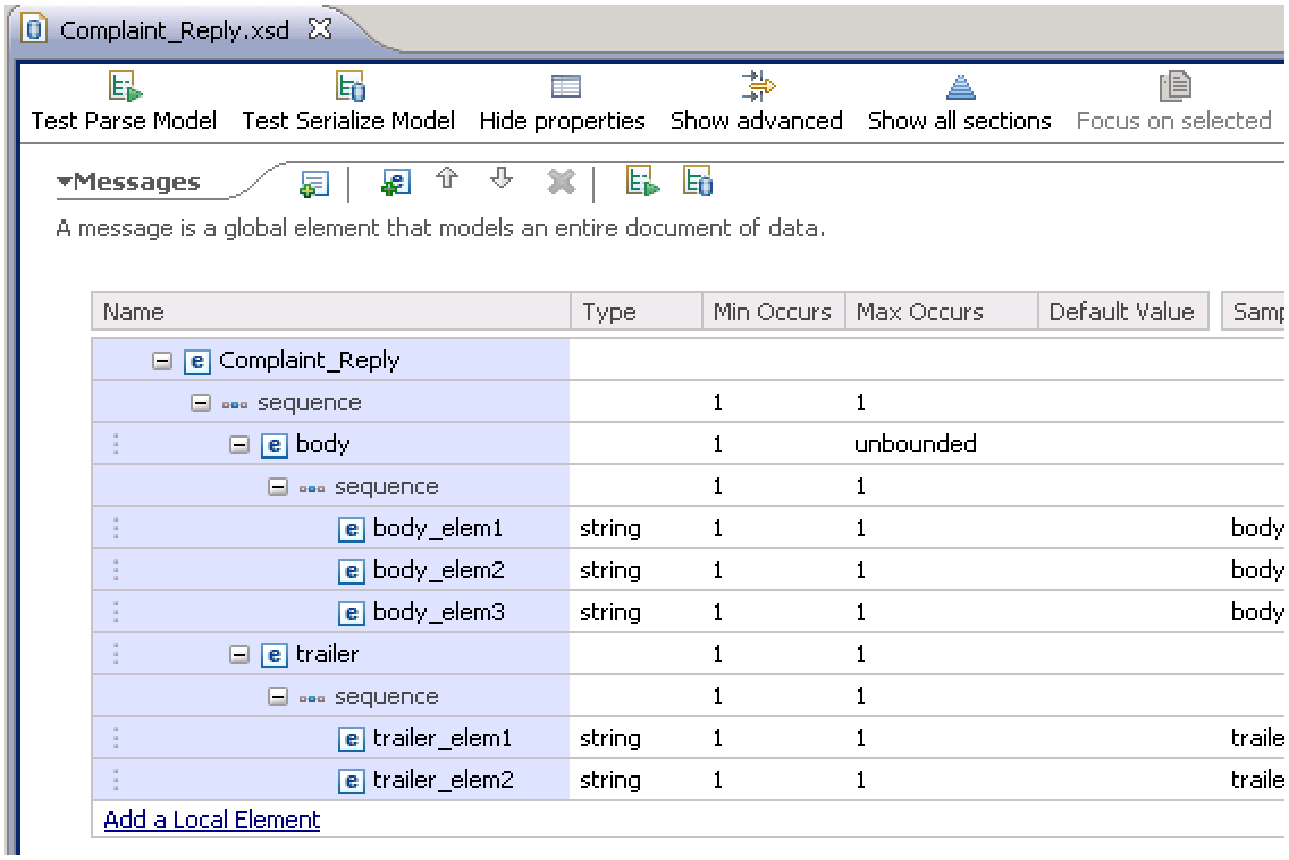

Application Development

```
RouteComplaint
ComplaintReply
    Schema Definitions
        (default namespace)
            Complaint_Reply.xsd
        http://www.ibm.com/dfdl/RecordSeparatedFieldFormat
            RecordSeparatedFieldFormat.xsd
Independent Resources
    TestClientBarFiles
```

__ 3. The DFDL schema editor opens with two panes. Expand the message elements so that you can see the structure and content of the Complaint_Reply message.

Reminder

To make it easier to see the entire view, double-click the **Complaint_Reply.xsd** tab to expand the view. When you are done, double-click the tab again to return the view to the default size.



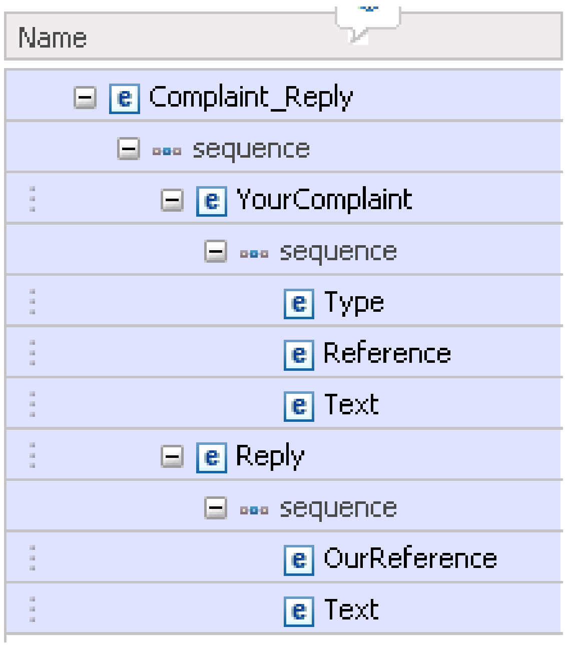
A message is a global element that models an entire document of data.

Name	Type	Min Occurs	Max Occurs	Default Value	Samp
Complaint_Reply					
sequence		1	1		
body		1	unbounded		
sequence		1	1		
body_elem1	string	1	1		body
body_elem2	string	1	1		body
body_elem3	string	1	1		body
trailer		1	1		
sequence		1	1		
trailer_elem1	string	1	1		traile
trailer_elem2	string	1	1		traile

Add a Local Element

__ 4. Rename the message elements to use more descriptive names by selecting the element name and then typing a new name.

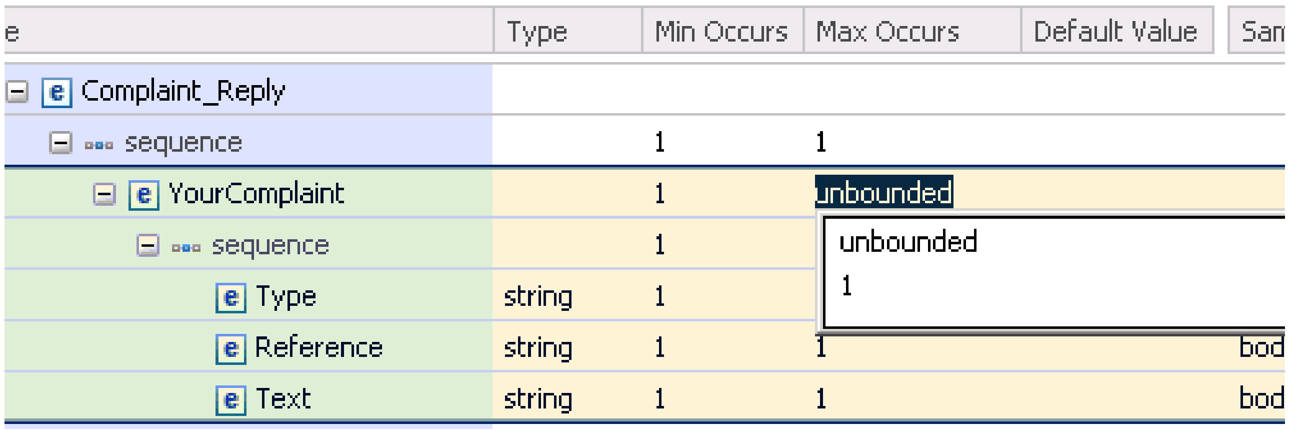
- Rename **body** to: `YourComplaint`
- Rename **body_elem1** to: `Type`
- Rename **body_elem2** to: `Reference`
- Rename **body_elem3** to: `Text`
- Rename **trailer** to: `Reply`
- Rename **trailer_elem1** to: `OurReference`
- Rename **trailer_elem2** to: `Text`



```
Name
Complaint_Reply
  sequence
    YourComplaint
      sequence
        Type
        Reference
        Text
    Reply
      sequence
        OurReference
        Text
```

__ 5. The Complaint_Reply message contains only one reply.

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



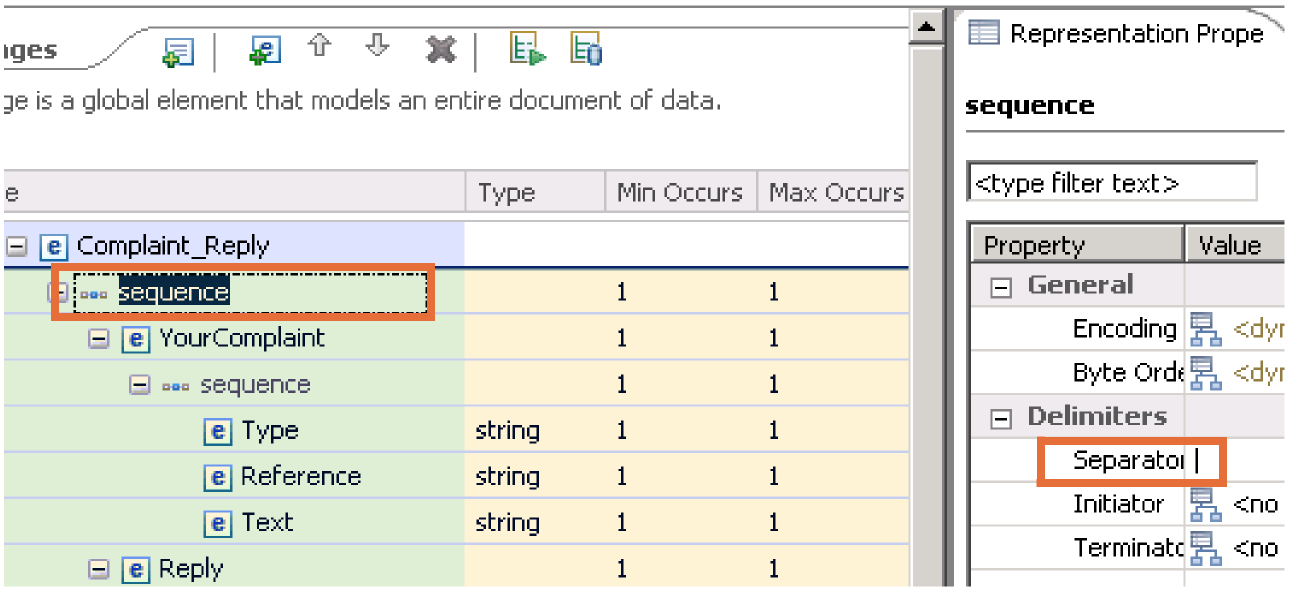
e	Type	Min Occurs	Max Occurs	Default Value	San
Complaint_Reply					
sequence		1	1		
YourComplaint		1	unbounded		
sequence		1			
Type	string	1			
Reference	string	1	1		bod
Text	string	1	1		bod

unbounded
1

__ 6. The Complaint_Reply uses a “|” character to distinguish the first three fields (**YourComplaint**) from the fourth and fifth fields (**Reply**).

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

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



ges

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

e	Type	Min Occurs	Max Occurs
Complaint_Reply			
sequence		1	1
YourComplaint		1	1
sequence		1	1
Type	string	1	1
Reference	string	1	1
Text	string	1	1
Reply		1	1

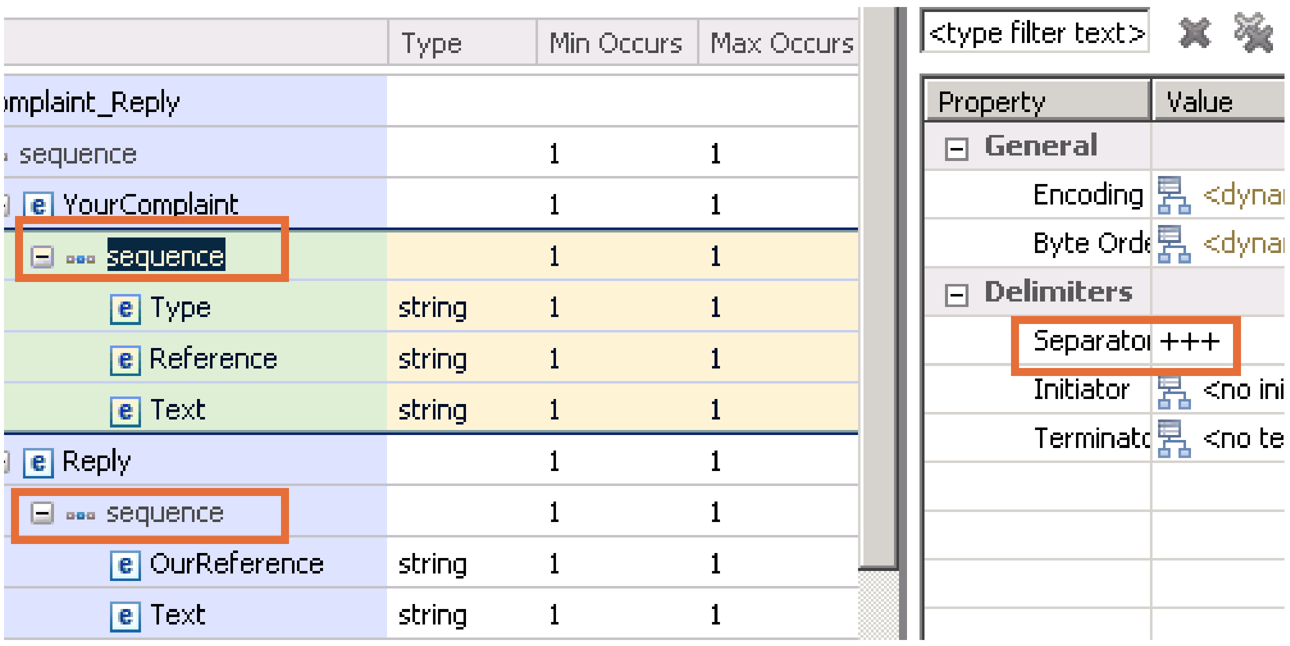
Representation Prope

sequence

<type filter text>

Property	Value
General	
Encoding	<dyr
Byte Orde	<dyr
Delimiters	
Separatoı	|
Initiator	<no
Terminatc	<no

__ 7. The fields in **YourComplaint** and **Reply** are separated by "+++". Change the **Separator** value for the **sequence** elements under **YourComplaint** and **Reply** to: +++



	Type	Min Occurs	Max Occurs
mplaint_Reply			
sequence		1	1
YourComplaint		1	1
sequence		1	1
Type	string	1	1
Reference	string	1	1
Text	string	1	1
Reply		1	1
sequence		1	1
OurReference	string	1	1
Text	string	1	1

<type filter text>

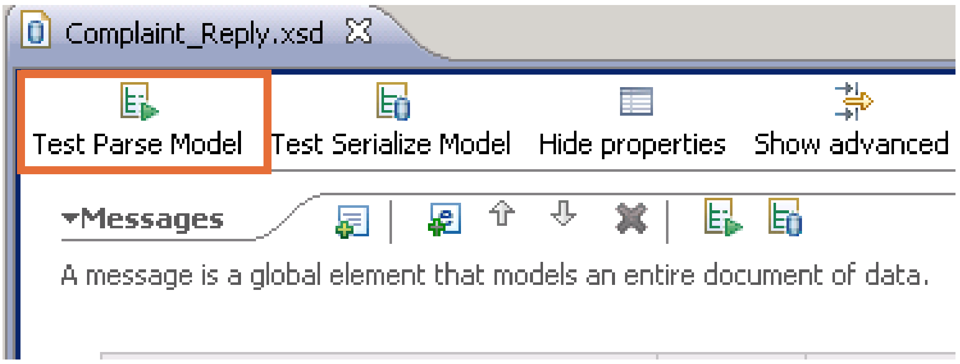
Property	Value
General	
Encoding	<dynai
Byte Orde	<dynai
Delimiters	
Separatoı	+++
Initiator	<no ini
Terminatc	<no te

8. Save the `Complaint_Reply.xsd` DFDL schema.

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

__ 9. To ensure that the DFDL model accurately describes the data, run the **Test Parse Model** option with a sample file.

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



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

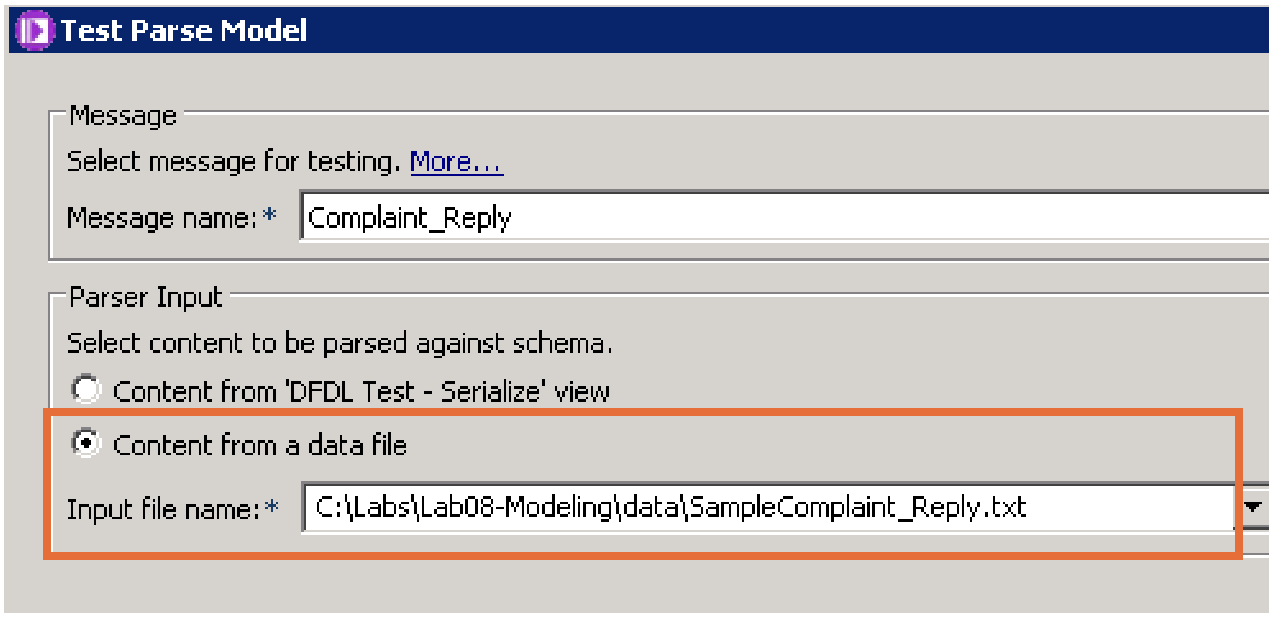
__ b. On the Test Parse Model window, select **Content from data file**.

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

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

__ e. Browse to the `C:\Labs\Lab08-Modeling\data` directory, select **SampleComplaint_Reply.txt**, and then click **Open**.

__ f. Click **OK** on **File Selection** window.



Select message for testing. More...

Message name:* Complaint_Reply

Select content to be parsed against schema.

Content from 'DFDL Test - Serialize' view

Content from a data file

Input file name:* C:\Labs\Lab08-Modeling\data\SampleComplaint_Reply.txt

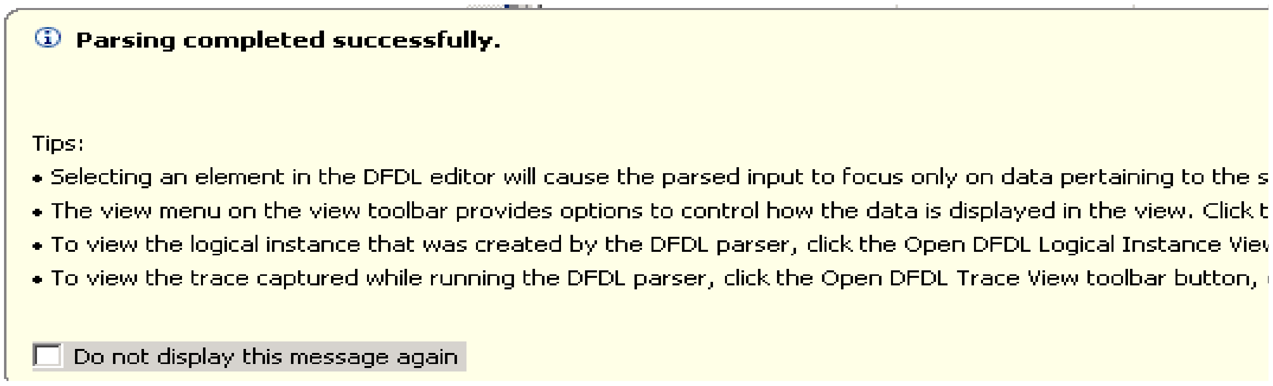
__ g. Click **OK** on the Test Parse Model window.

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

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

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

If parsing fails, an error message that describes the cause of the failure is displayed. Review the error message, modify the properties, save the model, and rerun the test until the test parse succeeds.



Parsing completed successfully.

Tips:

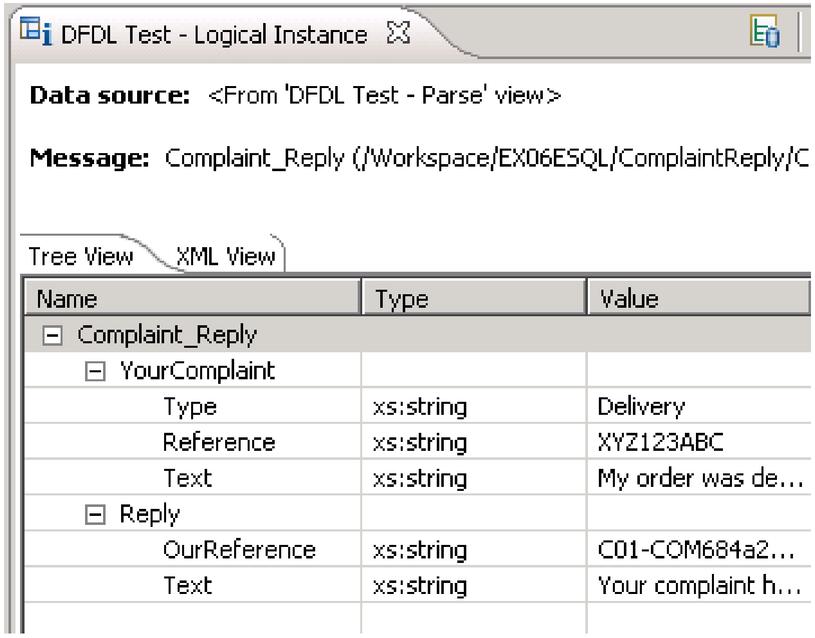
- Selecting an element in the DFDL editor will cause the parsed input to focus only on data pertaining to the s
- The view menu on the view toolbar provides options to control how the data is displayed in the view. Click t
- To view the logical instance that was created by the DFDL parser, click the Open DFDL Logical Instance View
- To view the trace captured while running the DFDL parser, click the Open DFDL Trace View toolbar button,

☐ Do not display this message again

__ j. Close the “Parsing completed successfully” window.

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

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



DFDL Test - Logical Instance

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

Message: Complaint_Reply (/Workspace/EX06ESQL/ComplaintReply/C

Tree View | XML View

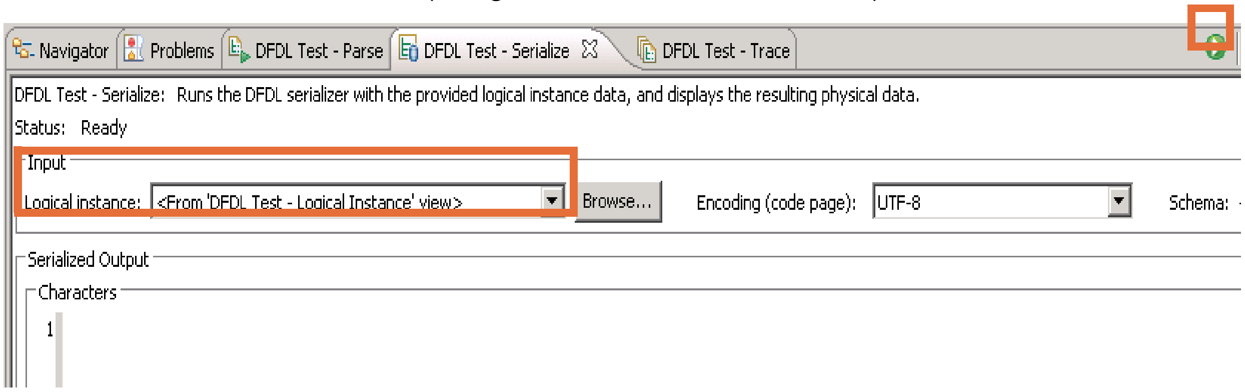
Name	Type	Value
⊟ Complaint_Reply		
⊟ YourComplaint		
Type	xs:string	Delivery
Reference	xs:string	XYZ123ABC
Text	xs:string	My order was de...
⊟ Reply		
OurReference	xs:string	C01-COM684a2...
Text	xs:string	Your complaint h...

__ 10. The message flow in this exercise is going to generate the Complaint_Reply from a logical model on output. Run the **Test Serialize Model** option against the logical instance to ensure that the model builds the output data correctly.

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

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

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



DFDL Test - Serialize: Runs the DFDL serializer with the provided logical instance data, and displays the resulting physical data.
Status: Ready
Logical instance: <From 'DFDL Test - Logical Instance' view>
Encoding (code page): UTF-8
Schema:

__ d. If the DFDL Serializer can successfully generate the output from the model, the message “`Serialization completed successfully`” is displayed. Close this window.

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

Information

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

__ 11. Return to the Integration Development perspective.

__ 12. Close the DFDL schema editor.

Part 3: Extend the message flow

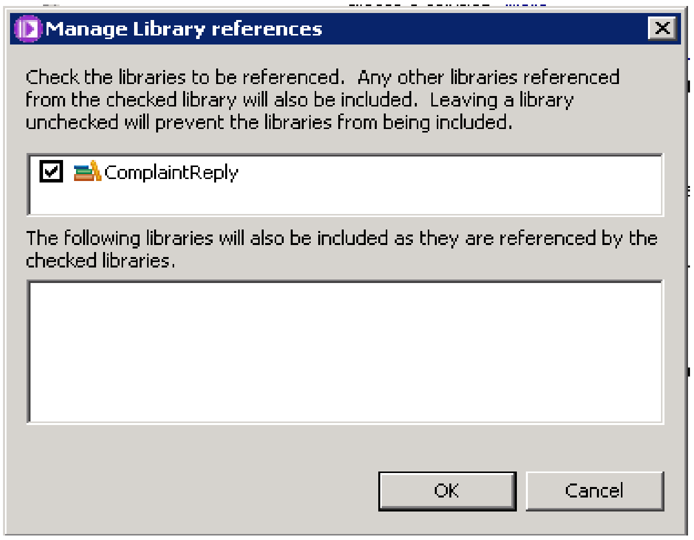
In this part of the exercise, you add a reference to the ComplaintRoute application for the new Complaint_Reply DFDL schema and add nodes to the main message flow to generate the Complaint_Reply message.

__ 1. Add a reference to the Complaint_Reply DFDL schema in the **RouteComplaint** application so that you can use the model in a transformation.

__ a. Right-click the **RouteComplaint** application in the Application Development navigator.

__ b. Select **Manage Library References**.

__ c. Select **ComplaintReply** and then click **OK**.



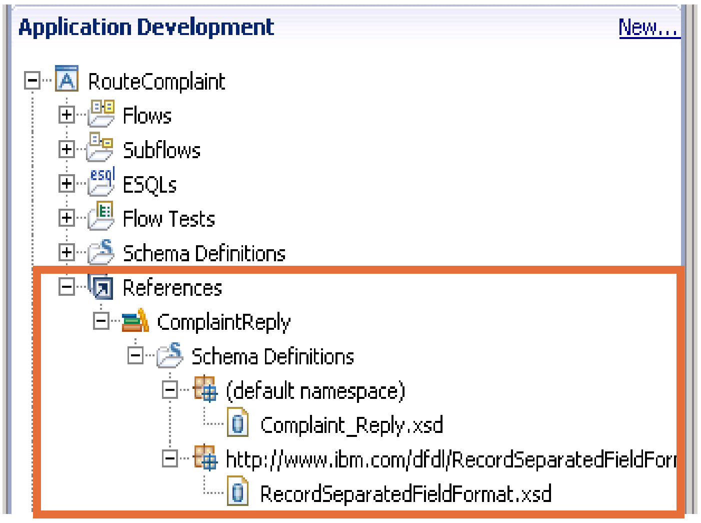
Manage Library references

Check the libraries to be referenced. Any other libraries referenced from the checked library will also be included. Leaving a library unchecked will prevent the libraries from being included.

☑ ComplaintReply

The following libraries will also be included as they are referenced by the checked libraries.

__ d. Expand the **RouteComplaint** application in the Application Development navigator and verify that the application now includes a reference to the **ComplaintReply** library.



__ 2. Add the new nodes to the **RouteComplaint** message flow **MyFlow.msgflow**.

  __ a. Open the message flow in the Message Flow editor.

  __ b. Add one Compute (or JavaCompute) node from the **Transformation** drawer and one MQReply node from the **WebSphere MQ** drawer.

  __ c. Rename the new Compute (JavaCompute) node to: `Compute Reply`

__ 3. Wire the new nodes as follows:

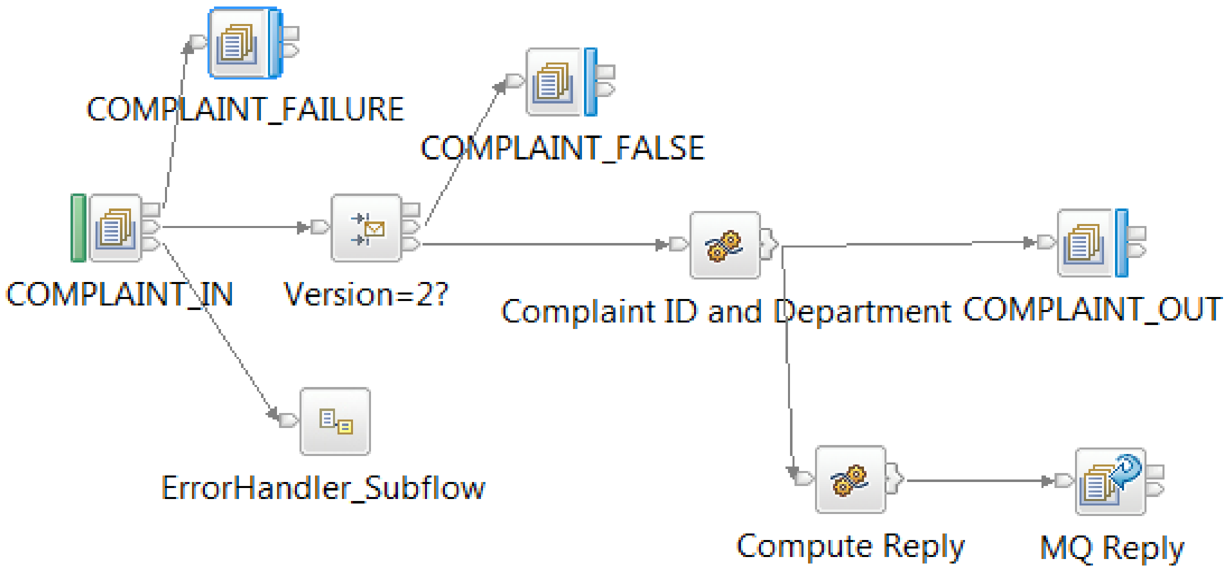
  __ a. Wire the **Complaint ID and Department** Compute node **Out** terminal to the **Compute Reply** Compute node **In** terminal.

Note

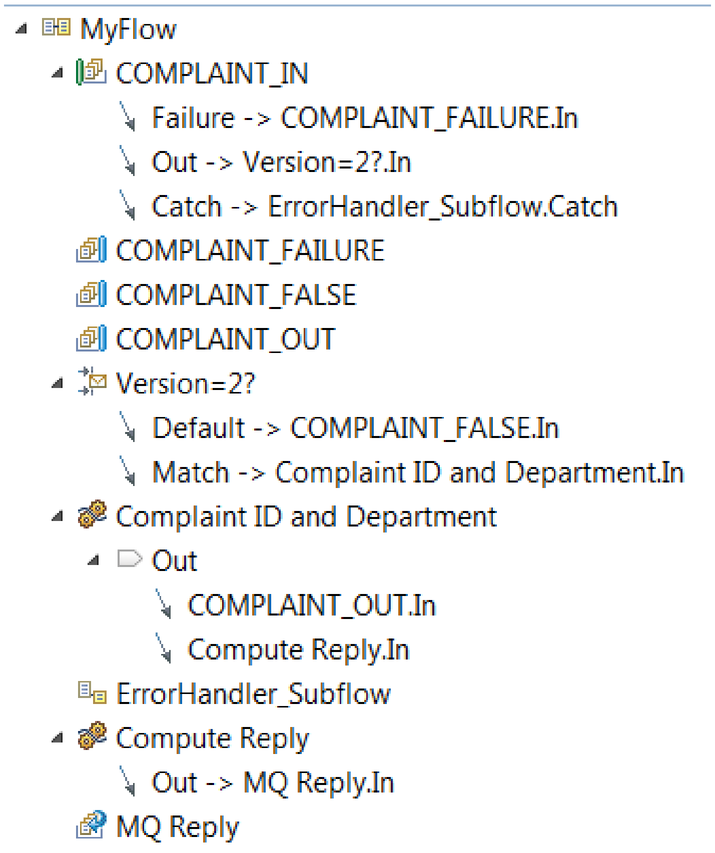
Use care when attaching the wire to the **Compute Reply** Compute node. Because the **Out** terminal is already in use, the Message Flow editor selects **Out1** as the default terminal.

Both the **COMPLAINT_OUT** MQOutput node and the **Compute Reply** Compute node should be wired to the **Out** terminal of **Complaint ID and Department**. Verify that you wired the correct terminal.

  __ b. Wire the **Compute Reply** Compute node **Out** terminal to the **MQReply In** terminal.



You can also use the Outline view to verify the connections. If the Outline view is not available, select **Window > Show View > Outline**.



```
MyFlow
  COMPLAINT_IN
    Failure -> COMPLAINT_FAILURE.In
    Out -> Version=2?.In
    Catch -> ErrorHandler_Subflow.Catch
  COMPLAINT_FAILURE
  COMPLAINT_FALSE
  COMPLAINT_OUT
  Version=2?
    Default -> COMPLAINT_FALSE.In
    Match -> Complaint ID and Department.In
  Complaint ID and Department
    Out
      COMPLAINT_OUT.In
      Compute Reply.In
  ErrorHandler_Subflow
  Compute Reply
    Out -> MQ Reply.In
  MQ Reply
```

Option 1: Write a transformation in ESQL for the Compute node

__ 1. Configure the node properties for the new compute node.

On the **Compute Reply** node **Basic** tab, set the **ESQL module** property to: `ComputeReply`

__ 2. Write ESQL for the **Compute Reply** node to create the reply message body in the DFDL domain.

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\Lab08-Modeling\resources` folder.

__ a. Double-click the **Compute Reply** node. The ESQL editor opens on a new module.

__ b. Write ESQL to:

- Copy the message headers
- Set `OutputRoot.Properties` for message set `Complaint_Reply.msd` and message type to `{}:Complaint_Reply`.

- Set the `OutputRoot DOMAIN` to: `DFDL`

__ c. Create the message body. Use content assist (**Ctrl + Space**) to fill the message path for the DFDL part.

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

```
CREATE COMPUTE MODULE Compute_Reply
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        -- Set Properties for DFDL domain
        SET OutputRoot.Properties.MessageSet = 'Complaint_Reply.msd';
        SET OutputRoot.Properties.MessageType = '{}:Complaint_Reply';
        -- Fill Complaint_Reply message field-by-field
        CREATE LASTCHILD OF OutputRoot DOMAIN('DFDL');
        SET OutputRoot.DFDL.Complaint_Reply.YourComplaint.Type =
        InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TYPE;
        SET OutputRoot.DFDL.Complaint_Reply.YourComplaint.Reference =
        InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_REF;
        SET OutputRoot.DFDL.Complaint_Reply.YourComplaint.Text =
        InputBody.CUSTOMERCOMPLAINT.COMPLAINT.C_TEXT;

        SET OutputRoot.DFDL.Complaint_Reply.Reply.OurReference =
        InputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.DEPT
        || '-' ||
        InputRoot.XMLNSC.CUSTOMERCOMPLAINT.ADMIN.REFERENCE;
        SET OutputRoot.DFDL.Complaint_Reply.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. Save the **ComputeReply** ESQL file.

__ 4. Save the message flow.

__ 5. Verify that there are no errors or warnings for this message flow.

Option 2: Write a transformation in Java for the JavaCompute node

__ 1. Configure the node properties for the new JavaCompute node.

For the JavaCompute **Compute Reply** node, set the **Basic, Java class** property to: `ComputeReply`

__ 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**.

__ 3. Write Java to set `Properties` for the message set to `Complaint_Reply.msd` and message type to `{}:Complaint_Reply`.

__ a. Create a message body in the `DFDL` domain.

__ b. Create the message body. The structure of `YourComplaint` is filled with matching fields from the input message `CUSTOMERCOMPLAINT.COMPLAINT`.

__ c. `Reply.OurReference` is a concatenation of input `CUSTOMERCOMPLAINT.ADMIN.DEPT` and `'-'` and `CUSTOMERCOMPLAINT.ADMIN.REFERENCE`.

__ d. `Reply.Text` is a literal text.

Note

You can copy the Java from the `C:\Labs\Lab08-Modeling\resources\Cut&Paste.txt` file.

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

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

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

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

Part 4: Test with the Test Client

__ 1. The BAR file must contain all the resources that the message flow requires to run. As the message flow increases in complexity, the size of the BAR file might also increase. To avoid possible timeout errors when deploying the BAR with the Test Client in this exercise, the BAR file is deployed manually.

__ a. Right-click the **RouteComplaint** application in the Application Development navigator and then select **New > Bar** file.

__ b. Enter `RouteComplaintWithReply` for the BAR file name and then click **Finish**.

__ c. On the BAR file editor **Prepare** tab, select the **RouteComplaint** application and then click **Build and Save**. The BAR file is created under the `BARs` folder and the BAR file editor opens.

__ d. Click the BAR file editor **Manage** tab and verify the BAR file components.

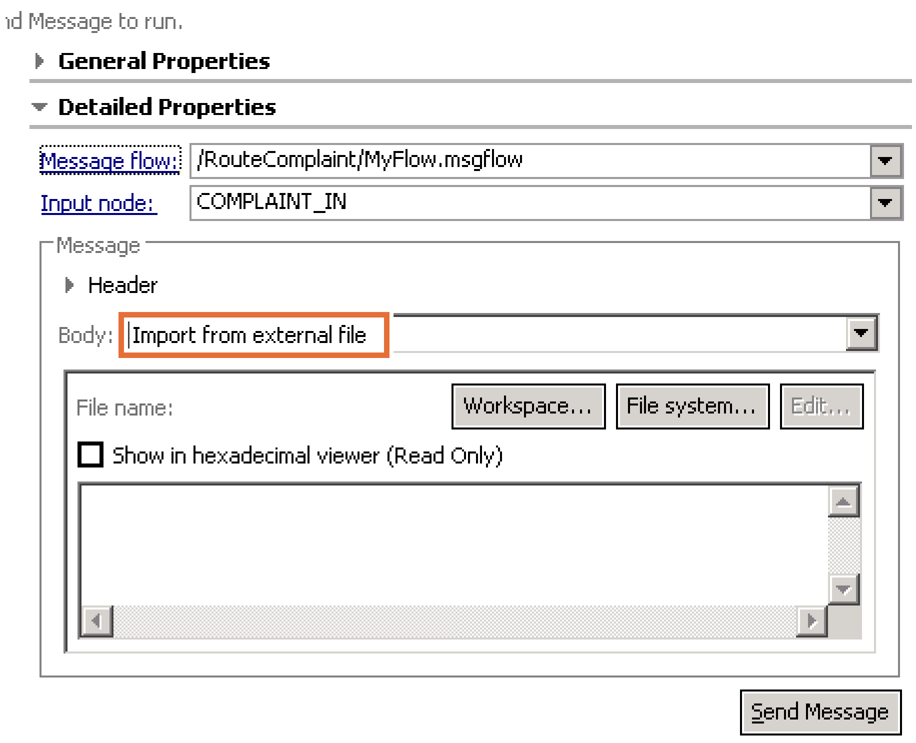
__ 2. Deploy the BAR file to the **default** integration server.

Drag the **RouteComplaintWithReply.bar** BAR file from the **BARs** folder in the Application Development navigator to the **default** integration server in the **Integration Nodes** view.

__ 3. Start the Test Client by right-clicking the **RouteComplaint** application in the Application Development navigator and then selecting **Test**.

__ 4. The input node expects an XML message, but this time the message flow is not associated with a message definition. Therefore, you must import source data from the following file: `C:\Labs\Lab08-Modeling\data\Complaint_2d.xml`

__ a. Under **Detailed Properties**, in the **Message** section, change **Body** to **Import from external file**.



Message flow: /RouteComplaint/MyFlow.msgflow

Input node: COMPLAINT_IN

Body: Import from external file

__ b. Click **File system,** and then browse to `C:\Labs\Lab08-Modeling\data` and import the `Complaint_2d.xml` file.

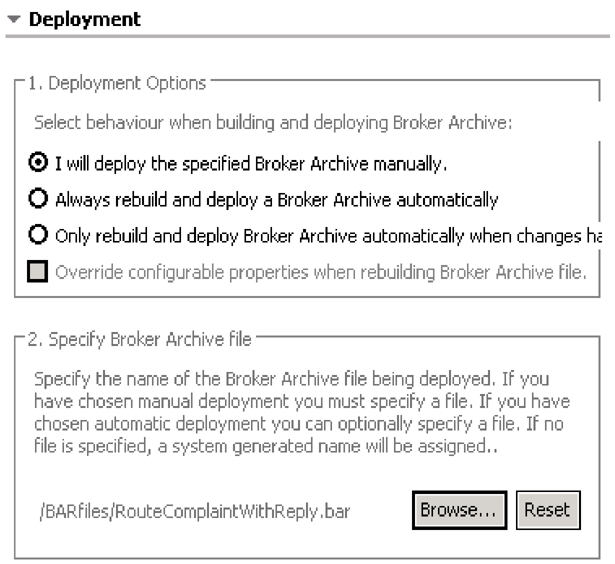
__ 5. Configure the Test Client to use the BAR file that was manually deployed to the **default** integration server.

__ a. Click the **Configuration** tab of the Test Client.

__ b. Select **Deployment**.

__ c. Under the **Deployment Options**, select **I will deploy the specified Broker Archive manually**.

__ d. Under **Specify Broker Archive file**, click **Browse**, select **RouteComplaintWithReply.bar**, and then click **OK**.



Deployment

1. Deployment Options

Select behaviour when building and deploying Broker Archive:

[✓ I will deploy the specified Broker Archive manually.]

Always rebuild and deploy a Broker Archive automatically

Only rebuild and deploy Broker Archive automatically when changes ha

Override configurable properties when rebuilding Broker Archive file.

2. Specify Broker Archive file

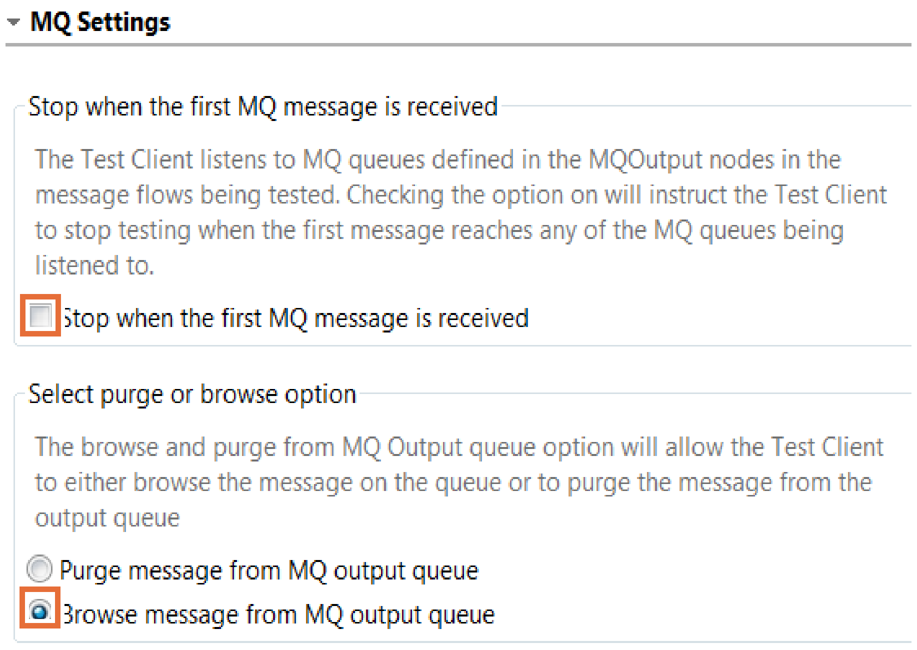
Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

/BARfiles/RouteComplaintWithReply.bar

__ 6. When this flow runs correctly, it generates two output messages: one on the COMPLAINT_OUT queue, and one on the COMPLAINT_REPLY queue.

__ a. On the Test Client Configuration tab, click **MQ Settings**.

__ b. Clear the **Stop when the first MQ message is received** option and set the **Browse message from MQ output queue** 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

Select purge or browse option

The browse and purge from MQ Output queue option will allow the Test Client to either browse the message on the queue or to purge the message from the output queue

Purge message from MQ output queue

[✓ Browse message from MQ output queue]

Note

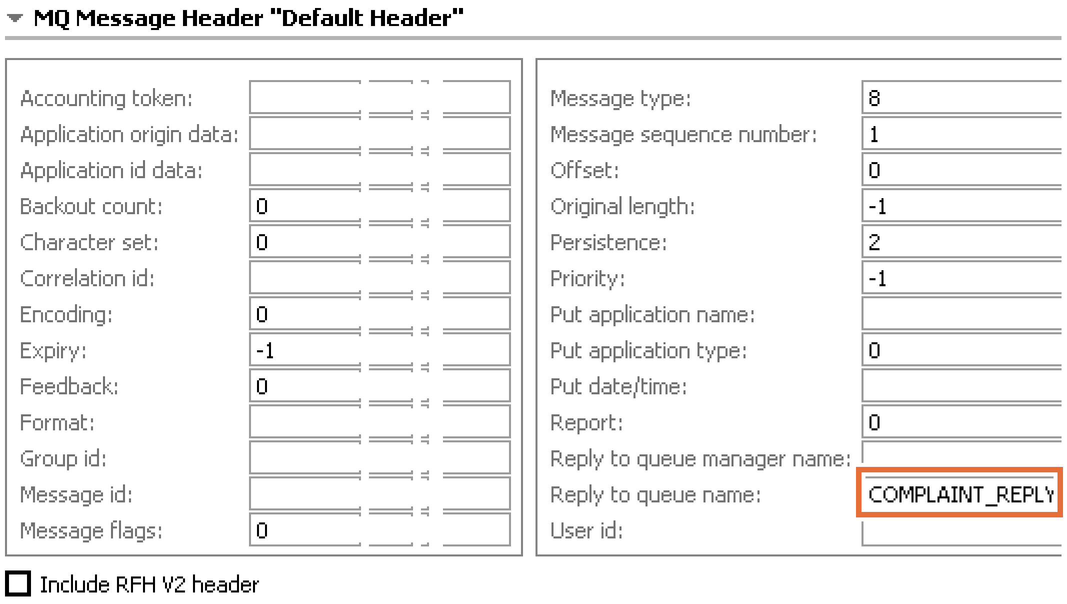
Now the Test Client waits indefinitely for output messages, rather that timing out. You must explicitly stop it to rerun or reinvoke the test.

Also, the Test Client no longer deletes the messages from the output queues. Now you can also use RFHUtil or WebSphere MQ Explorer to examine the message queues and contents.

__ 7. Configure the MQMD ReplyToQueue in the Test Client.

__ a. On 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

__ 8. Verify that all queues that the application uses allow messages to be put. Some of them might still be "put inhibited" from your testing of previous exercises. Use WebSphere MQ Explorer to verify, and change if necessary.

__ 9. Send the test message.

__ a. On the **Events** tab, click **Send Message**. The Select Deployment Location is displayed.

__ b. Click **Finish**.

__ 10. Inspect the outcome of this test. You see that the XML complaint reply message was sent to the COMPLAINT_OUT queue.

You should also see that a message was sent to the COMPLAINT_REPLY queue. The contents of the message should be delimited text:

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

You can use WebSphere MQ Explorer or RFHUtil to review the queues.

To use RFHUtil:

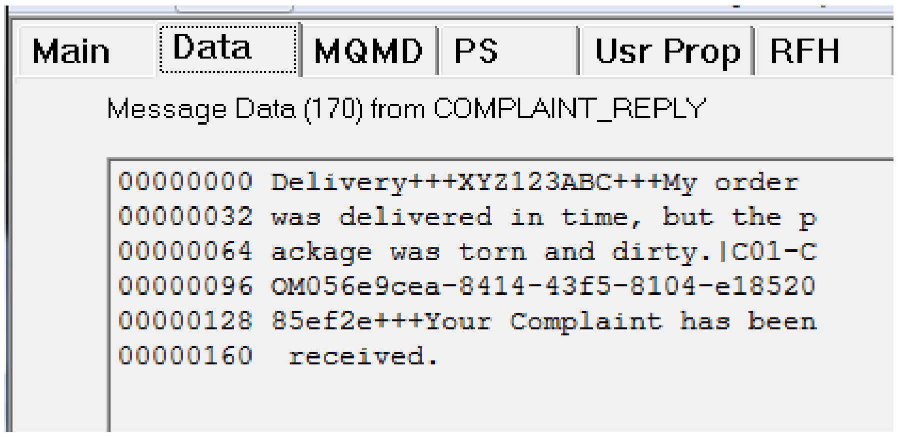
__ a. Start RFHUtil by double-clicking `rfhutil.exe` in the `C:\Labs\Tools` directory or clicking the RFHUtil icon on the Windows taskbar.

__ b. For **Queue Manager**, select **IB9QMGR**.

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

__ d. Click **Browse Q** to browse the message.

__ e. Click the **Data** tab. The data should match the format of the data that is described in the exercise introduction.



Message Data (170) from COMPLAINT_REPLY

```
00000000 Delivery+++XYZ123ABC+++My order
00000032 was delivered in time, but the p
00000064 ackage was torn and dirty.|C01-C
00000096 OM056e9cea-8414-43f5-8104-e18520
00000128 85ef2e+++Your Complaint has been
00000160  received.
```

__ 11. If the queues are empty, check the Windows Event View and ERROR_OUT queue for error messages. Correct the problem and try 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 to Part 5 (Optional)

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 9. Referencing a database in a map

What this exercise is about

In this exercise, you use a Database node in a message flow to store a message in a database. You also import a COBOL Copybook and XML schema to create a data model, and use the Graphical Data Mapping editor to transform the message.

What you should be able to do

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

- Discover database definitions
- Add a database node to a message flow
- Create the logic to store a message in a database
- Use the Graphical Data Mapping editor to map message elements
- Reference an external database when mapping message elements

Introduction

In this exercise, you extend the Complaint message flow that you created in previous exercises.

All messages are predefined in a DFDL model with input from a COBOL Copybook and output from an XML schema file. As soon as the message arrives in the flow, it is stored in a database for auditing purposes. The output message is enhanced with information about the manager of the department that is to handle the complaint. This data is selected from a database.

Requirements

- A workstation with IBM Integration Bus components installed and
- The IBM Integration Bus default configuration
- The files in `C:\Labs\Lab09-DB` and `C:\Labs\Tools`
- DB2 with SAMPLE and MSGSTORE databases (created by running the `Create_Q_DB.cmd` script in the `C:\Labs\Tools` directory).
- WebSphere MQ queues that are created by running the `Create_Q_DB.cmd` script (in the `C:\Labs\Tools` directory)

Input message example: DFDL CWF

```
2Ed        Fletcher  Mail Point 135      Hursley Park        WinchesterSO21
2JN  UKDelivery  XYZ123ABC I placed an order on 15-11-99, well in time for
Christmas and I still have not had a delivery schedule sent to me.  Please
cancel the order and refund me NOW.                                          X
```

Output message example:

```
<Complaint_Out>
    <Version>2</Version>
    <Customer>
    <Name>
        <First>Ed</First>
        <Last>Fletcher</Last>
    </Name>
    <Address>
        <Line>Mail Point 135</Line>
        <Line>Hursley Park</Line>
        <Town>Winchester</Town>
        <Zip>SO21 2JN</Zip>
        <Country>UK</Country>
    </Address>
    <Complaint>
        <Type>Delivery</Type>
        <Reference>XYZ123ABC</Reference>
        <Text>I placed an order on 15-11-99, well in time for Christmas and I
still have not had a delivery schedule sent to me.  Please cancel the order
and refund me NOW.</Text>
    </Complaint>
        <Admin>
    <Reference>COMda1b71b2</Reference>
    <Manager>
    <FirstName>SALLY A.</FirstName>
    <LastName>Kwan</LastName>
    <Phone>4738</Phone>
    </Manager>
    <Text>Please action</Text>
    </Admin>
```

The reply message is the same as in the previous exercise.

Exercise instructions

Only new steps are described in detail and with screen captures. See previous exercises for more instructions if necessary.

Part 1: Start components and import resources

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

__ 2. Make sure that all local IBM Integration Bus components are running.

__ 3. Start a new workspace.

__ a. Click **File > Switch Workspace > Other** from the toolbar. The Workspace Launcher is displayed.

__ b. For Workspace, enter `c:\Workspace\EX09` and then click **OK**.

__ c. When the new workspace opens, close the Welcome pane.

__ 4. Import the starting application and library from the `DB_ProjectInterchange.zip` file in `C:\Labs\Lab09-DB\resources` directory.

Select the **ComplaintReply** and **RouteComplaint** folders.

The project interchange file is opened in the Application Development view and the workspace is built.

Part 2: Import the copybook

In this part of the exercise, you create a library that contains the DFDL message model schema for a COBOL Copybook.

__ 1. Create a library that is named **Complaint**.

__ a. In the IBM Integration Toolkit Integration Development perspective, select **File > New > Library**.

__ b. For the **Library Name**, enter `Complaint` and then click **Finish**.

__ 2. Create a DFDL message model schema from the `C:\Labs\Lab09-DB\data\COMPLAINT_IN.cpy` COBOL Copybook file:

__ a. Under the **Complaint** library folder in the Application Development view, click **New > Message Model** to start the New Message Model wizard.

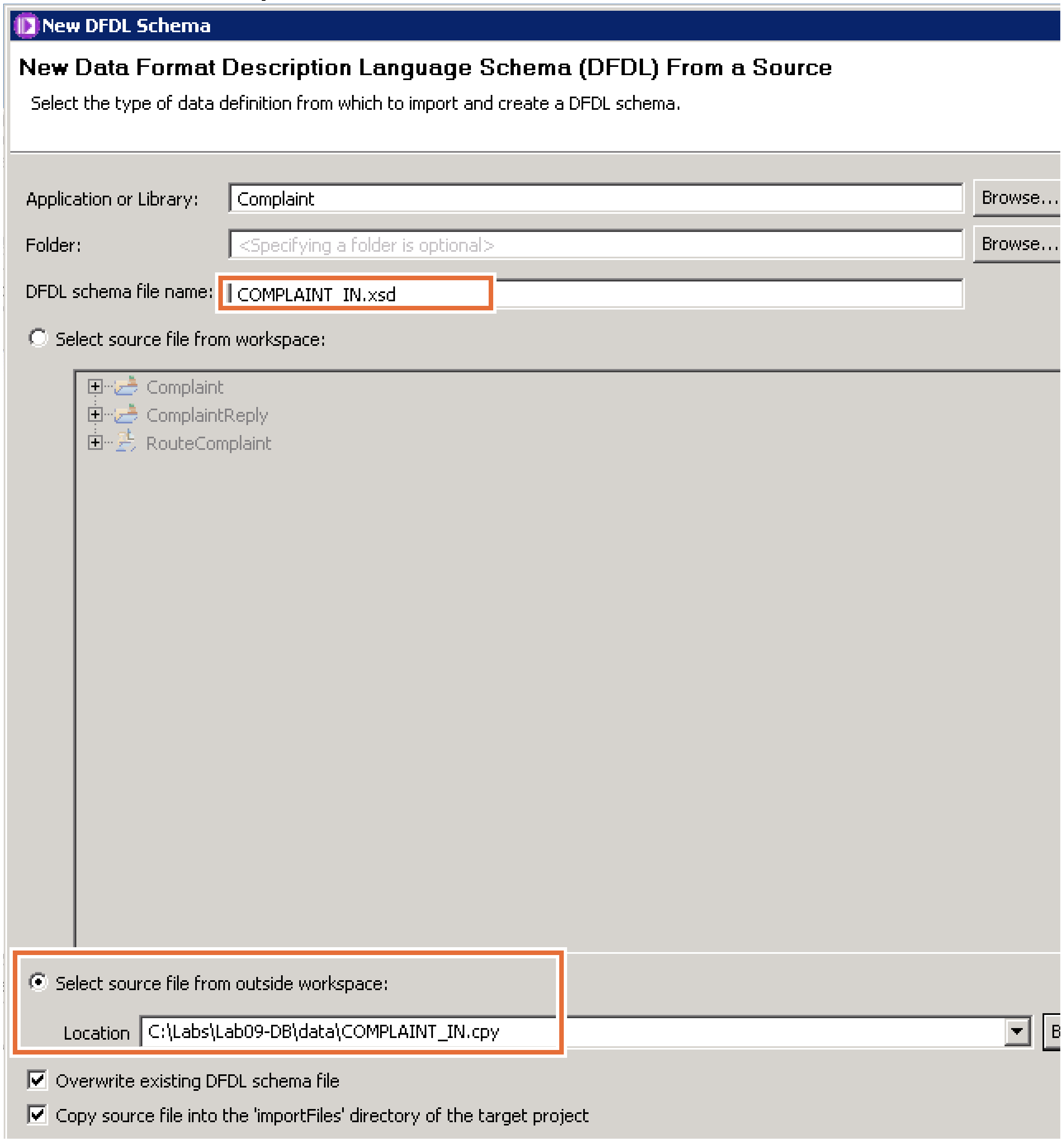
__ b. Select **COBOL** and then click **Next**.

__ c. Select **Create a DFDL schema file by importing a COBOL Copybook or program** and then click **Next**.

__ d. Click **Select source from outside workspace**.

__ e. To the right of **Location**, click **Browse**.

__ f. Browse to `C:\Labs\Lab09-DB\data` directory, select **COMPLAINT_IN.cpy**, and then click **Open**.



New DFDL Schema

New Data Format Description Language Schema (DFDL) From a Source

Select the type of data definition from which to import and create a DFDL schema.

Application or Library: Complaint

Folder: <Specifying a folder is optional>

DFDL schema file name: COMPLAINT_IN.xsd

Select source file from workspace:

- Complaint
- ComplaintReply
- RouteComplaint

(●) Select source file from outside workspace:

Location C:\Labs\Lab09-DB\data\COMPLAINT_IN.cpy

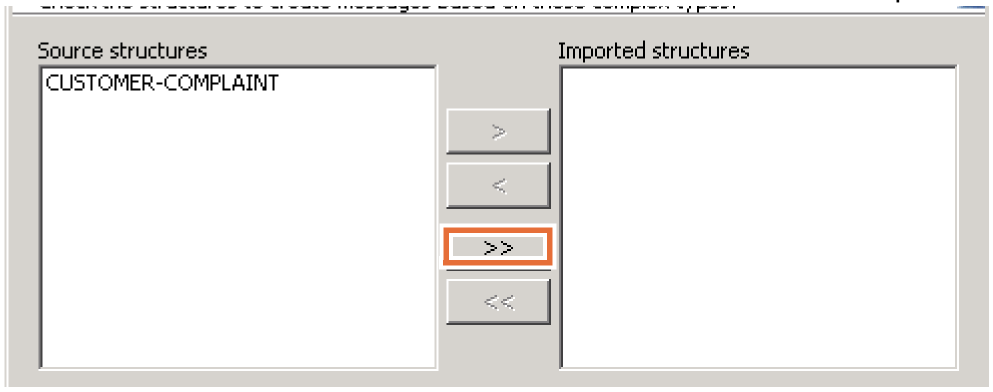
[✓] Overwrite existing DFDL schema file

[✓] Copy source file into the 'importFiles' directory of the target project

__ g. Click **Next**.

The COBOL Copybook is read and analyzed. Next, you must select which source 01 level structures to import, and whether to create a message from them. For this exercise, the copybook contains only one 01 level structure.

__ h. On the Structure and Message Selection page, verify that **CUSTOMER-COMPLAINT** is shown in the **Source structures** pane.



__ i. Click **>>** to import all source structures.

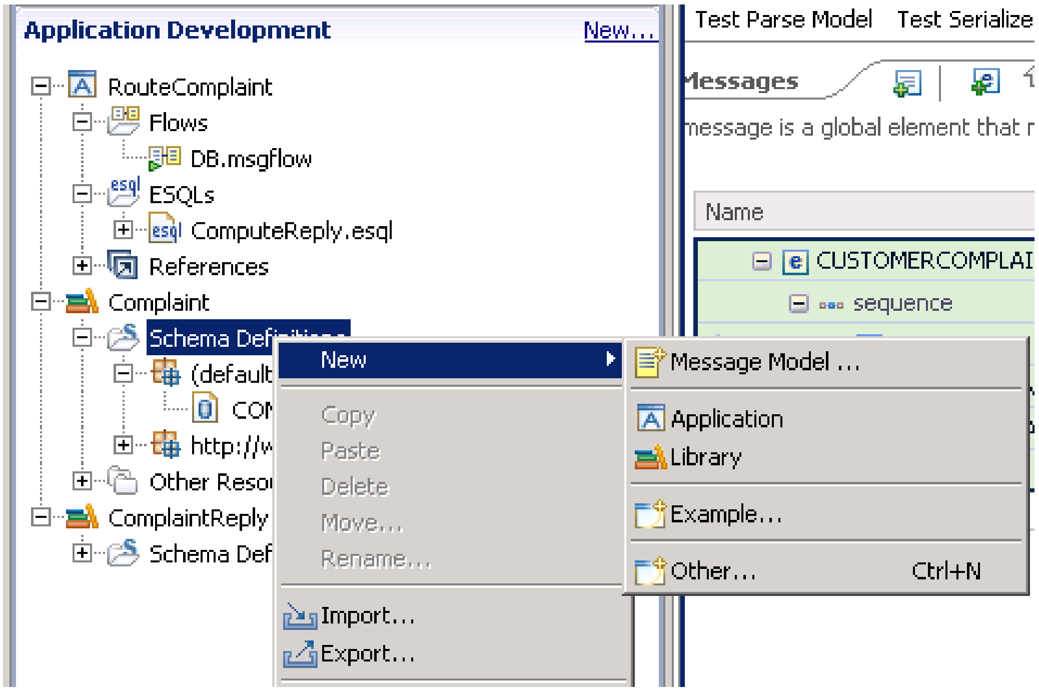
__ j. Click **Finish**. The DFDL schema `COMPLAINT_IN.xsd` is generated and is opened in the DFDL schema editor.

Review the schema so that you understand its structure and elements.

Part 3: Import XML schema

In this part of the exercise, you create a message definition from the XML schema `C:\Labs\Lab08-DB\data\COMPLAINT_OUT.xsd` in the Complaint library.

__ 1. In the Application Development view, right-click **Schema Definitions** under the **Complaint** library folder and then select **New** > **Message Model**.



The New Message Definition File menu is displayed.

__ 2. Under the XML section, select **Other XML** and then click **Next**.

__ 3. On the Other XML page, select **I already have an XML schema file for my data** and then click **Next**.

__ 4. Click **Select file from outside workspace** and then click **Browse**.

__ 5. Browse to `C:\Labs\Lab08-DB\data`, select **COMPLAINT_OUT.xsd**, and then click **Open**.

__ 6. Click **Finish**.The XML schema is read and analyzed.

__ 7. Review the newly created message definition **COMPLAINT_OUT.xsd** in the **Outline** view. The `COMPLAINT_OUT.xsd` file also opens in the schema editor.

Part 4: Discover the database definition

In this part, you create a container to hold the JDBC connections that the message flow uses to access the DB2 databases. You then define the connections.

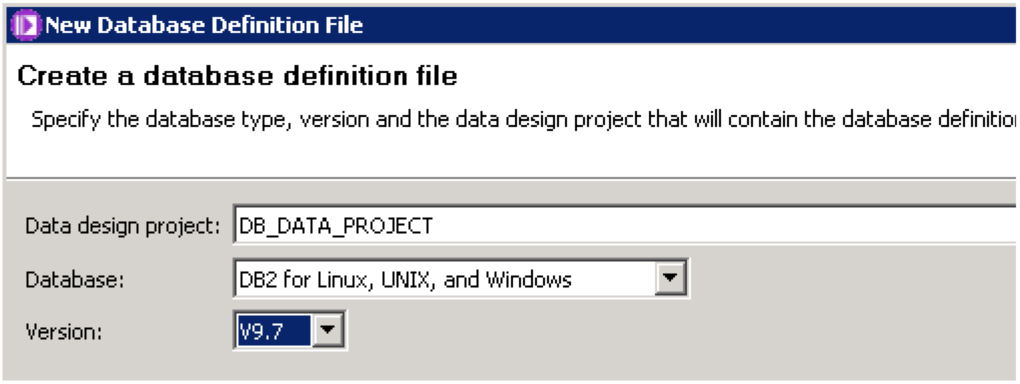
__ 1. Discover the database definition for the DB2 database SAMPLE, schema ADMINISTRATOR, and store it in DB_DATA_PROJECT.

__ a. From the toolbar, click **File** > **New > Database Definition**. The New Database Definition File menu is shown.

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

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

__ d. For the **Version**, select **V9.7** and then click **Next**.



New Database Definition File

Create a database definition file

Specify the database type, version and the data design project that will contain the database definitio

Data design project: DB_DATA_PROJECT

Database: DB2 for Linux, UNIX, and Windows

Version: V9.7

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

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

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

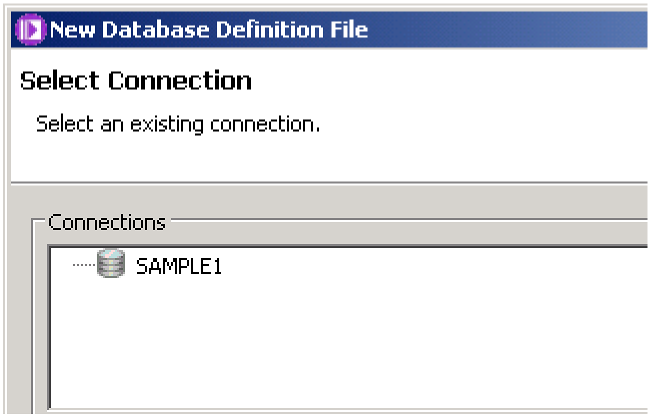
__ h. For **Password**, enter: `web1sphere`

__ i. Select **Save password**.

__ j. Click **Test Connection**. The JDBC connection is tested.

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

___ l. Click **Finish**. The Select Connection menu reappears, with the SAMPLE1 connection displayed.



New Database Definition File

Select Connection

Select an existing connection.

Connections

SAMPLE1

___ m. Select **SAMPLE1**, and then click **Next**. The **Select Schema** menu is shown.

___ n. Select **ADMINISTRATOR** and then click **Finish**. The model is created. The database connection opens in the Physical Model data editor.

___ o. Review the model, but do not change anything. Close the editor when you are finished reviewing the model.

___ 2. Discover the database definition for the DB2 database MSGSTORE, schema ADMINISTRATOR, and store it in DB_DATA_PROJECT.

___ a. From the toolbar, click **File** > **New > Database Definition**. The New Database Definition File menu is shown.

___ b. Select **DB_DATA_PROJECT** as the **Project name**, set **DB2 for Linux, UNIX, and Windows** for the **Database**, and set **Version** to **9.7**.

___ c. Click **Next**. The **Select Connection** menu opens.

___ d. Click **New**.

___ e. In the **Properties** pane, for **Database**, enter: `MSGSTORE`

___ f. For **User name**, enter: `Administrator`

___ g. For **Password**, enter: `web1sphere`

___ h. Select **Save password**.

___ i. Click **Test Connection**. The JDBC connection is tested.

___ j. A “Connection succeeded” message is shown. Click **OK**.

___ k. Click **Finish**. The Select Connection menu reappears.

___ l. In the **Connections** pane, select **MSGSTORE1**, and then click **Next**. The **Select Schema** menu opens.

___ m. Select **ADMINISTRATOR**, and then click **Finish**. The model is created.

___ n. The database connection opens in the Physical Model data editor.

__ o. Review the model, but do not change anything. Close the editor when you are finished reviewing the model.

Note

Database definition files are associated with the Data Project Explorer view and the Database Explorer view. Tools are available in these views for working with your databases. Database definition files are not automatically updated. If a change is made to your database, you must re-create the database definition files.

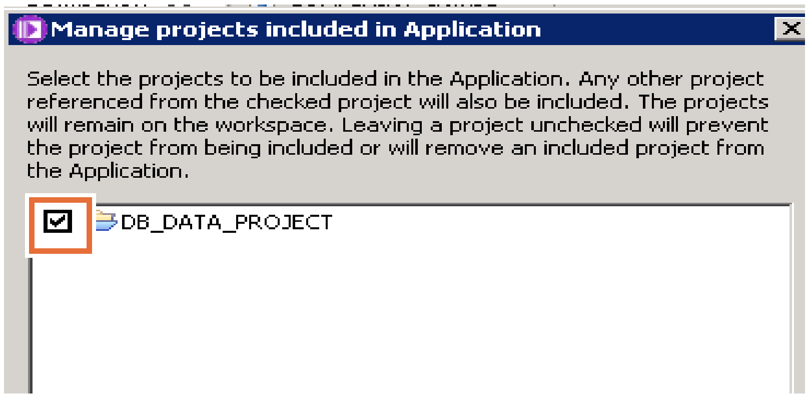
Part 5: Define references to models

In this part of the exercise, you define references to the database definition project and data definition library from the message flow.

__ 1. The `DB.msgflow` needs access to the database definitions that are stored in DB_DATA_PROJECT. So, you must add DB_DATA_PROJECT as a reference.

__ a. Right-click the **RouteComplaint** application in the Application Development view and then select **Manage included projects**.

__ b. Select **DB_DATA_PROJECT** and then click **OK**.


Manage projects included in Application

Select the projects to be included in the Application. Any other project referenced from the checked project will also be included. The projects will remain on the workspace. Leaving a project unchecked will prevent the project from being included or will remove an included project from the Application.

[✓] DB_DATA_PROJECT

__ 2. The `DB.msgflow` needs access to the schema data models that are stored in the Complaint library. So, you must add Complaint as a reference.

__ a. Right-click the **RouteComplaint** application in the Application Development view and then select **Manage Library References**.

__ b. Select **Complaint** and ensure that **ComplaintReply** is still selected.

__ c. Click **OK**.

Part 6: Complete the message flow

__ 1. In the Application Development view, expand **RouteComplaint > Flows**, and then double-click the **DB.msgflow** file.

As you can see, some nodes are missing from the message flow.

__ 2. Add one **Database** node from the **Database** drawer and rename it to: `Database Store Msg`

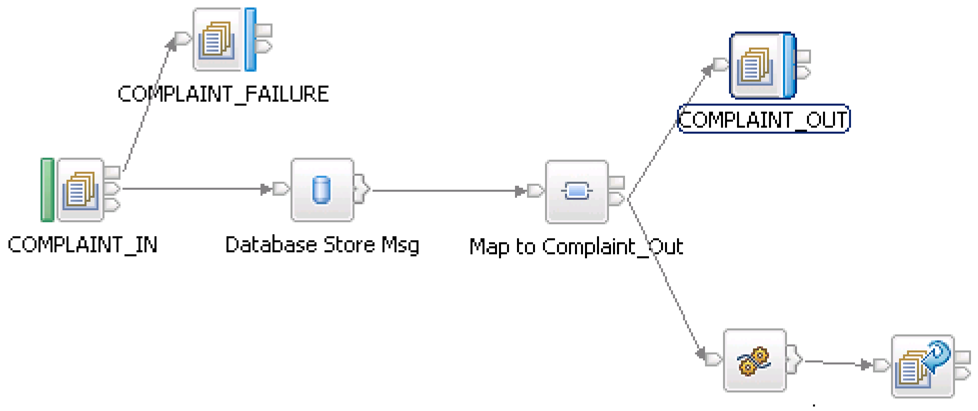
__ 3. Add one **Mapping** node from the **Transformation** drawer and rename it to: `Map to Complaint_Out`

__ 4. Connect the new nodes as follows:

__ a. Wire the COMPLAINT_IN node **Out** terminal to the Database Store Msg **In** terminal.

__ b. Wire the Database Store Msg node **Out** terminal to the **Map to Complaint_Out** node **In** terminal.

__ c. Wire the **Map to Complaint_Out** node **Out** terminal to the **Compute Reply** node **In** terminal and the **MQOutput COMPLAINT_OUT** node **In** terminal.



__ 5. Configure the node properties.

Node	Properties
COMPLAINT_IN	On the **Input Message Parsing** tab, select (do not type): **Message Domain = DFDL** **Message = { }:CUSTOMERCOMPLAINT**
Database Store Msg	Data source = MSGSTORE
Map to Complaint_Out	Use the default values

Part 7: Write ESQL for the Database node

__ 1. Double-click the **Database Store Msg** node. The ESQL editor opens on a new module within the existing ESQL file.

__ 2. Write ESQL to insert into the **COMPLAIN** table; the **MESSAGE**, **MSGID**, and **RECEIVED** columns; the complete message bit stream; and the MessageID and Timestamp for identification.

__ 3. Use content assist (Ctrl+Space) to fill the message path.

Note

You can copy the ESQL from `C:\Labs\Lab09-DB\resources\Cut&Paste.txt`.

The completed module is similar to the following code:

```
CREATE DATABASE MODULE StoreMsg
 CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
    INSERT INTO Database.ADMINISTRATOR.COMPLAIN(MSGID, RECEIVED,
       MESSAGE)
    VALUES (Root.MQMD.MsgId,CURRENT_TIMESTAMP,ASBITSTREAM(Root));
    RETURN TRUE;
  END;
END MODULE;
```

__ 4. Save the file and close the ESQL editor.

Note

Warning messages about ambiguous database table references are OK; the references are resolved at run time.

Part 8: Create the mappings

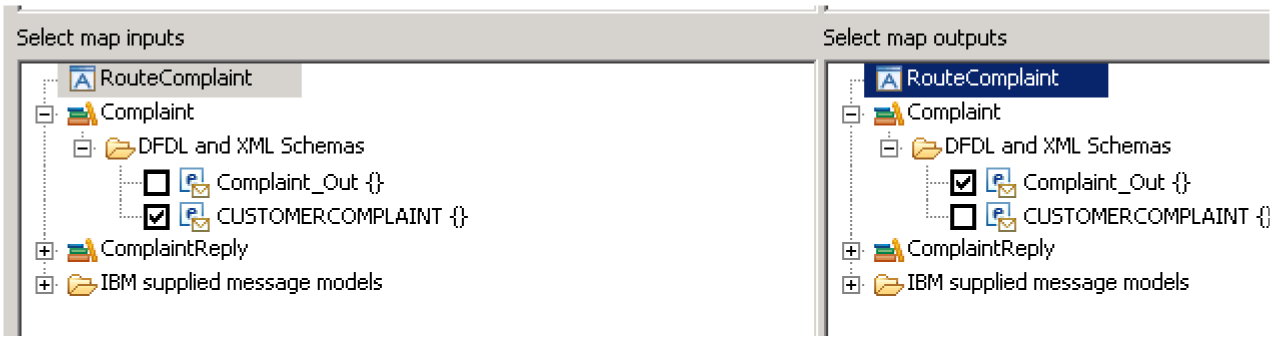
__ 1. Create a map.

__ a. Double-click the **Map to Complaint_Out** Mapping node in the message flow. The New Message Map menu opens. Accept the default values on this page for container and map name, and click **Next**.

The **Select map inputs and outputs** menu opens.

__ b. Under **Select map inputs**, expand **Complaint > DFDL and XML schemas** and then select **CUSTOMERCOMPLAINT { }**.

__ c. Under **Select map outputs**, expand **Complaint > DFDL and XML schemas**, and then select **Complaint_Out { }**.



__ d. Click **Finish**. The Mapping editor opens.

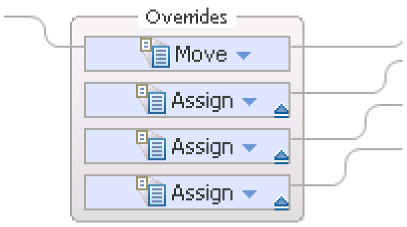
__ 2. You must define how the source elements map to the target elements.

If you choose not to map message headers or the LocalEnvironment explicitly in your message map, the output message is produced with the same message headers as the input message.

When you populate the message map, the **Properties** folder for the source and target are displayed in the message map. MessageSet and MessageType are initially set based on the target message.

View the map properties.

Observe that the mapping is already defined between the source and target properties:



The mapping options are:

- A **Move** transform is created that maps the entire source message to the entire target message.
- An **Assign** transform on the target message for the **MessageSet** field. Click the **Assign** transform and view the **Properties**. In the **General** tab, **Value** is set to DB_SET.
- An **Assign** transform on the target message for the **MessageType** field. Click the **Assign** transform and view the **Properties**. In the **General** tab, **Value** is set to { } Compaint_Out.
- An **Assign** transform on the target message for the **MessageFormat** field. Click the **Assign** transform and view the **Properties**. In the **General** tab, **Value** is set to XML1.

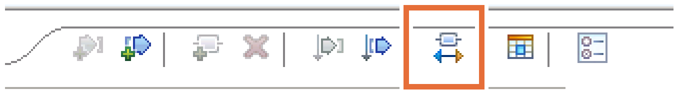
Note

If you click any of the transforms, or the wires between the source and target message, the editor colors the elements of the source and target that the transform affects. For example, if you click the **Move** transform, you see that the **Properties** folder in both the source and target are shaded in green. This visual cue helps you to see which elements a particular transform affects.

__ 3. Map the first part of the message body. Since many of the field names in the source and target are similar, use the auto map feature to quickly map the source structure to the target structure.

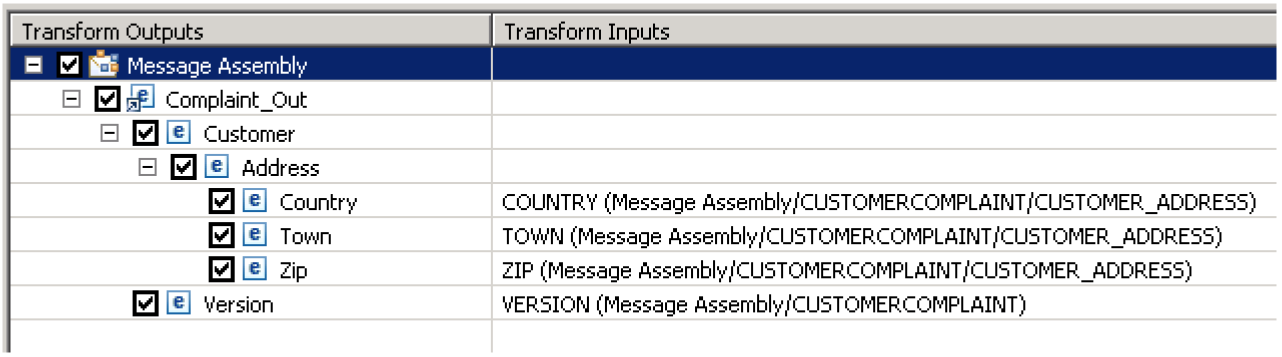
__ a. Click the **CUSTOMERCOMPLAINT** header in the source structure to select the entire message. The message turns blue to indicate that it is selected.

__ b. Click **Auto map input to output** (the seventh icon in the mapping toolbar).



The AutoMap properties menu opens.

__ c. Without changing any of the auto map parameters, click **Next**. A preview of the fields to be mapped is displayed.



Transform Outputs	Transform Inputs
☑ Message Assembly	
☑ Complaint_Out	
☑ Customer	
☑ Address	
☑ Country	COUNTRY (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Town	TOWN (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Zip	ZIP (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Version	VERSION (Message Assembly/CUSTOMERCOMPLAINT)

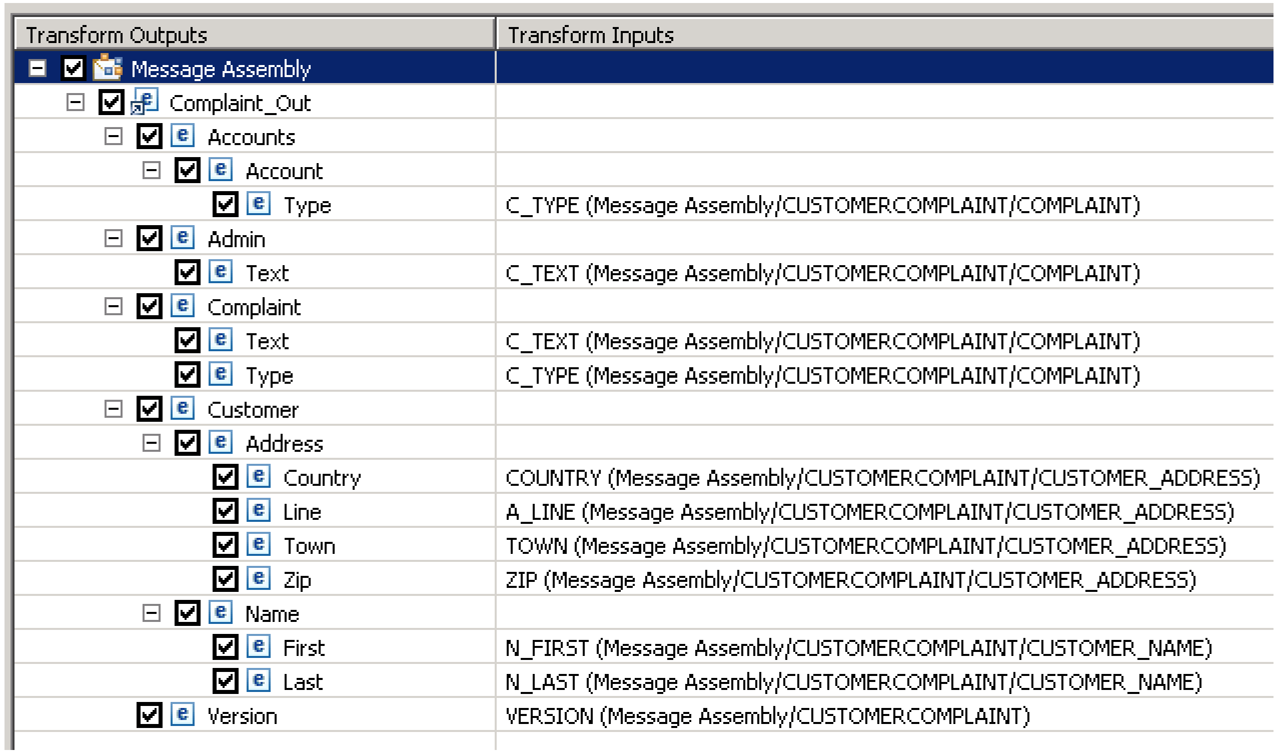
You see that only the fields that matched exactly on name are mapped. You can control this “strictness” in the matching so that names do not need to match exactly to match.

__ d. Click **Back**.

__ e. Under **Mapping Criteria**, click **Create transforms when the names of input and outputs are more similar than**.

__ f. Leave the default of **60%** selected.

__ g. Click **Next**. Now the preview shows that more matches were found, based on “looser” matching of the source and target fields.



Transform Outputs	Transform Inputs
☑ Message Assembly	
☑ Complaint_Out	
☑ Accounts	
☑ Account	
☑ Type	C_TYPE (Message Assembly/CUSTOMERCOMPLAINT/COMPLAINT)
☑ Admin	
☑ Text	C_TEXT (Message Assembly/CUSTOMERCOMPLAINT/COMPLAINT)
☑ Complaint	
☑ Text	C_TEXT (Message Assembly/CUSTOMERCOMPLAINT/COMPLAINT)
☑ Type	C_TYPE (Message Assembly/CUSTOMERCOMPLAINT/COMPLAINT)
☑ Customer	
☑ Address	
☑ Country	COUNTRY (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Line	A_LINE (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Town	TOWN (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Zip	ZIP (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_ADDRESS)
☑ Name	
☑ First	N_FIRST (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_NAME)
☑ Last	N_LAST (Message Assembly/CUSTOMERCOMPLAINT/CUSTOMER_NAME)
☑ Version	VERSION (Message Assembly/CUSTOMERCOMPLAINT)

__ h. Clear the check boxes for the assignments of **Account/Type** and **Admin/Text**.

__ i. Click **Finish**. The automatic mapping runs.

__ j. Review the results of the auto map.

__ k. The COMPLAINT/C_REF field did not get mapped to the Complaint/Reference field because the names were too dissimilar.

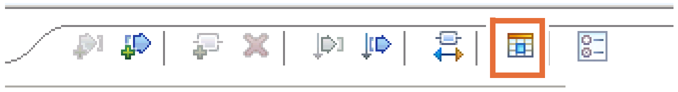
Manually map the fields by hovering the mouse cursor over COMPLAINT/C_REF in the source message. An orange wiring handle is shown. Drag the handle to the Complaint/Reference field in the target message. The editor makes a connection and creates a **Move** transform by default. Now, all the fields in the source message are mapped.

__ l. Save the message map (**Ctrl + S**).

Part 9: Select from the database

The manager information is selected from database SAMPLE, table EMPLOYEE. In this part of the exercise, you set up a database retrieval to return data to populate an output field.

__ 1. In the Mapping editor toolbar, click **Select rows from a database** (the eighth icon in the toolbar).



The New Database Select menu opens.

__ 2. Under **Choose a database to select from**, select **SAMPLE.**

__ 3. Under **Choose the columns to include**, expand **ADMINISTRATOR > EMPLOYEE**, and then select **FIRSTNME**, **LASTNAME**, and **PHONENO**.

__ 4. In the **SQL where clause**, under **Define a where clause**, enter:

```
EMPLOYEE.WORKDEPT = 'C01' AND EMPLOYEE.JOB = 'MANAGER'
```

This statement selects only rows from the database for Department C01 (which corresponds to Complaint type of “Delivery”), where the employee type is “MANAGER”

__ 5. Click **OK**. The Mapping editor pane returns.

__ 6. Scroll down under the source message, and you see the database retrieval that you created. It has a Select and Failure transform associated with it. The Select path is used if the database retrieval was successful. You map the fields that are returned from the database SELECT statement next.

__ 7. Connect the **Select** transform to **Complaint_Out.Admin**. Because you are mapping a structure to a structure, a local map is created. A message is displayed to tell you how to proceed with the local mapping of fields.

__ 8. Click the **Click here** or **Edit** link to open the editor for the nested map.

__ 9. Connect these **ResultSet** fields to the **Admin** fields:

- **FIRSTNME** to **Manager.FirstName**
- **LASTNAME** to **Manager.LastName**
- **PHONENO** to **Manager.Phone**

__ 10. Save the map (**Ctrl + S**), and close the mapping editor.

Note

The **Problems** view might contain *Cardinality* and *Ambiguous database table reference* and some *Unresolvable message field reference* warning messages. These references are resolved at run time and can be ignored.

__ 11. Save the message flow (**Ctrl + S**).

Part 10: Test with the Test Client

__ 1. Create and manually deploy the BAR file for the application.

__ a. Right-click the RouteComplaint application in the Application Development view and then select **New > Bar** file.

__ b. For the BAR file name, enter: `DB`

__ c. Click **Finish**.

__ d. On the BAR file editor **Prepare** tab, select the RouteComplaint application and then select **Build and Save**.

__ e. Drag the `DB.bar` file from the Application Development view onto the **default** integration server in the **Integration Nodes** view.

__ 2. Start the Test Client for the RouteComplaint application. The input node expects a COBOL (binary) message. Therefore, you must import source data from the `C:\Labs\Lab09-DB\data\Complaint_2d_cwf.txt` file.

__ a. In the Application Development view, right-click the **RouteComplaint** application and then select **Test**.

__ b. Click **Import Source**, and Import `Complaint_2d_cwf.txt` from `C:\Labs\Lab09-DB\data`.

__ 3. Configure the Test Client for this message flow. You can expect two output messages (one to COMPLAINT_OUT, and one on COMPLAINT_REPLY).

__ a. Select the **Configuration** tab of Test Client.

__ b. In the Deployment options, select **I will deploy the specified Broker Archive manually** and then browse to and select the `DB.bar` file.

__ c. In **MQ Settings**, clear the **Stop when the first MQ message is received** option and select the **Browse message from MQ output queue** option.

Note

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

__ 4. Configure the MQMD ReplyToQueue in the Test Client.

__ a. In the **Configuration** tab of the Test Client, select **MQ Message Header “Default Header”**.

__ b. Enter a **Reply to queue name** of: `COMPLAINT_REPLY`

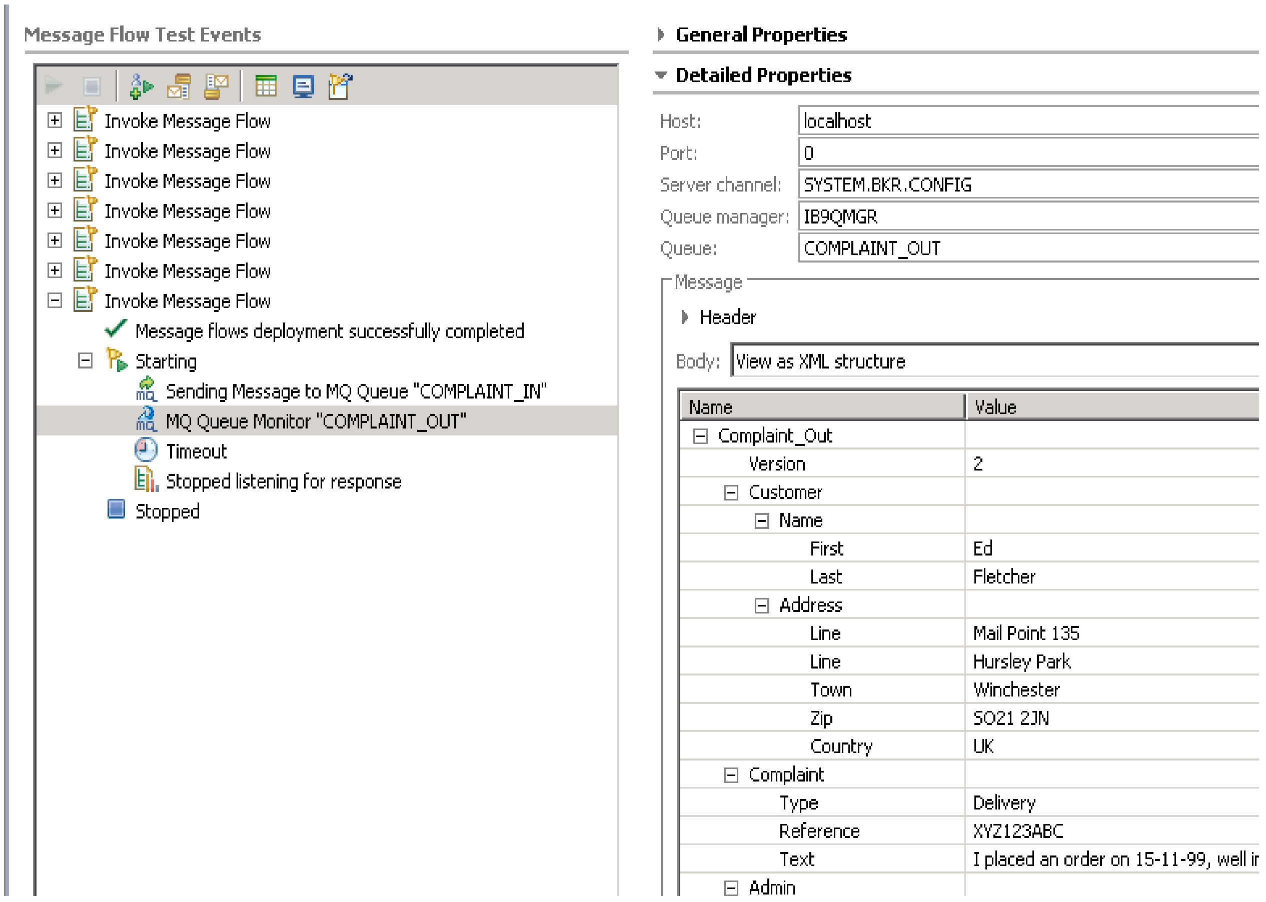
__ 5. Send the test message.

__ a. On the **Events** tab, click **Send Message**.

__ b. Select **default** integration server as the deployment location.

__ c. Click **Finish**.

__ 6. Verify that the message flow ran successfully.



Message Flow Test Events

- Invoke Message Flow
- Invoke Message Flow
- Invoke Message Flow
- Invoke Message Flow
- Invoke Message Flow
- Invoke Message Flow
- Invoke Message Flow
  - Message flows deployment successfully completed
  - Starting
    - Sending Message to MQ Queue "COMPLAINT_IN"
    - MQ Queue Monitor "COMPLAINT_OUT"
    - Timeout
    - Stopped listening for response
  - Stopped

General Properties

Detailed Properties

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

Message

Header

Body: View as XML structure

Name	Value
Complaint_Out	
Version	2
Customer	
Name	
First	Ed
Last	Fletcher
Address	
Line	Mail Point 135
Line	Hursley Park
Town	Winchester
Zip	SO21 2JN
Country	UK
Complaint	
Type	Delivery
Reference	XYZ123ABC
Text	I placed an order on 15-11-99, well ir
Admin	

__ 7. Use WebSphere MQ Explorer or RFHUtil to verify that you received a message in COMPLAINT_OUT and COMPLAINT_REPLY and that they are in the correct format as described in the exercise introduction.

__ 8. Use DB2 Control Center to review the inserted rows in the COMPLAIN table of the MSGSTORE database.

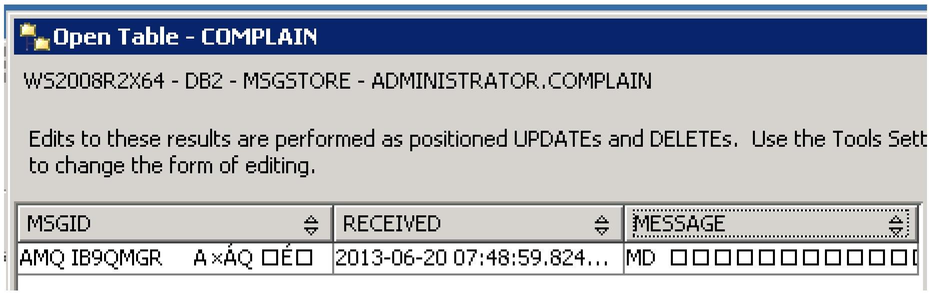
__ a. Click **Start > All Programs > IBM DB2 > General Administration Tools > Control Center** (or use the icon that is on the desktop of the exercise image).

__ b. Expand **All Databases > MSGSTORE**.

__ c. Click **Tables**.

__ d. In the upper-right pane, double-click **COMPLAIN**. The table opens in the editor.

__ e. It might be necessary to expand the columns. There are graphic characters in the MSGID and MESSAGE fields. These characters are generated because of the conversion of the message data into bitstream format.



WS2008R2X64 - DB2 - MSGSTORE - ADMINISTRATOR.COMPLAIN

Edits to these results are performed as positioned UPDATEs and DELETEs. Use the Tools Sett
to change the form of editing.

MSGID	RECEIVED	MESSAGE
AMQ IB9QMGR A×ÁQ □É□	2013-06-20 07:48:59.824...	MD □□□□□□□□□□□□[

__ f. Close the table editor.

__ g. Close the DB2 control center.

__ 9. If necessary, correct the message flow, and retest. Use the problem determination tools that you learned in this course to determine the problem.

End of exercise

Exercise 10.Creating a runtime-aware message flow

What this exercise is about

In this exercise, you modify an existing message flow to make it aware of its runtime environment.

What you should be able to do

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

- Configure nodes, message flows, integration servers, and integration node properties
- Define, configure, and use user-defined properties to provide runtime awareness within a message flow application
- Query various runtime attributes to allow the message flow to display runtime data and alter the flow of message processing

Introduction

Message flows can be made even more powerful and flexible if they can interact with the environment in which they are operating. A message flow that can determine whether it is running in a “production” environment might process differently than the same flow that is running in a “development” or “test” environment. Similarly, because of the reusability characteristics of a subflow, you might have multiple versions of it running in a particular environment. In that situation, it is useful to know what version is running.

Environment-aware message flows can assist you with these tasks. In addition, promoted properties help you change how a message flow operates at the BAR file level, rather than at the message flow level.

Requirements

- A workstation with the IBM Integration Bus components installed with the default configuration
- The files are in `C:\Labs\Lab010-UDP` and `C:\Labs\Tools`
- WebSphere MQ queues that the message flow requires

Exercise instructions

Part 1: Start components and import resources

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

__ 2. Make sure that all local IBM Integration Bus components are running.

__ 3. Start a new workspace:

__ a. Click **File > Switch Workspace > Other** from the toolbar. The Workspace Launcher is displayed.

__ b. For **Workspace**, enter `C:\Workspace\EX10` and then click **OK**.

__ c. When the new workspace opens, close the Welcome pane.

__ 4. Import the solution project interchange file for Exercise 7.

__ a. Click **File > Import > Other > Project Interchange**, and then click **Next**.

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

__ c. Browse to the `C:\Labs\Lab10-UDP\Resources` directory.

__ d. Select the **RouteComplaint_WithErrorHandlerSubflow_PI.zip** file and then click **Open**.

__ e. Ensure that **RouteComplaint** is selected, and then click **Finish**. The project interchange file is opened in the Application Development view, and the workspace is built.

Part 2: Add version and runtime information to the subflow

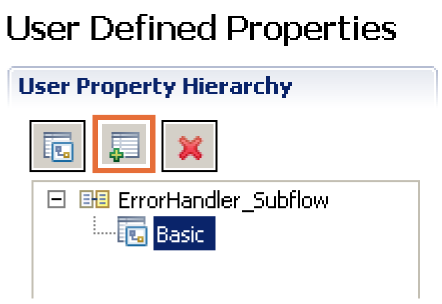
__ 1. In the Application Development view, expand **RouteComplaint > Subflows**.

__ 2. Double-click **ErrorHandler_Subflow** to open it in the Message Flow editor.

__ 3. Add a user-defined property to the subflow.

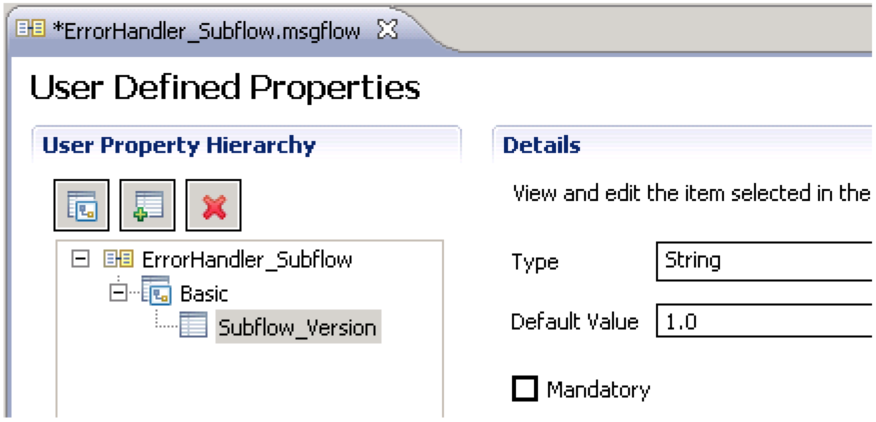
__ a. At the bottom of the Message Flow editor window, select the **User Defined Properties** tab.

__ b. In the **User Property Hierarchy**, under **ErrorHandler_Subflow**, right-click **Basic**, and then select **Add Property**.



__ c. Type over the default property name **Property1** with: `Subflow_Version`

__ d. Under **Details**, leave the **Type** set to **String**, and for **Default Value**, enter: `1.0`



__ e. Save the subflow (**Ctrl + S**).

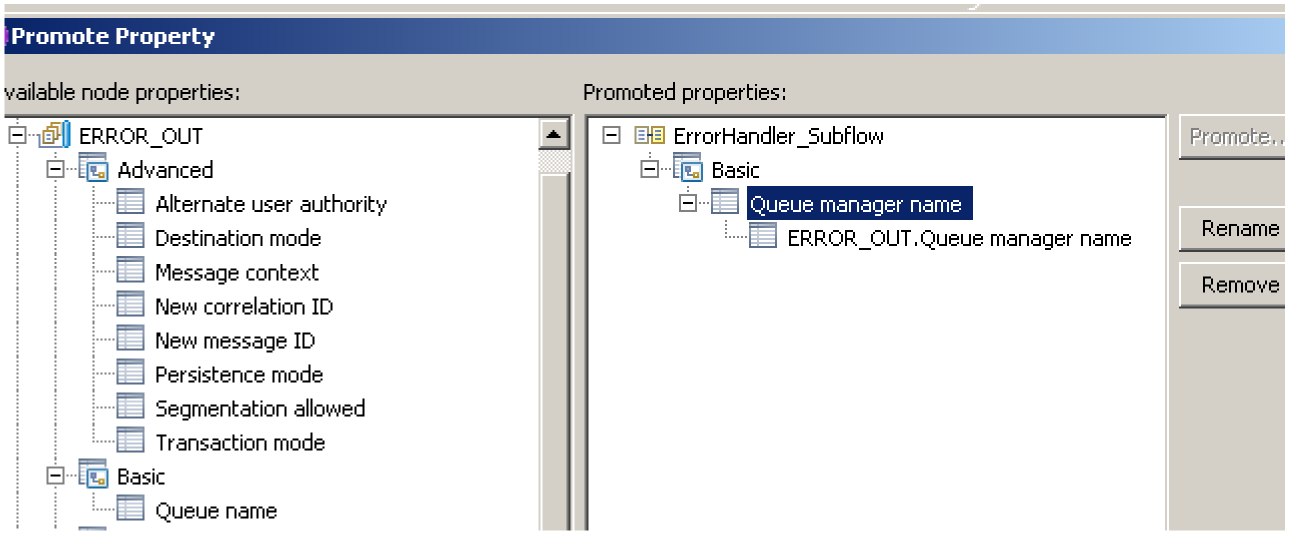
__ 4. Promote the **Subflow_Version** property that you just added, along with the **Queue name** and **Queue manager name** properties of the WebSphere MQ Output node in ErrorHandler_Subflow.

__ a. Click the **Graph** tab at the bottom of the Message Flow editor window to display the message flow diagram.

__ b. Right-click the **ERROR_OUT MQ** Output node, and then select **Promote Property** from the menu.

__ c. In the **Available node properties** window, under **Basic**, select **Queue manager name**, and then click **Promote**. The **Target Selection** menu is displayed.

__ d. Select **ErrorHandler_Subflow**, and then click **OK**. The **Queue manager name** property is shown in the **Promoted properties** window under the group **Basic**. If you expand the property, you see the property name, prefixed by the node label (ERROR_OUT).



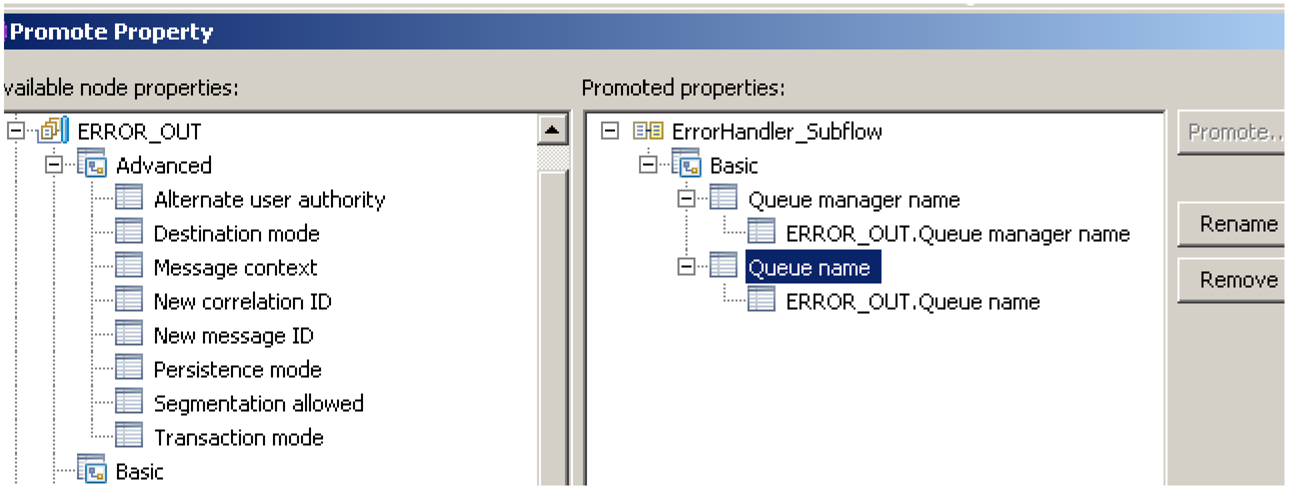
Note

Remember that if two or more properties within a message flow have the same name, you probably want to assign them to separate groups. If you do not do so, then a change to the value of the promoted property changes the value of all of the properties by that name within that group. For example, the Queue name property is shown as a property to all WebSphere MQ Input and WebSphere MQ Output nodes. If you promote one Queue name property, changing that promoted value changes all of the values of the Queue name property that are within that group.

__ e. From the **Available node properties** window, select **Queue name**, and then click **Promote**. The **Target Selection** menu is displayed.

__ f. Under **ErrorHandler_Subflow**, select **Basic**, and then click **OK**.

The **Queue name** property is shown in the **Promoted properties** window. If you expand the property, you see the property name, prefixed by the node name.



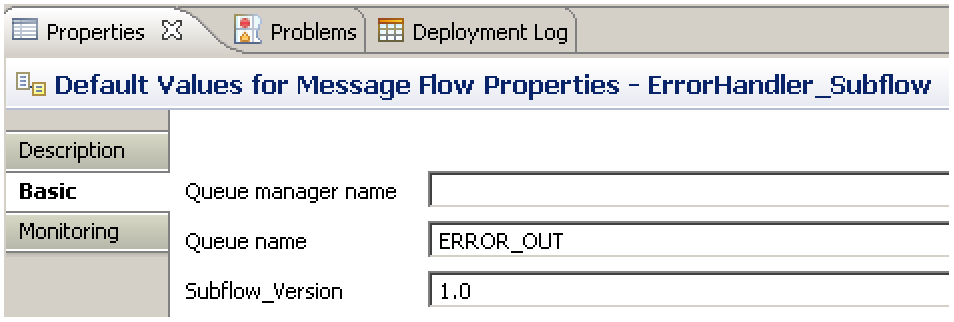
__ g. In the **Promote Property** window, click **OK**.

The Properties for the WebSphere MQ Output node are displayed. Observe that the **Queue manager name** and **Queue name** values now are shown as **Promoted to flow**. This display indicates that you cannot change those values here at the “node” level; you must change them at the “flow” level.

__ 5. Review the promoted properties.

__ a. Right-click in the white space in the Message Flow editor drawing canvas, and then select **Properties** from the menu.

The **Queue manager name** and **Queue name** properties that you promoted in the previous steps are displayed, as is the **Subflow_Version** user-defined property. Although you are not changing the values as this time, this location is where you would do so.



Default Values for Message Flow Properties - ErrorHandler_Subflow

Queue manager name

Queue name ERROR_OUT

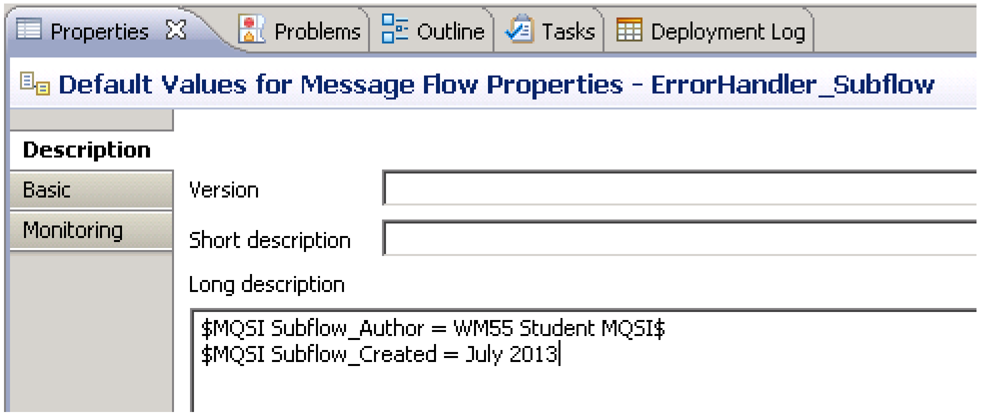
Subflow_Version 1.0

__ 6. Create more keywords for the subflow to be displayed in the deployed message flow.

__ a. In the **Properties** tab of the **ErrorHandler_Subflow**, click the **Description** tab.

__ b. In **Long Description**, enter the following lines:

```
$MQSI Subflow_Author = WM665 Student MQSI$
$MQSI Subflow_Created = July 2013 MQSI$
```



Default Values for Message Flow Properties - ErrorHandler_Subflow

Version

Short description

Long description

```
$MQSI Subflow_Author = WM55 Student MQSI$
$MQSI Subflow_Created = July 2013
```

This action defines two new keywords, **Subflow_Author** and **Subflow_Created**, and assigns values to them. You can use different values from those values that are shown here if you want.

These values are displayed when you examine the properties of the deployed subflow.

__ 7. Review the runtime environment information that the subflow uses. The Compute node (Capture Error) ESQL code already references a number of the integration node properties that are available.

__ a. In the Message Flow editor, double-click the **Capture Error** Compute node. The EQSL for the node opens in the editor.

__ b. Locate the `SET ID =` statement.

```
SET ID = BrokerName || '/' ||
         ExecutionGroupLabel || '/' ||
         MessageFlowLabel;
```

It concatenates the integration node name (BrokerName), the label for the integration server (ExecutionGroupLabel), and label for the message flow, as you saw in the exercise where you tested the subflow.

Several other runtime properties are accessible from ESQL and Java. For more information, see the IBM Integration Bus Information Center.

Note

Runtime property names are case-sensitive.

__ 8. Save the workspace (**Ctrl + S**).

__ 9. Close the EQSL editor.

Part 3: Create and deploy the BAR file

In this step, you build and deploy a broker archive file manually so that you can examine the promoted properties.

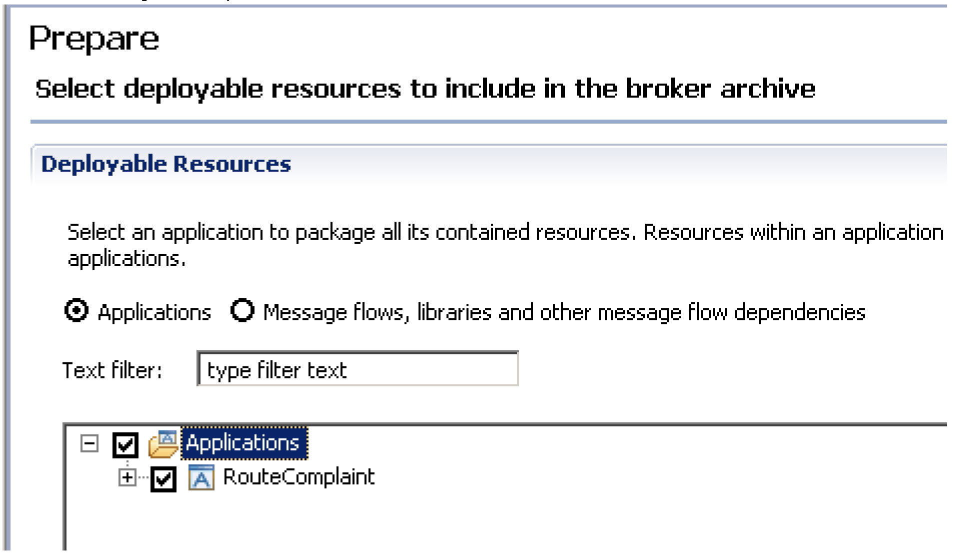
__ 1. Create a BAR file.

__ a. In the Integration Development perspective, select **File > New > BAR file**. The New Bar File windows opens.

__ b. Leave **Container** and **Folder** set to their default values. For **Name**, enter: `UserProperties`

__ c. Click **Finish**. The BAR file is created in the **BARfiles** project.

__ d. In the BAR file editor, under Deployable Resources, click **Applications** (or **RouteComplaint**).



Prepare

Select deployable resources to include in the broker archive

Deployable Resources

Select an application to package all its contained resources. Resources within an application applications.

[✓ Applications] Message flows, libraries and other message flow dependencies

Text filter: type filter text

☑ Applications
☑ RouteComplaint

__ e. In the upper-right area of the editor window, click **Build And Save**. The components are added to the BAR file.

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

__ 2. Deploy the BAR file to the integration server.

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

__ b. Right-click **UserProperties.bar > BARfile/UserProperties.bar** and then select **Deploy** from the menu. The Deploy menu opens.

__ c. Select **default**, integration server and then click **Finish**.

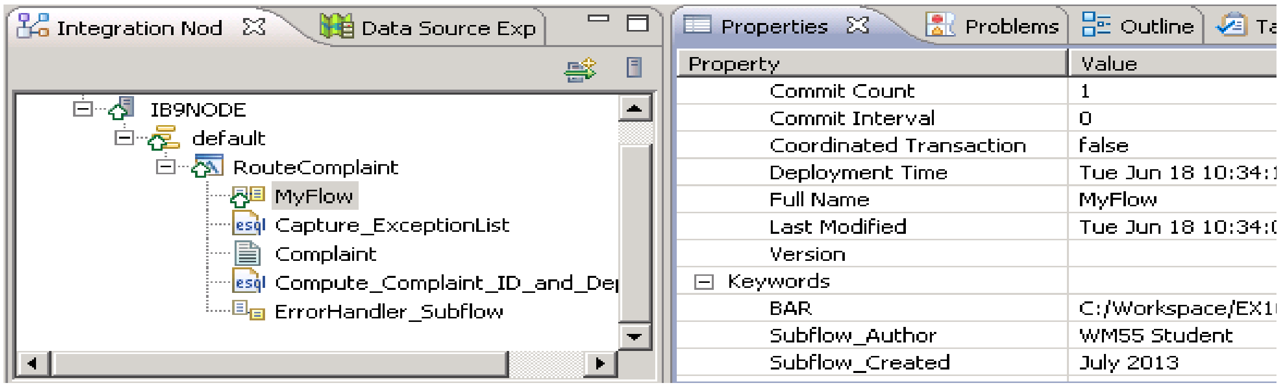
Part 4: Display the user-defined properties

__ 1. Review the BAR file attributes.

__ a. In the Integration Nodes view, expand **default > Route Complaint**.

__ b. Select the message flow **MyFlow**.

__ c. Verify that the Properties view contains the `Subflow Author` and `Subflow_Created` keywords that you defined earlier in the exercise.



Property	Value
Commit Count	1
Commit Interval	0
Coordinated Transaction	false
Deployment Time	Tue Jun 18 10:34:1
Full Name	MyFlow
Last Modified	Tue Jun 18 10:34:0
Version	
Keywords	
BAR	C:/Workspace/EX1
Subflow_Author	WM55 Student
Subflow_Created	July 2013

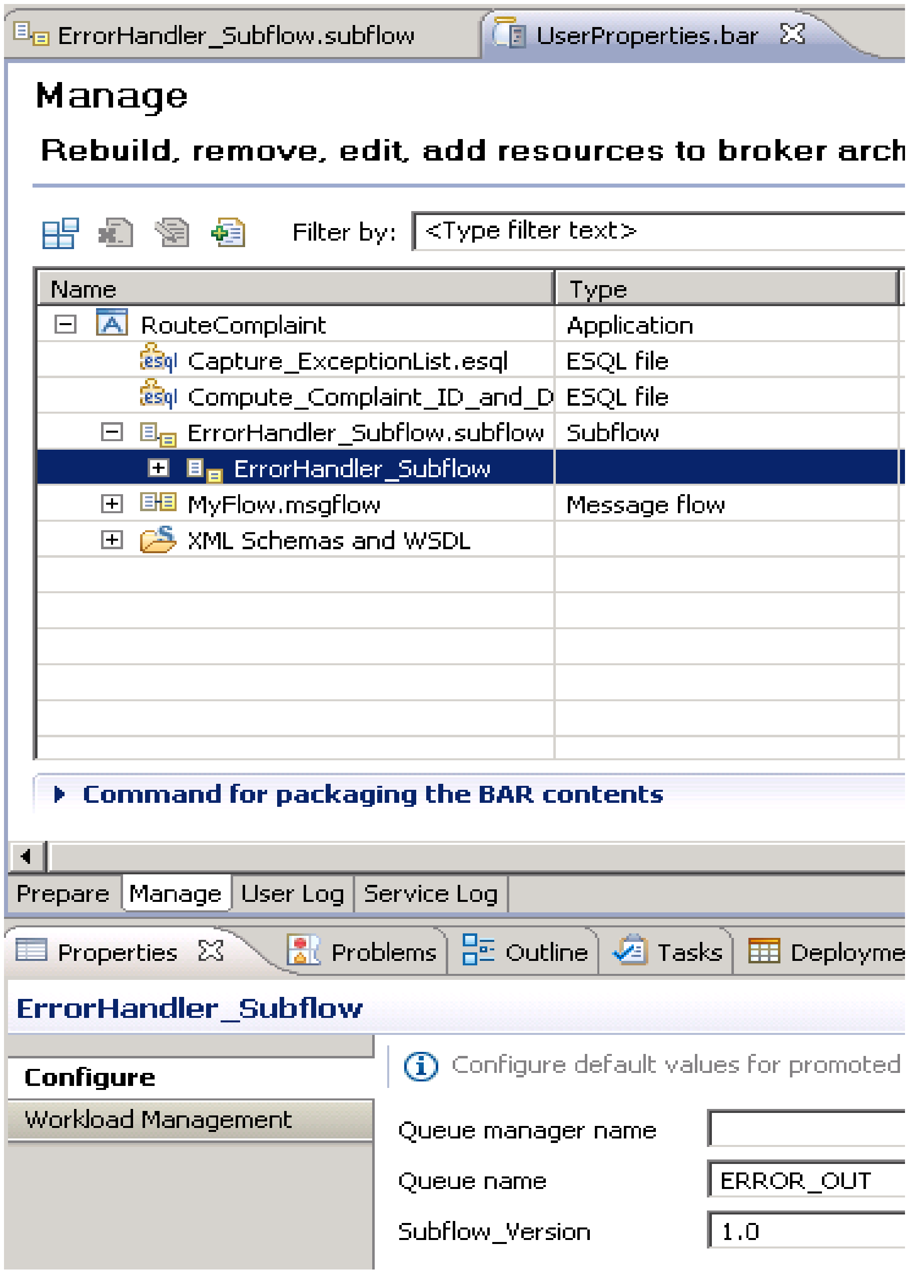
__ 2. Check the subflow user-defined properties and promoted properties.

__ a. Select the **Manage** tab on the BAR file editor for **UserProperties.bar** file.

__ b. Expand **ErrorHandler_Subflow.subflow**.

__ c. Select **ErrorHandler_Subflow**.

__ d. Verify that the **Properties** view contains the **Queue manager name** and **Queue name** properties that you promoted and the **Subflow_Version** property that you created.



Name	Type
RouteComplaint	Application
Capture_ExceptionList.esql	ESQL file
Compute_Complaint_ID_and_D	ESQL file
ErrorHandler_Subflow.subflow	Subflow
ErrorHandler_Subflow	
MyFlow.msgflow	Message flow
XML Schemas and WSDL	

Queue manager name	
Queue name	ERROR_OUT
Subflow_Version	1.0

Part 5: Clean up the environment

__ 1. Close all open editors.

__ 2. In the **Integration Nodes** view, right-click the **default** integration server and click **Delete > All Flows and Resources**.

End of exercise

Appendix A. Understanding ESQL statements

What this exercise is about

This optional exercise provides an opportunity to gain some familiarity with ESQL code that is used in the Filter and Compute nodes.

What you should be able to do

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

- Explain the basic syntax of ESQL
- List correlation names for messages
- Distinguish between input and output messages
- Explain index subscripts and the cardinality function
- Declare, move, and use dynamic field references (message cursor)
- Create new message fields

Requirements

You need these instructions and a pen.

Exercise instructions

Part 1: Complete the following ESQL statements

You can try completing this section without hints, or use the “missing parts table”. Review your student notes for ESQL syntax, or use the information center.

__ 1. Copy the entire input message to the output message.

```
SET ___________= _________________;
```

__ 2. Copy the body of the input message to the output message (which is also in the XML domain).

```
SET _____________________= ___________________________;
```

__ 3. Delete (omit) the Customer structure from the output message.

```
SET _____________.XMLNSC.OrderMsg.Customer = _____________;
```

__ 4. Output the last name in uppercase.

```
SET __________________________.OrderMsg.Customer.LastName =
_______________(_____________________.OrderMsg.Customer.LastName
);
```

__ 5. Delete any trailing spaces from the Customer Nr field.

```
SET OutputRoot.XMLNSC.OrderMsg.Customer.Nr =
__________(___________ ' ' FROM InputBody.OrderMsg.Customer.Nr);
```

__ 6. Append a new field (NewField) at the message body.

Hint

The XML specification requires exactly one root tag, which, in this case, is `<OrderMsg>`. Thus, you must append `NewField` below the `<OrderMsg>` tag.

```
SET _______________________________________________.NewField ='any
text';
```

__ 7. Create an output message in which the top-level XML element (the XML root tag) is named `<CustMsg>` instead of `<OrderMsg>`. The rest of the message is the same as in the input message.

```
SET OutputRoot.____________._____________= __________.______;
```

__ 8. Test (in a Filter node) if the message came in from queue “OrderInQ”.

Hint

This information is contained in the MQMD, in the **SourceQueue** field.

```
______ _________.MQMD.SourceQueue = 'OrderInQ';
```

__ 9. Test (in a Filter node) if the customer has a hyphenated LastName (like Miller-Smith).

```
RETURN ___________.OrderMsg.Customer.LastName ________’%-%’;
```

__ 10. In the output message, double the Price for the first OrderItem.

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1].Price =
2*__________(InputBody.OrderMsg.OrderItem[1].Price
________________);
```

__ 11. Set a new field DeliveryDate to three days later than OrderData.Date.

```
SET OutputRoot.XMLNSC.OrderMsg.DeliveryDate = _______
(_________.OrderMsg.OrderData.Date ___________) +
__________'3' _____;
```

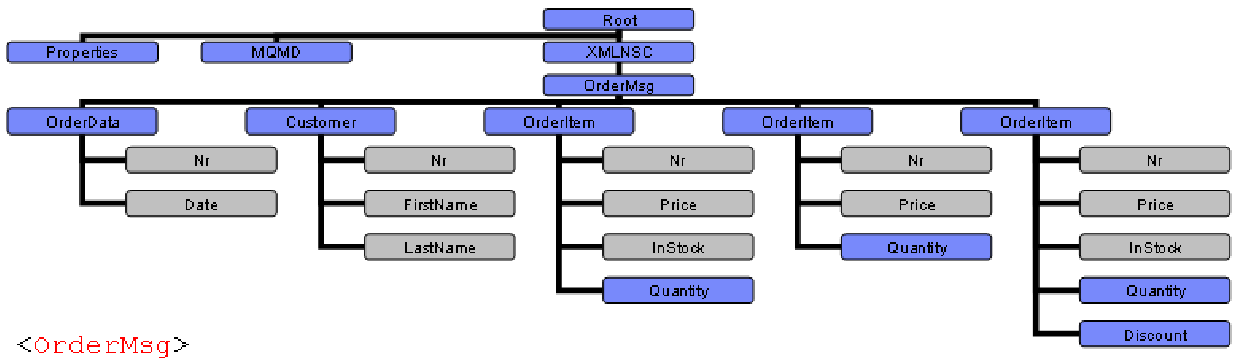
__ 12. Delete (omit) the last OrderItem structure from the output message.

```
SET _____________________________.OrderMsg.OrderItem_____ =
_________;
```

Here are the missing parts of ESQL:

[<]	InputRoot	OutputRoot.XMLNSC
AS DATE	InputRoot	OutputRoot.XMLNSC
AS DECIMAL	InputRoot	OutputRoot.XMLNSC
Body	INTERVAL	OutputRoot.XMLNSC.OrderMsg
CAST	LIKE	RETURN
CAST	NULL	Root
CustMsg	NULL	TRAILING
DAY	OrderMsg	TRIM
InputBody	OutputRoot	UPPER
InputBody	OutputRoot	XMLNSC

Part 2: What is the value of V after the following ESQL statements?



```
<OrderMsg>
   <OrderData Nr="300524" Date="2001-11-05"/>
   <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
   <OrderItem Nr="111" Price="27.95" InStock="100">
        <Quantity>2</Quantity></OrderItem>
   <OrderItem Nr="222" Price="8765.12">
        <Quantity>2</Quantity></OrderItem>
   <OrderItem Nr="333" Price="3.75" InStock="456">
        <Quantity>10</Quantity>
        <Discount>3</Discount></OrderItem></OrderMsg>
```

__ 1. SET V = InputBody.OrderMsg.OrderItem[2];

__ 2. SET V = InputBody.OrderMsg.OrderItem.*[2];

__ 3. SET V = InputBody.OrderMsg.OrderItem[<].*[<3];

__ 4. SET V = CARDINALITY(InputBody.OrderMsg.OrderItem);

__ 5. SET V = CARDINALITY(InputBody.OrderMsg.OrderItem[]);

__ 6. SET V = CARDINALITY(InputBody.OrderMsg.*[]);

__ 7. SET V = CARDINALITY(InputBody.OrderMsg.*);

Part 3: How would the message body look after the following ESQL statements?

__ 1. SET OutputRoot=InputRoot;

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1] = NULL;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2] = NULL;
```

Hint

Think about the order in which ESQL statements are executed.

__ 2. SET OutputRoot=InputRoot;

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2] = NULL;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1] = NULL;
```

Part 4: Reference variables

__ 1. What is the value of V after each of the following ESQL statements?

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderData;
SET V=FIELDNAME(myRef);  ______________________________
MOVE myRef NEXTSIBLING NAME 'OrderItem';
SET V=FIELDNAME(myRef); _____________________________
MOVE myRef LASTCHILD TYPE XMLNSC.Attribute;
SET V=FIELDNAME(myRef); ______________________________
```

__ 2. What is the value of V after each of the following ESQL statements?

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderItem.Discount;
SET V=FIELDNAME(myRef); _______________________________
MOVE myRef PARENT;
SET V=FIELDNAME(myRef); _______________________________
MOVE myRef FIRSTCHILD;
SET V=FIELDNAME(myRef); ________________________________
```

__ 3. What is the value of V after each of the following ESQL statements, and how will the resulting message look?

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderItem[<];
SET V=FIELDNAME(myRef); _______________________________
CREATE PREVIOUSSIBLING of myRef TYPE Name NAME 'OrderItemNew'
;
SET V=FIELDNAME(myRef); ______________________________
CREATE FIRSTCHILD of myRef FROM
OutputRoot.XMLNSC.OrderMsg.OrderItem.Nr;
SET V=FIELDNAME(myRef); _____________________________________
```

__ 4. Copy the entire message, and create a single new field (tag) `WorkDate` (value: current date) ***before*** the first `OrderItem`.

Hint

Use REFERENCE variables and a CREATE statement.

__ 5. Reordering output fields: Create an output message where the Order data is moved ***after*** the Customer data. You can achieve this result in various ways. Try to find the alternatives. You do not have to write all ESQL statements, but it is a good idea to outline your ESQL program.

Hint

You can use dynamic field references with DETACH / ATTACH or CREATE, SELECT, or build a new message body in the right order.

End of exercise

Solutions

Part 1, "Complete the following ESQL statements"

You can try to complete this section without hints, or use the “missing parts table”. Review your student notes for ESQL syntax, or use the information center.

__ 1. Copy the entire input message to the output message.

```
SET ___________= __________________;
```
`SET OutputRoot = InputRoot;`

__ 2. Copy the body of the input message to the output message (which is also in the XML domain).

```
SET ______________________= ____________________________;
```
`SET OutputRoot.XMLNSC = InputBody;`

__ 3. Delete (omit) the Customer structure from the output message.

```
SET ______________.XMLNSC.OrderMsg.Customer = ______________;
```
`SET OutputRoot.XMLNSC.OrderMsg.Customer = NULL;`

__ 4. Output the last name in uppercase.

```
SET __________________________.OrderMsg.Customer.LastName =
________________(______________________.OrderMsg.Customer.LastName
);
```
`SET OutputRoot.XMLNSC.OrderMsg.Customer.LastName =`
`UPPER(InputBody.OrderMsg.Customer.LastName);`

__ 5. Delete any trailing spaces from the Customer Nr field.

```
SET OutputRoot.XMLNSC.OrderMsg.Customer.Nr =
__________(___________ ' 'FROM InputBody.OrderMsg.Customer.Nr);
```
`SET OutputRoot.XMLNSC.OrderMsg.Customer.Nr = TRIM(TRAILING ' '`
`FROM InputBody.OrderMsg.Customer.Nr);`

__ 6. Append a new field (NewField) at the message body.

Hint

The XML specification requires exactly one root tag, which is in this case `<OrderMsg>`. Thus, you must append NewField below the `<OrderMsg>` tag.

```
SET ________________________________________________.NewField =’any
text’;
```
`SET OutputRoot.XMLNSC.OrderMsg.NewField = 'any text';`

__ 7. Create an output message in which the top-level XML element (the XML root tag) is named `<CustMsg>` instead of `<OrderMsg>`. The rest of the message is the same as in the input message.

```
SET OutputRoot.___________.____________= _________._____;
SET OutputRoot.XMLNSC.CustMsg = InputBody.OrderMsg;
```

__ 8. Test (in a Filter node) if the message came in from queue “OrderInQ”.

Hint: This information is contained in the MQMD, in field SourceQueue.

```
______ _________.MQMD.SourceQueue = 'OrderInQ';
RETURN Root.MQMD.SourceQueue = 'OrderInQ';
```

__ 9. Test (in a Filter node) if the customer has a hyphenated LastName (like Miller-Smith).

```
RETURN ___________.OrderMsg.Customer.LastName ________'%-%';
RETURN Body.OrderMsg.Customer.LastName LIKE '%-%';
```

__ 10. In the output message, double the Price for the first OrderItem.

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1].Price =
2*__________(InputBody.OrderMsg.OrderItem[1].Price
________________);
```

You must explicitly **cast** to DECIMAL for the calculation, but the XML parser (OutputRoot.XML) does an implicit cast to character.

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1].Price =
2*CAST(InputBody.OrderItem[1].Price AS DECIMAL);
```

__ 11. Set a new field DeliveryDate to three days later than OrderData.Date.

```
SET OutputRoot.XMLNSC.OrderMsg.DeliveryDate = _______
(_________.OrderMsg.OrderData.Date ___________) +
__________'3' _____;
SET OutputRoot.XMLNSC.OrderMsg.DeliveryDate =
CAST(InputBody.OrderMsg.OrderData.Date AS DATE) + INTERVAL '3'
DAY;
```

__ 12. Delete (omit) the last OrderItem structure from the output message.

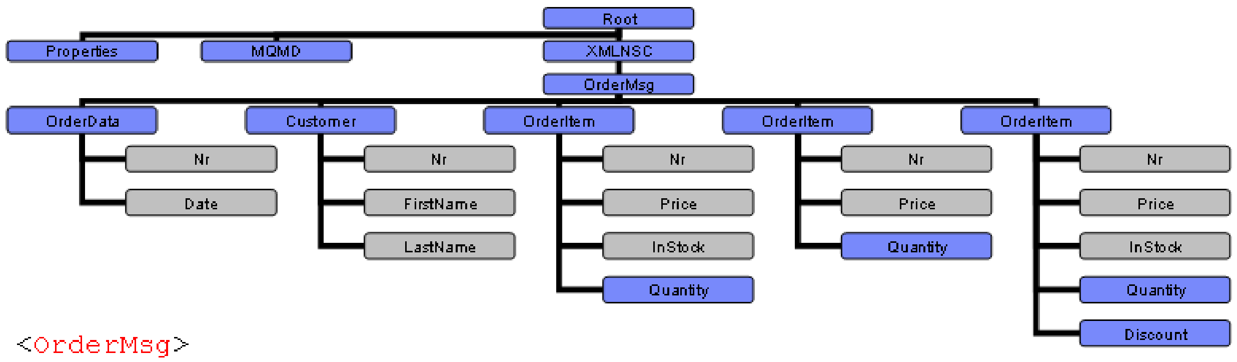
```
SET _____________________________.OrderMsg.OrderItem_____ =
_________;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[<] = NULL;
```

Here are the missing parts of ESQL:

[<]	InputBody	OutputRoot.XMLNSC
AS DATE	InputBody	OutputRoot.XMLNSC
AS DECIMAL	InputBody	OutputRoot.XMLNSC

Body	INTERVAL	OutputRoot.XMLNSC.OrderMsg
CAST	LIKE	RETURN
CAST	NULL	Root
CustMsg	NULL	TRAILING
DAY	OrderMsg	TRIM
InputBody	OutputRoot	UPPER
InputBody	OutputRoot	XMLNSC

Part 2, "What is the value of V after the following ESQL statements?"



```
<OrderMsg>
   <OrderData Nr="300524" Date="2001-11-05"/>
   <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
   <OrderItem Nr="111" Price="27.95" InStock="100">
        <Quantity>2</Quantity></OrderItem>
   <OrderItem Nr="222" Price="8765.12">
        <Quantity>2</Quantity></OrderItem>
   <OrderItem Nr="333" Price="3.75" InStock="456">
        <Quantity>10</Quantity>
        <Discount>3</Discount></OrderItem></OrderMsg>
```

__ 1. `SET V = InputBody.OrderMsg.OrderItem[2];`

This statement fails because the right side of the SET statement points to a structure, which cannot be assigned to a normal ESQL variable, which is a scalar. However, assigning a structure to a message field is valid:

```
SET
OutputRoot.XMLNSC.OrderMsg.NewField=InputBody.OrderMsg.OrderI
tem[2];
```

__ 2. `SET V = InputBody.OrderMsg.OrderItem.*[2];`

'27.95'

__ 3. `SET V = InputBody.OrderMsg.OrderItem[<].*[<3];`

'456'

__ 4. `SET V = CARDINALITY(InputBody.OrderMsg.OrderItem);`

This statement does not deploy. CARDINALITY needs a list (field array) as its argument.

__ 5. `SET V = CARDINALITY(InputBody.OrderMsg.OrderItem[]);`

3

__ 6. `SET V = CARDINALITY(InputBody.OrderMsg.*[]);`

5

__ 7. `SET V = CARDINALITY(InputBody.OrderMsg.*);`

This statement does not deploy. Cardinality needs a list (field array) as its argument. And, to be syntactically correct, the list must be expressed with brackets (**`[]`**).

Part 3, "How would the message body look after the following ESQL statements?"

__ 1. `SET OutputRoot=InputRoot;`

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1] = NULL;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2] = NULL;
```

Hint

Think about the order in which ESQL statements are executed.

```
<OrderMsg>
   <OrderData Nr="300524" Date="2001-11-05"/>
   <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
   <OrderItem Nr="222" Price="8765.12">
      <Quantity>2</Quantity></OrderItem></OrderMsg>
```

This result might not be what you expected. Because repeating elements are stored on a stack, when you take one above, the next takes its place. Thus, you should delete repeating elements in reverse order, as in the next question:

__ 2. `SET OutputRoot=InputRoot;`

```
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[2] = NULL;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[1] = NULL;
```

```
   <OrderMsg>
      <OrderData Nr="300524" Date="2001-11-05"/>
      <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
      <OrderItem Nr="333" Price="3.75" InStock="456">
         <Quantity>10</Quantity>
         <Discount>3</Discount></OrderItem></OrderMsg>
```

Part 4, "Reference variables"

__ 1. What is the value of V after each of the following ESQL statements?

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderData;
SET V=FIELDNAME(myRef);
______________________________'OrderData'
MOVE myRef NEXTSIBLING NAME 'OrderItem';
SET V=FIELDNAME(myRef); _____________________________
'OrderItem'
MOVE myRef LASTCHILD TYPE XMLNSC.Attribute;
SET V=FIELDNAME(myRef);
______________________________'InStock'
```

__ 2. What is the value of V after each of the following ESQL statements?

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderItem.Discount;
SET V=FIELDNAME(myRef);
____________________________'Root'
MOVE myRef PARENT;
SET V=FIELDNAME(myRef);
____________________________'Root'
MOVE myRef FIRSTCHILD;
SET V=FIELDNAME(myRef);
_____________________________'Properties'
```

`myRef` was declared for a nonexisting field, and thus points to `Root`. If a move was not successful (as in the first move here, Root has no parent), the dynamic reference stays fixed. No runtime errors occur. Use the LASTMOVE function to check the success of the move.

__ 3. What is the value of V after each of the following ESQL statements, and how will the resulting message look?

```
DECLARE V CHAR;
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderItem[<];
SET V=FIELDNAME(myRef);
_______________________________'OrderItem'
CREATE PREVIOUSSIBLING of myRef TYPE Name NAME 'OrderItemNew';
SET V=FIELDNAME(myRef);
______________________________'OrderItem'
CREATE FIRSTCHILD of myRef FROM
OutputRoot.XMLNSC.OrderMsg.OrderItem.Nr;
SET V=FIELDNAME(myRef);
______________________________'OrderItem'
```

The position of a reference variable stays fixed, even if the message tree is modified.

```
<OrderMsg>
  <OrderData Nr="300524" Date="2001-11-05"/>
  <Customer Nr="123456" FirstName="Andrew" LastName="Smith"/>
  <OrderItem Nr="111" Price="27.95" InStock="100">
    <Quantity>2</Quantity></OrderItem>
  <OrderItem Nr="222" Price="8765.12">
    <Quantity>2</Quantity></OrderItem>
  **<OrderItemNew/>**
  <OrderItem **Nr="111"** Nr="333" Price="3.75" InStock="456">
    <Quantity>10</Quantity>
    <Discount>3</Discount></OrderItem></OrderMsg>
```

__ 4. Copy the entire message, and create a single new field (tag) `WorkDate` (value: current date) ***before*** the first `OrderItem`.

Hint

Use REFERENCE variables and a CREATE statement.

```
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.Customer;
CREATE NEXTSIBLING OF myRef TYPE Name NAME 'WorkDate' VALUE
CURRENT_DATE;
```

__ 5. Reordering output fields: Create an output message where the Order data is moved ***after*** the Customer data. There are various ways to achieve this goal. Try to find the alternatives. You do not have to write all ESQL statements, but you should outline your ESQL program.

Hint

You can use dynamic field references with DETACH, ATTACH, or CREATE, SELECT; or build a new message body in the right order.

Alternative solution 1:

```
SET OutputRoot = InputRoot;
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.OrderData;
DETACH myRef;
ATTACH myRef TO OutputRoot.XMLNSC.OrderMsg.Customer AS
NEXTSIBLING;
```

Alternative solution 2 (this solution is almost the same as 1, but with CREATE instead of ATTACH):

```
SET OutputRoot = InputRoot;
SET OutputRoot.XMLNSC.OrderMsg.OrderData = NULL; /* delete
field */
DECLARE myRef REFERENCE TO
OutputRoot.XMLNSC.OrderMsg.Customer;
CREATE NEXTSIBLING OF myRef FROM InputBody.OrderMsg.OrderData;
```

Alternative solution 3 (this solution is cumbersome for long messages because you must build each field with a separate ESQL statement; however, it is better to use a Mappings node for it):

```
CALL ddCopyMessageHeaders();
SET OutputRoot.XMLNSC.OrderMsg.Customer =
InputBody.OrderMsg.Customer;
SET OutputRoot.XMLNSC.OrderMsg.OrderData =
InputBody.OrderMsg.OrderData;
SET OutputRoot.XMLNSC.OrderMsg.OrderItem[] =
InputBody.OrderMsg.OrderItem[];
```

END OF SOLUTION