16L70

IBM Integration Bus

MQ Flexible Topologies:
Configuring MQ nodes
using
MQ Connection Properties

Featuring:

- Running MQ applications on IIB Nodes with and without an associated queue manager
- Reuse of Graphical Data Maps stored in Shared Libraries
- Configuring MQ node connection properties in Toolkit
- IIB Flow Exerciser testing
- Bindings and Client connections to MQ queue managers

January 2016

Hands-on lab built at product
Version 10.0.0.3

1. Introduction

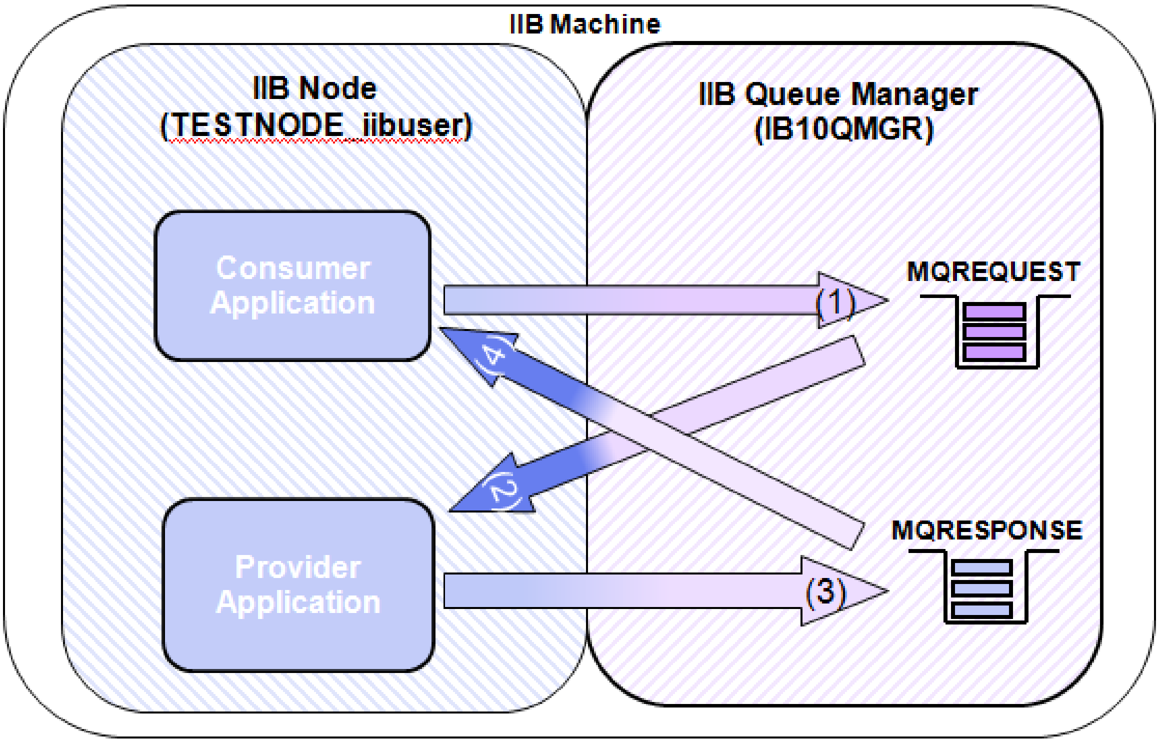
This lab guide covers the flexible MQ topologies available in IBM Integration Bus V10.

IBM Integration Bus V10 permits direct access to messages on any queue manager using:

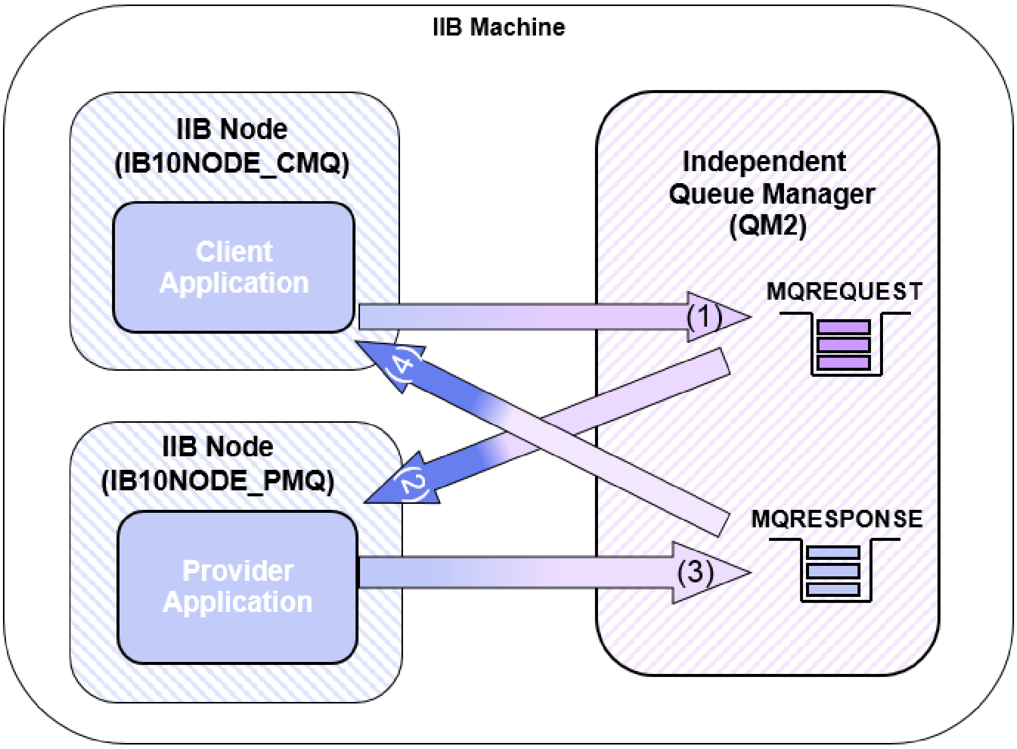
1) Local (server) bindings connections
2) Client connectivity through a local WebSphere MQ client installation
3) Client connectivity using a Client Connection Definition Table (CCDT).

In this lab guide you will configure two MQ applications to show how MQ queues can be accessed on queue managers where:

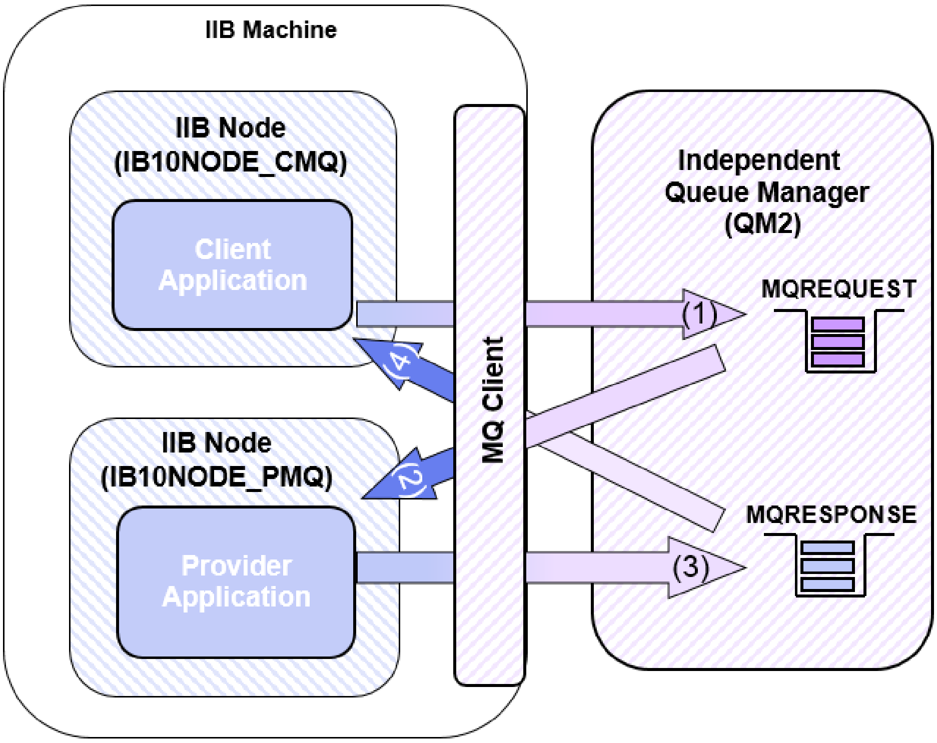
1) The queue manager is defined with the Integration Node (*you will cover this in **Scenario 1: Using an IIB node with an associated queue*** manager ***on page 19***):



2) The queue manager is local to the Integration Node (Bindings connection) *but not defined as the Integration Node queue manager*. You will cover this in ***Scenario 2: IIB node with no associated queue*** manager: Using an MQ SERVER connection ***on page 34 :***



3) The queue manager is remote to the Integration Node (MQ Client connection). You will cover this in ***Scenario 3: IIB node with no associated queue manager:*** Using an MQ CLIENT connection ***on page 45 :***



Important note

This lab, version 10.0.0.3, has been updated significantly from earlier versions. The following changes have been made:

You should use the Windows user "iibuser". This user is a member of mqbrkrs and mqm, but is not a member of Administrators. The user "iibuser" can create new IIB nodes and do all required IIB development work. However, installation of the IIB product requires Administrator privileges (not required in this lab).

The database has been changed from the DB2 SAMPLE database to the DB2 HRDB database. HRDB contains two tables, EMPLOYEE and DEPARTMENT. These tables have been populated with data required for this lab. (The DDL for the HRDB is available in the student10 folder; we intend to provide corresponding DDL for Microsoft SQL/Server and Oracle over time).

The map node now retrieves multiple rows from the database, using an SQL "LIKE" function . Additionally, the map has been refactored to use a main map and a submap. Both the main map and submap are located in a shared library.

Input to the integration service is now a simple schema containing just one element, the required employee number.

As a consequence, this version of the lab, and the associated solution, can only be used with the corresponding changes in other labs. Use version 10.0.0.3 of all labs in this series of lab guides.

1.1 The MQ based applications

Similar to the Web Services based JSON client and EmployeeService provided in other guides throughout this workshop, a Client and Provider application is supplied.

The Provider application obtains requests and provides responses using queues. The client is driven by providing JSON data to a URL controlled by an HTTP input node and provides a JSON based response over http.

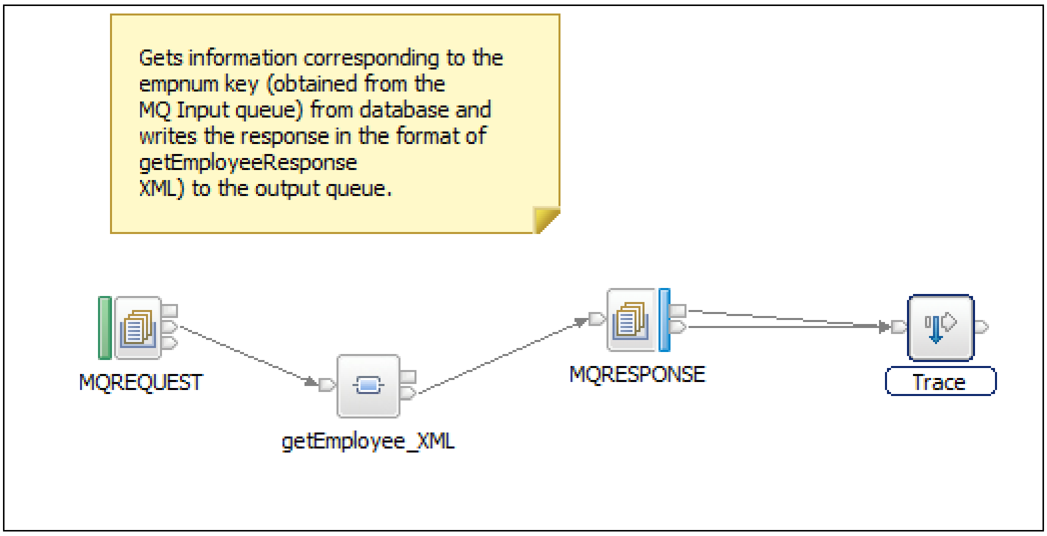
The following sections describe these applications in more detail.

NOTE: *The applications are provided purely to show (within the context of this workshop).*

1) The flexibility of MQ in IIB V10 and
2) The reuse of Maps stored in a Shared Library

The applications only work as expected when one user is submitting requests in a controlled way. A more complex Request/Response message correlation Pattern is available in the Patterns gallery, however is out of the scope of this lab guide.

1.1.1 The Provider application



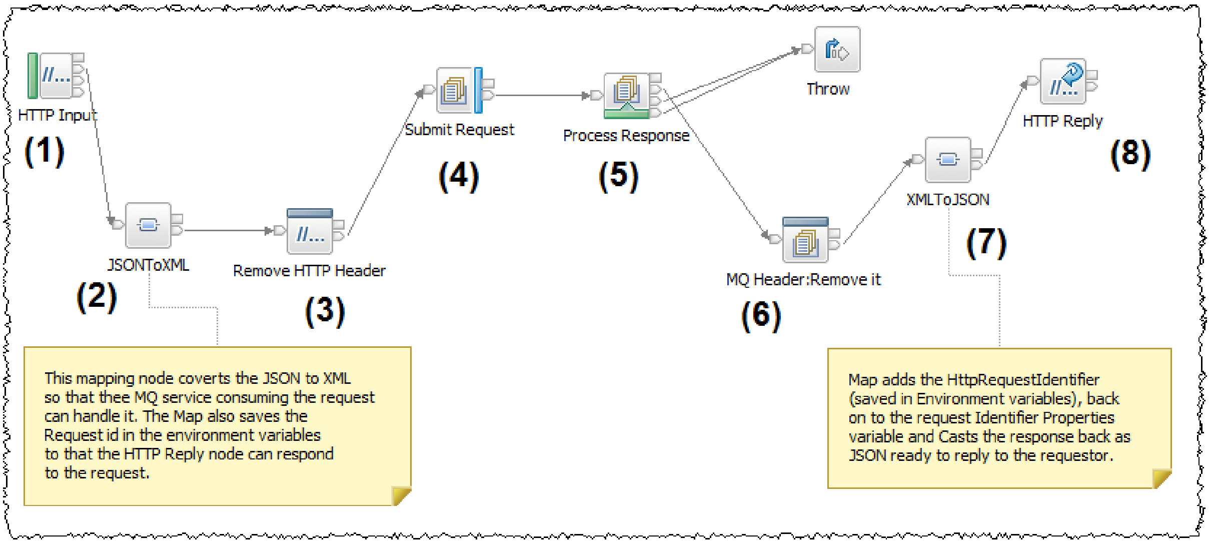
Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

The function of the EmployeeMQProvider application is to retrieve Employee details from the HRDB database.

It will use two queues (MQREQUEST and MQRESPONSE) to handle requests and provide the responses. Request data from MQREQUEST queue is passed to a mapping node, getEmployee_XML. The Mapping node uses XML data passed in the request to obtain details of the employee from the EMPLOYEE table in the database and provides XML Response data. The Response data is then written to a queue called MQRESPONSE.

The Mapping node is supplied in a Shared Library. The lab guide will demonstrate how to reuse assets previously created and stored in a Shared Library in IIB V10.

1.1.2 The Client application



This mapping node coverts the JSON to XML so that thee MQ service consuming the request can handle it. The Map also saves the Request id in the environment variables to that the HTTP Reply node can respond to the request.

Map adds the HttpRequestIdentifier (saved in Environment variables), back on to the request Identifier Properties variable and Casts the response back as JSON ready to reply to the requestor.

The EmployeeService_JSON_MQClient contains a message flow that:

1) Accepts a JSON request from an HTTP Input node
2) Converts the JSON to the required XML format for the EmployeeMQProvider to process the request using a mapping node. Note this node also demonstrates a new Graphical Data mapping feature in IIB V10 where it is possible to address the IIB Environment tree in the mapping node. The map saves the HTTP Request ID in Environment variables so that the HTTP Reply node works correctly after removing the HTTP headers from the Message tree in the scenario.
3) Removes the HTTP Headers
4) Writes the XML version of the request to the MQREQUEST queue
5) Waits for XML response data to appear on the MQRESPONSE queue
6) Removes the MQ headers from the Message Tree
7) Uses a second mapping node:
    a. Transforms the XML provided through the MQRESPONSE queue back to JSON
    b. Reinstates the data saved HTTP Request ID from the Environment Variables into the message tree so that the HTTP reply node can work correctly.
8) Provides the Response data back to the requestor as JSON data.

2. Complete the Provider application

Login to Windows as the user "iibuser", password = "passw0rd".

Start the IIB Toolkit from the Start menu.

The EmployeeMQProvider application is supplied with an empty message flow.

In this section you will complete the application by configuring the message flow to process the MQ messages written to the queue MQREQUEST by the MQ Client application.

You will then configure the message flow to reuse the "**getEmployee_XML**" map. This map invokes the getEmployees submap.

Response messages are written to the queue MQRESPONSE.

2.1 Configure Integration Bus node to work with DB2

If you have already done Lab 1 in this series (create an Integration Service), you can skip straight to "Prepare your workspace" on the next page.

To run this lab, the Integration Bus node must be enabled to allow a JDBC connection to the HRDB database.

1. Open an IIB Command Console (from the Start menu), and navigate to

```
c:\student10\Create_HR_database
```

2. Run the command

```
3_Create_JDBC_for_HRDB
```

   Accept the defaults presented in the script. This will create the required JDBC configurable service for the HRDB database.

3. Run the command

```
4_Create_HRDB_SecurityID
```

4. Stop and restart the node to enable the above definitions to be activated

```
mqsistop TESTNODE_iibuser
```

```
mqsistart TESTNODE_iibuser
```

This will create the necessary security credentials enabling TESTNODE_iibuser to connect to the database.

2.2 Prepare your workspace

1. To avoid naming clashes with earlier labs, this lab will be developed using a new workspace.

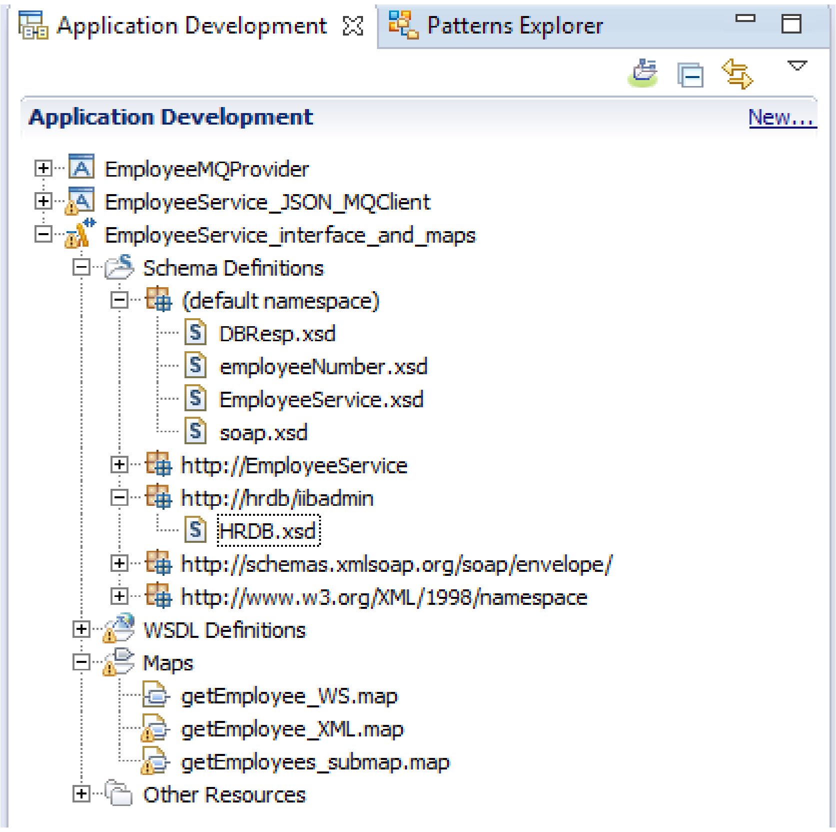
   If you already have a workspace open, click File, Switch Workspace. Give the new workspace the name

   c:\users\iibuser\IBM\IIB 10\workspace_MQTop

2. Import the PI file

   c:\student10\MQTopology\Resources\MQTopology_startingPoint.10.0.0.3.zip

   Import all projects in the PI file. Your workspace will be updated and look like this:

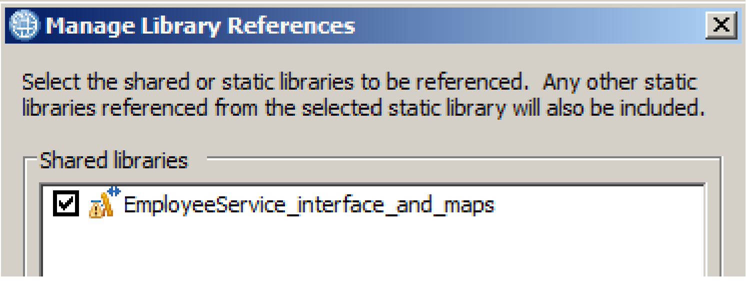


```
Application Development
  EmployeeMQProvider
  EmployeeService_JSON_MQClient
  EmployeeService_interface_and_maps
    Schema Definitions
      (default namespace)
        DBResp.xsd
        employeeNumber.xsd
        EmployeeService.xsd
        soap.xsd
      http://EmployeeService
      http://hrdb/iibadmin
        HRDB.xsd
      http://schemas.xmlsoap.org/soap/envelope/
      http://www.w3.org/XML/1998/namespace
    WSDL Definitions
    Maps
      getEmployee_WS.map
      getEmployee_XML.map
      getEmployees_submap.map
    Other Resources
```

3. Check the Library References.

   Right-click the EmployeeMQProvider application, and select Manage Library References.

   The library EmployeeService_interface_and_maps should be ticked. Click OK to close.



Manage Library References

Select the shared or static libraries to be referenced. Any other static libraries referenced from the selected static library will also be included.

Shared libraries

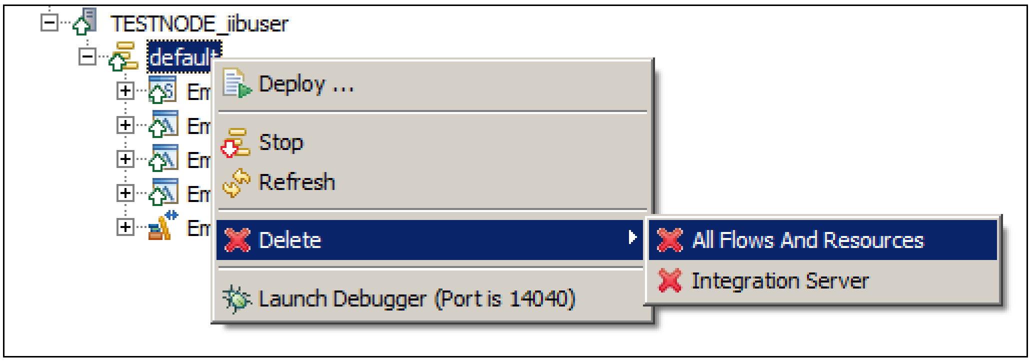
☑ EmployeeService_interface_and_maps

   Perform the same check on the application EmployeeService_JSON_MQClient.

2.3 Remove applications from TESTNODE

To avoid any conflicts with earlier scenarios, remove all deployed applications and other artefacts from the TESTNODE_iibuser node.

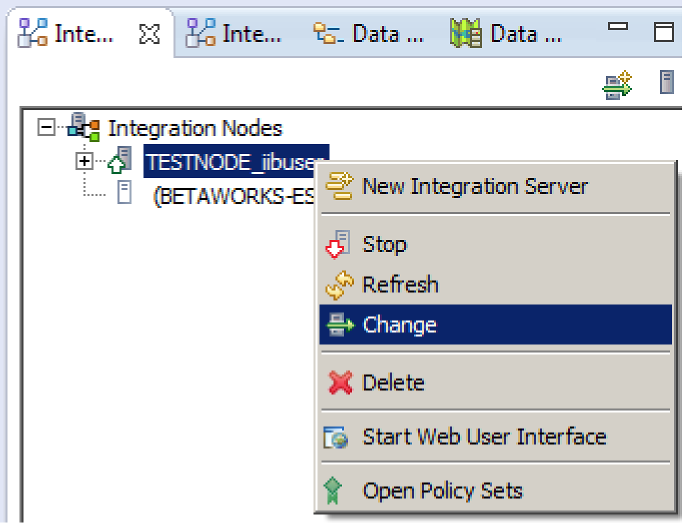
In the Integration Toolkit, in the Integration Nodes pane, right-click and select Delete, All Flows and Resources.



2.4 Set the Queue Manager for TESTNODE_iibuser

The TESTNODE_iibuser node was originally created automatically by IIB when the user iibuser opened the IIB Toolkit. This node was created without an associated queue manager. In this lab, the scenarios will be using MQ queues, so to access these queues using without specifying the name of the queue manager, the node has to be changed to have an associated queue manager.

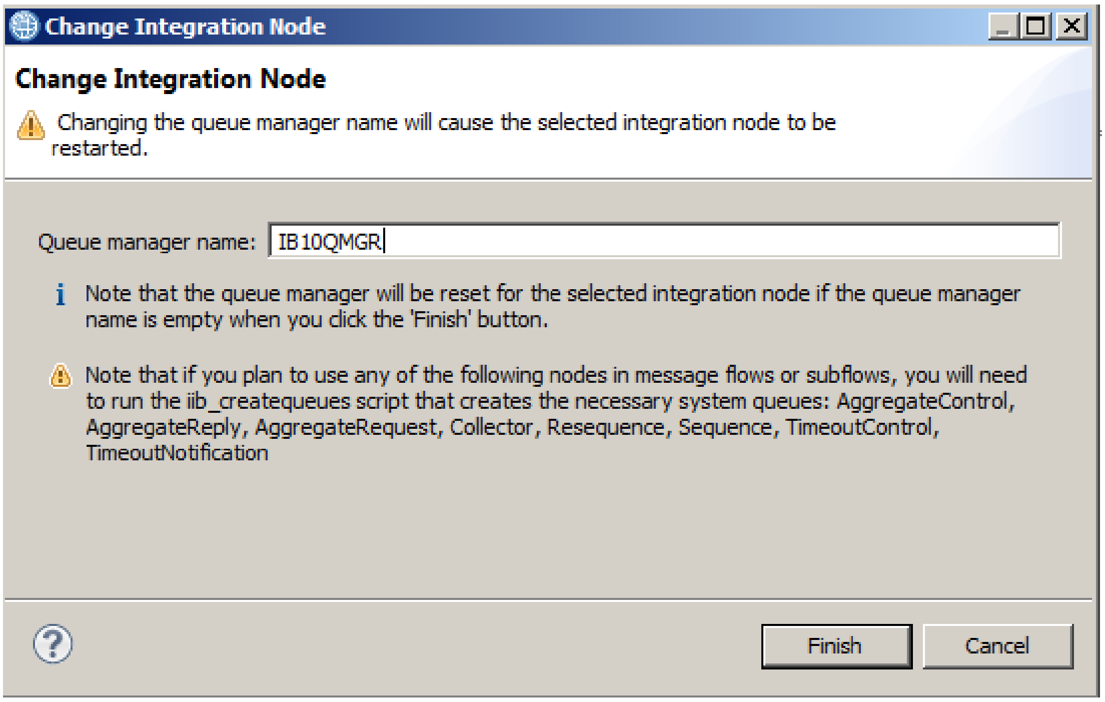
1. In the IIB Toolkit, in the Integration Nodes pane, right-click TESTNODE_iibuser, and select Change.



2. In the Change Integration Node window, specify the queue manager name IB10QMGR.

   Click Finish.

   This activity will stop and restart TESTNODE_iibuser.


Change Integration Node

Change Integration Node

Changing the queue manager name will cause the selected integration node to be restarted.

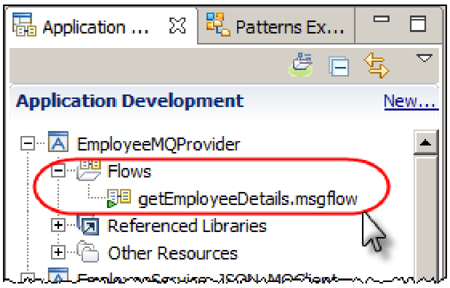
Queue manager name: IB10QMGR

Note that the queue manager will be reset for the selected integration node if the queue manager name is empty when you click the 'Finish' button.

Note that if you plan to use any of the following nodes in message flows or subflows, you will need to run the iib_createqueues script that creates the necessary system queues: AggregateControl, AggregateReply, AggregateRequest, Collector, Resequence, Sequence, TimeoutControl, TimeoutNotification

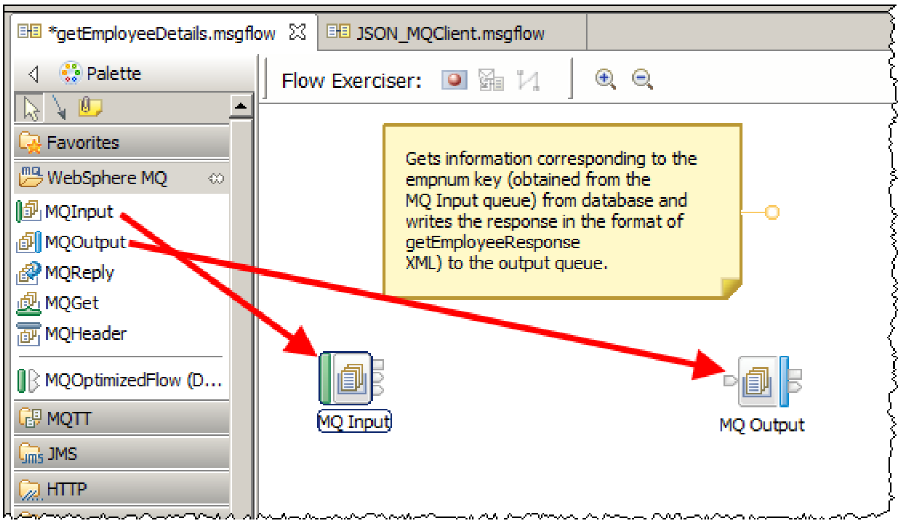
2.5 Configure the MQ Nodes

1. In the **Integration Toolkit,** expand the EmployeeMQProvider application and open the getEmployeeDetails message flow:



2. The message flow canvas has no nodes defined on it.

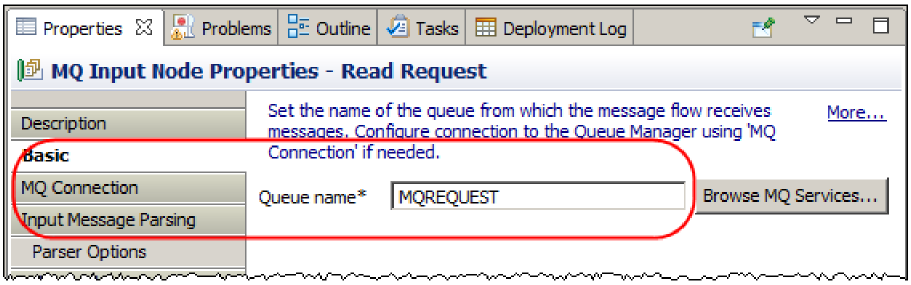
   In the Node Palette, open WebSphere MQ and add an MQInput Node then add MQOutput node as follows:


*getEmployeeDetails.msgflow
JSON_MQClient.msgflow
Palette
Flow Exerciser:
Favorites
WebSphere MQ
MQInput
MQOutput
MQReply
MQGet
MQHeader
MQOptimizedFlow (D...
MQTT
JMS
HTTP
Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.
MQ Input
MQ Output

3. Rename the MQInput node to “Read Request”

4. Rename the MQOutput node to “Write Response”.

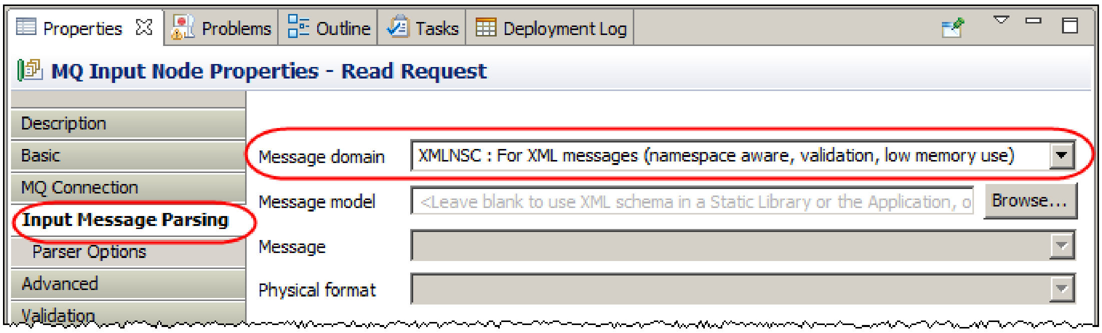
5. Click the MQInput node. In the Properties (Basic tab), specify a queue name MQREQUEST:



Set the name of the queue from which the message flow receives messages. Configure connection to the Queue Manager using 'MQ Connection' if needed.

Queue name* MQREQUEST

6. On the **Input Message Parsing** tab, change the Message domain to **XMLNSC**:



MQ Input Node Properties - Read Request

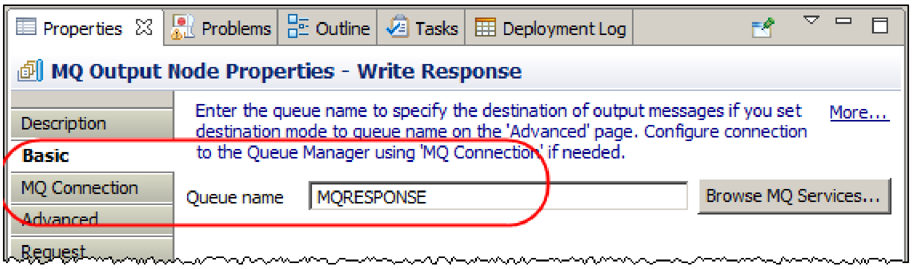
Message domain: XMLNSC : For XML messages (namespace aware, validation, low memory use)

Message model: <Leave blank to use XML schema in a Static Library or the Application, o

Message

Physical format

7. Click the **MQOutput** node and specify a queue name of "`MQRESPONSE`" on the Basic tab:

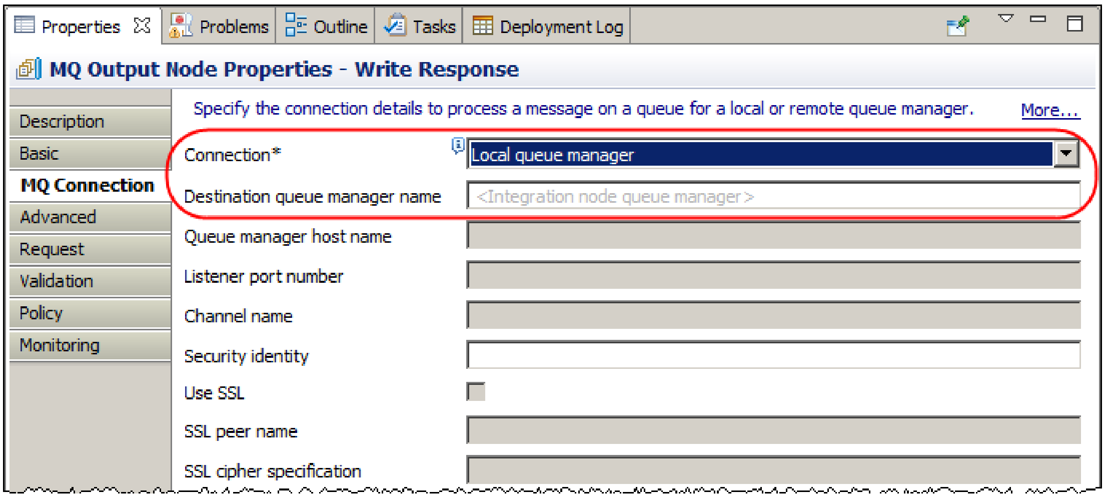


MQ Output Node Properties - Write Response

Enter the queue name to specify the destination of output messages if you set destination mode to queue name on the 'Advanced' page. Configure connection to the Queue Manager using 'MQ Connection' if needed.

Queue name: MQRESPONSE

8. Click the (***new in V10***) MQ Connection tab to show the MQ Connection properties available for the output node:



MQ Output Node Properties - Write Response

Specify the connection details to process a message on a queue for a local or remote queue manager.

Connection*: Local queue manager

Destination queue manager name: <Integration node queue manager>

Queue manager host name

Listener port number

Channel name

Security identity

Use SSL □

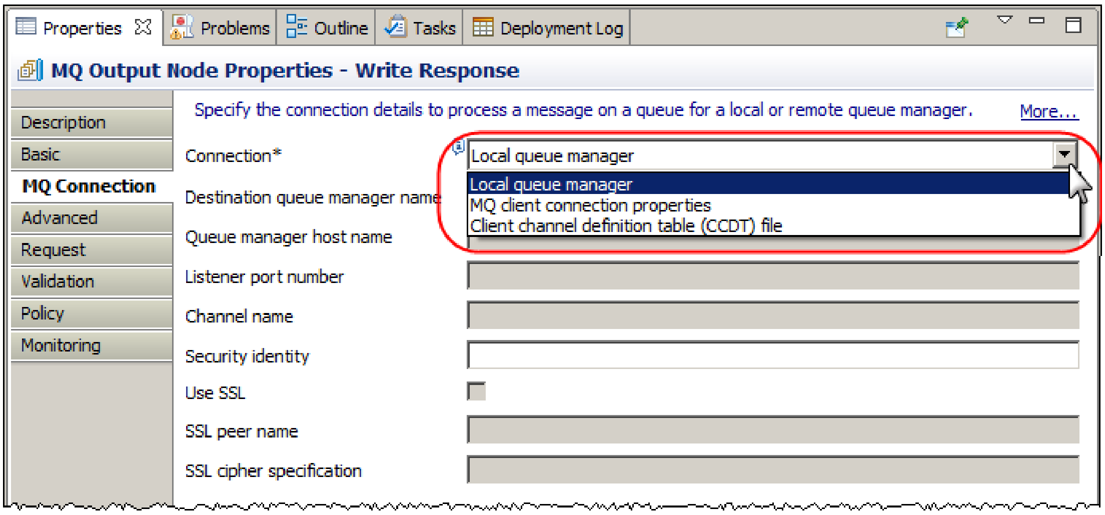
SSL peer name

SSL cipher specification

The default specification for the connection details for the queue is that messages will be written via the queue manager associated with the Integration Node.

Specifying "Local queue manager" option for "Connection" and no value in "*Destination queue manager name*" implies that the queue exists on the Integration Node queue manager.

9. Select "**Local queue manager**" to show the options available for the Connection:



MQ Output Node Properties - Write Response

Specify the connection details to process a message on a queue for a local or remote queue manager. More...

Connection* Local queue manager
- Local queue manager
- MQ client connection properties
- Client channel definition table (CCDT) file

Destination queue manager name

Queue manager host name

Listener port number

Channel name

Security identity

Use SSL □

SSL peer name

SSL cipher specification

With IIB V10 you can specify that the queue exists:

I. On the (local) queue manager defined with Integration Node (leave the "***Destination queue manager name***" blank)

II. On a local queue manager (specify the name of the queue manager in "***Destination queue manager name***")

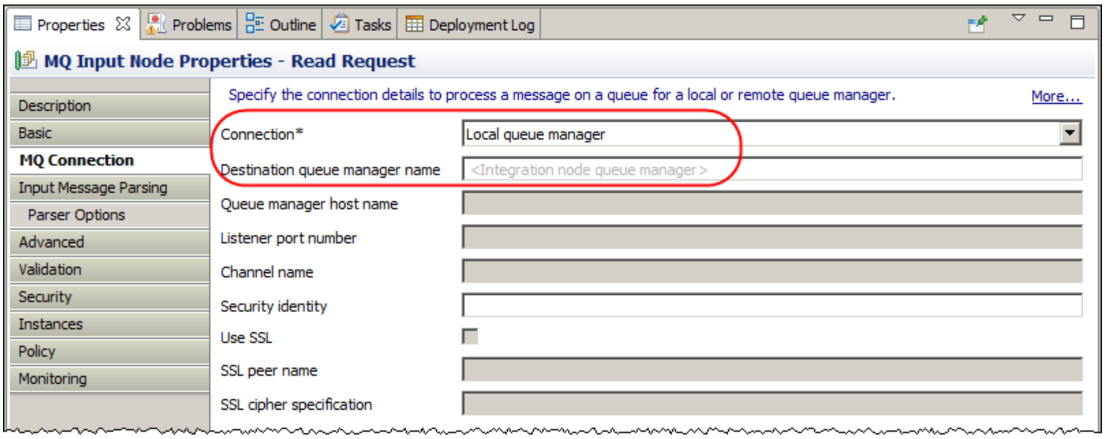
III. On a remote queue manager (client connection properties will be required)

IV. On a remote queue manager where connection properties are defined in a Client channel definition table (CCDT)

Leave **Connection** as "*Local queue manager*" (the default setting).

We will revisit the different connection options on this tab later in the lab guide.

10. Check the Connection details on the **MQInput**, called "**Read Request**", you will see the same default MQ Connection settings:



MQ Input Node Properties - Read Request

Specify the connection details to process a message on a queue for a local or remote queue manager. More...

Connection* Local queue manager

Destination queue manager name <Integration node queue manager>

Queue manager host name

Listener port number

Channel name

Security identity

Use SSL □

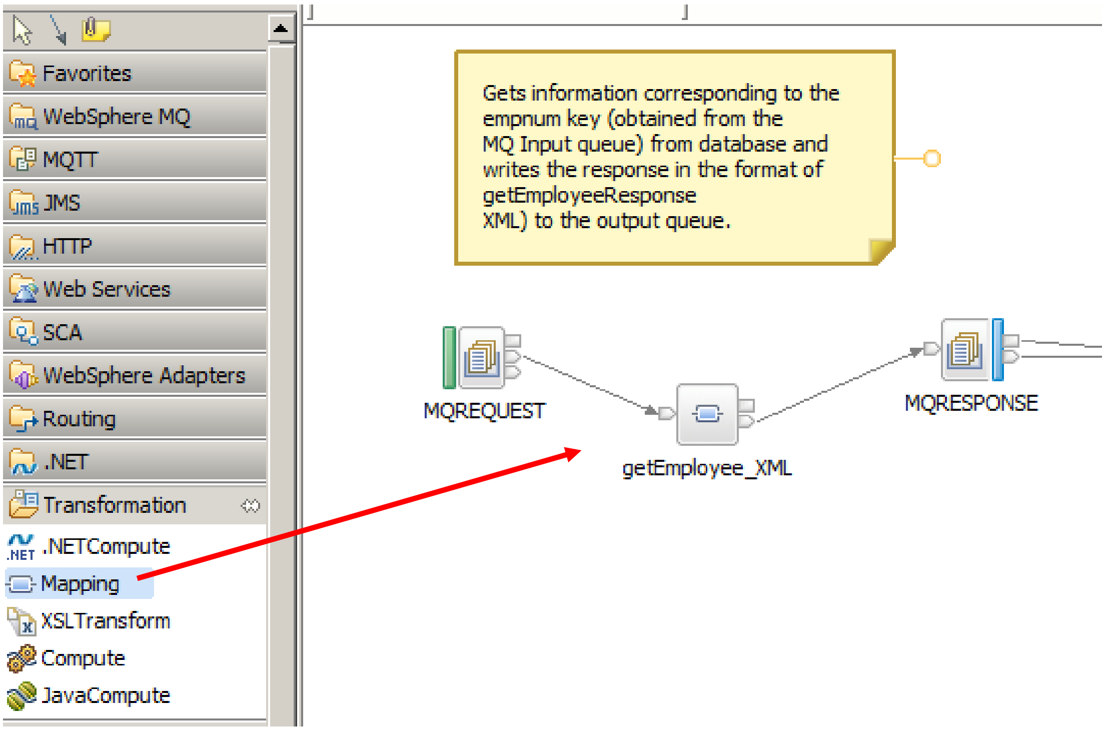
SSL peer name

SSL cipher specification

2.6 Add a Map Node

1. Open the Transformation folder in the node Palette and drop a mapping node on to the flow editor between the MQInput and MQOutput Nodes.

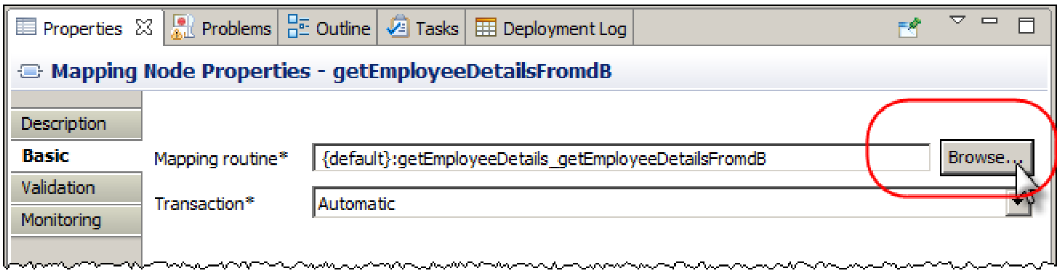
   Call the Mapping node “getEmployee_XML” and connect the terminals as shown:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

2. Click the Mapping node to show the node properties.

   On the Basic tab, click the Browse button to specify an existing mapping routine:

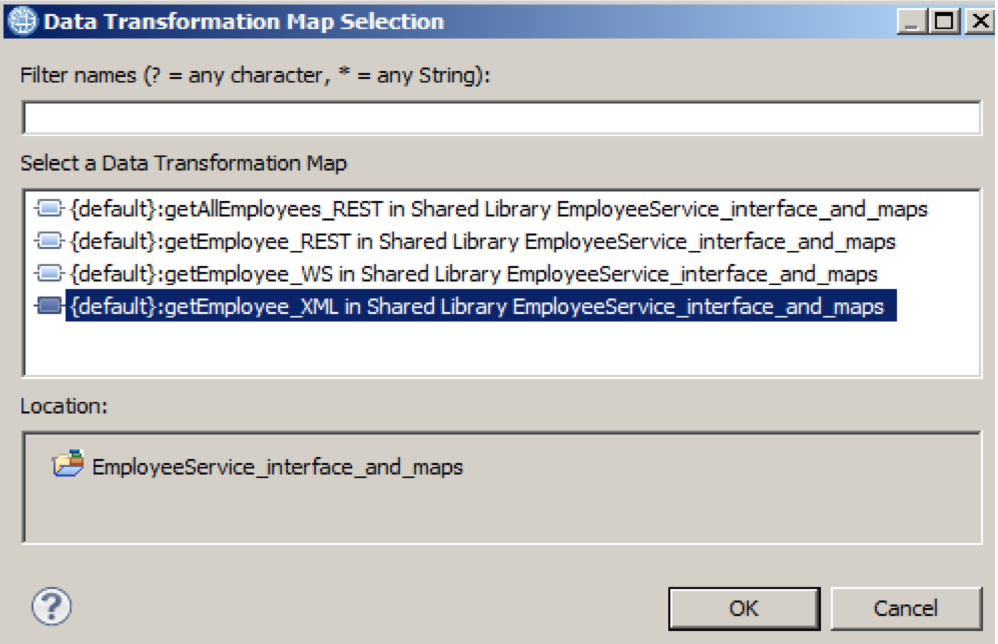


Mapping Node Properties - getEmployeeDetailsFromdB

Mapping routine* {default}:getEmployeeDetails_getEmployeeDetailsFromdB

Transaction* Automatic

3. In the "Data Transformation Map Selection" window, highlight "getEmployee_XML" in the shared library and press OK:



Data Transformation Map Selection

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

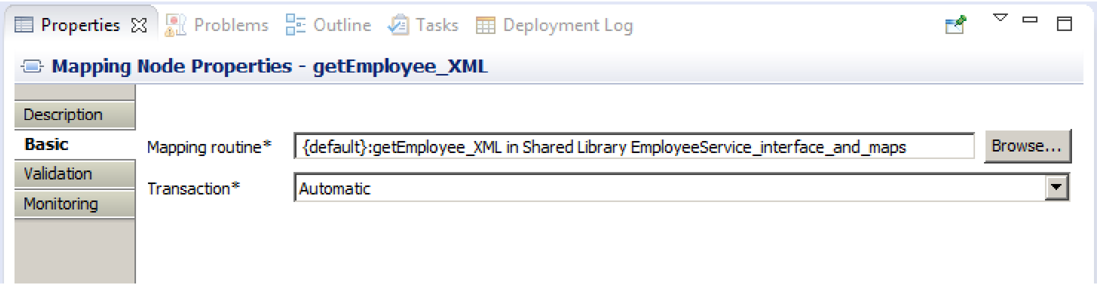
Select a Data Transformation Map

- {default}:getAllEmployees_REST in Shared Library EmployeeService_interface_and_maps
- {default}:getEmployee_REST in Shared Library EmployeeService_interface_and_maps
- {default}:getEmployee_WS in Shared Library EmployeeService_interface_and_maps
- {default}:getEmployee_XML in Shared Library EmployeeService_interface_and_maps

Location:

EmployeeService_interface_and_maps

4. The properties will change to use the existing map from the Shared Library EmployeeService_interface_and_maps.



Mapping Node Properties - getEmployee_XML

Mapping routine* {default}:getEmployee_XML in Shared Library EmployeeService_interface_and_maps

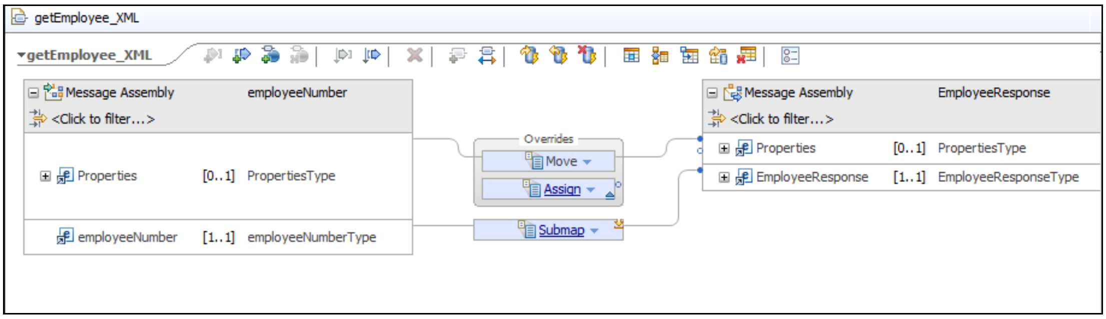
Transaction* Automatic

5. In the flow editor, double click on the map node you just dropped on the flow editor (getEmployee_XML).

In the map editor, you will see there is no reference to the transport protocol that is being used (eg. SOAP, REST). This is because the input and output messages will be simply XML messages, represented by the employeeNumber and EmployeeResponse messages.

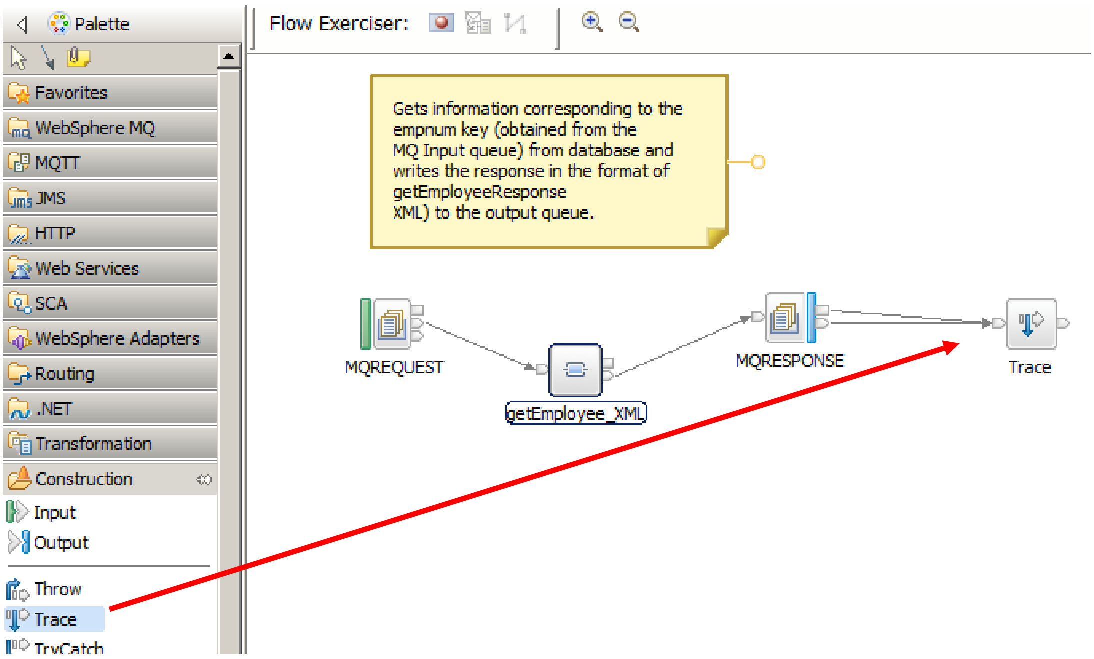
Note that this map uses a common submap to retrieve the data from the database.

Close the map editor.



2.7 Add a Trace node

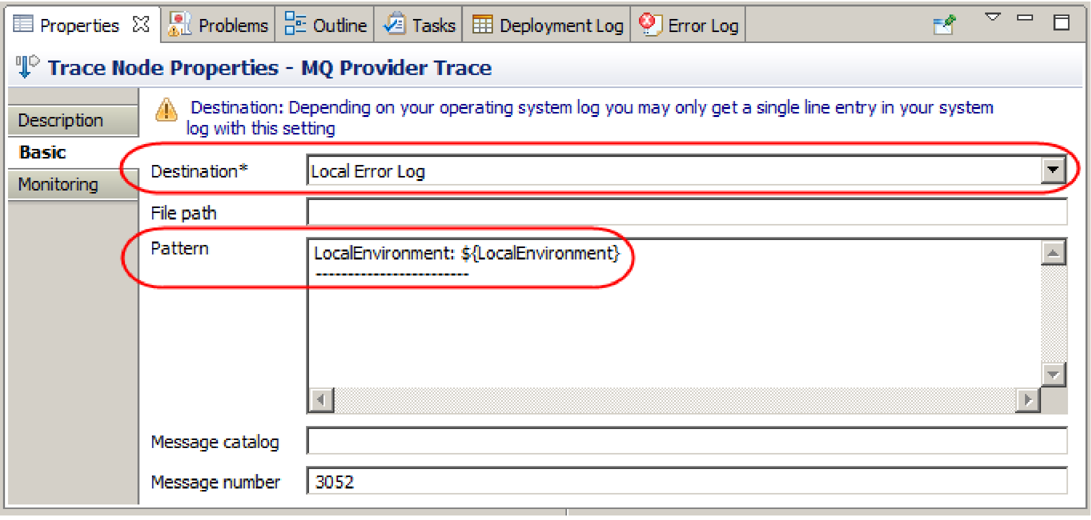
1. Add a trace node after the MQOutput node and connect both the MQOutput terminals to it. Call the trace node "MQ Provider Trace"



You will use this trace node to identify the effects of changes in the MQ Connection tab in the Flow Exerciser tool.

2. Configure the trace node with the following properties:

   - Destination: **`Local Error Log`**
   - Pattern: **`LocalEnvironment: ${LocalEnvironment}`**



Destination: Depending on your operating system log you may only get a single line entry in your system log with this setting

Destination* Local Error Log

File path

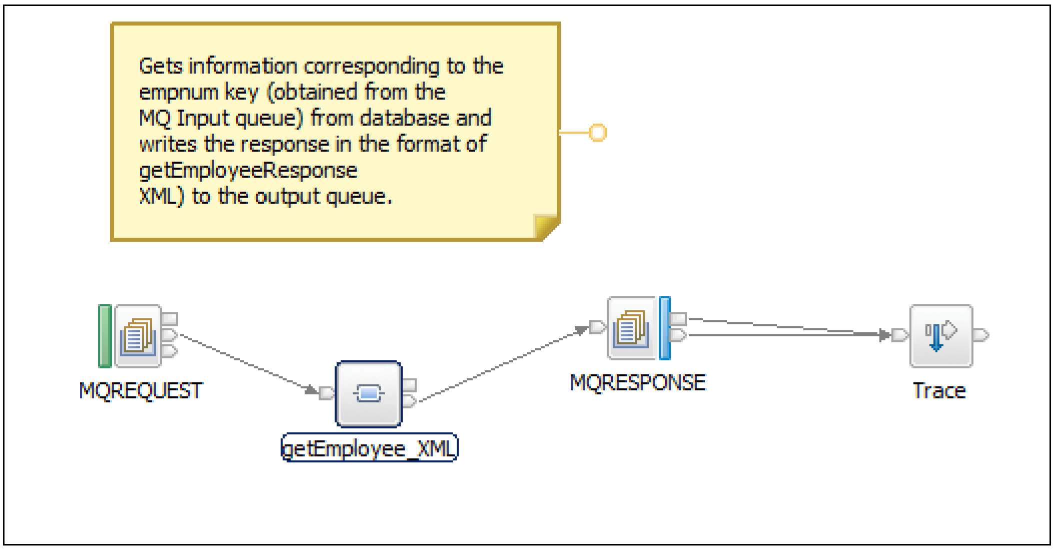
Pattern LocalEnvironment: ${LocalEnvironment}

Message catalog

Message number 3052

3. Save the message flow (Ctrl-S).

   It should look similar to the following when complete:



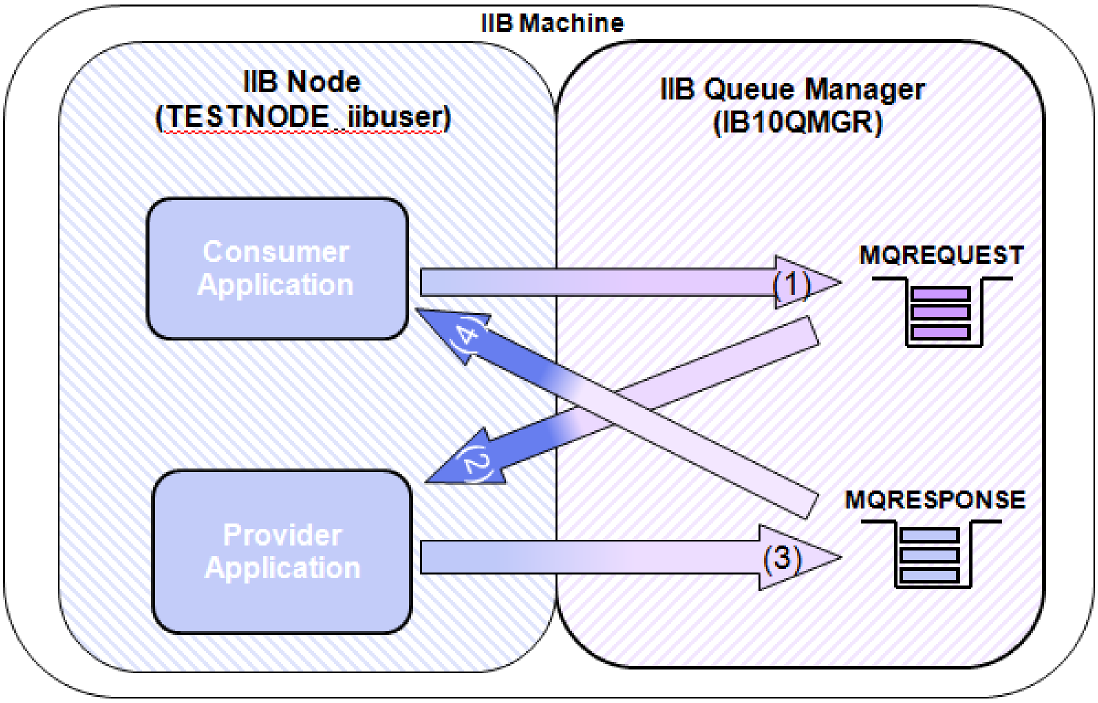
Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

4. Leave the message flow open in the flow editor.

3. Scenario 1: Using an IIB node with an associated queue manager

In this section you will configure the applications to use queues defined on the local queue manager that you have associated with the TESTNODE_iibuser. This is the structure that corresponds to the use of MQ in previous versions of IIB.

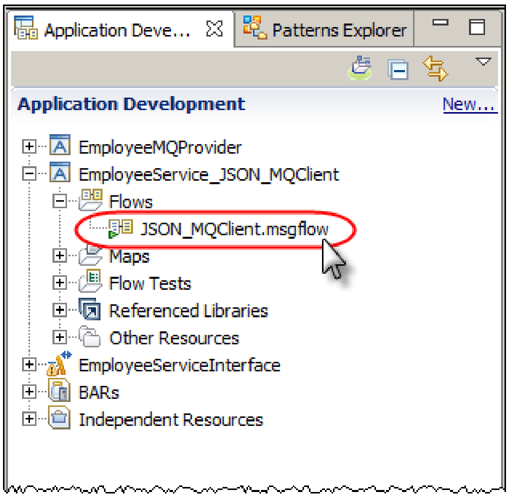
The IIB/MQ topology will look like this:



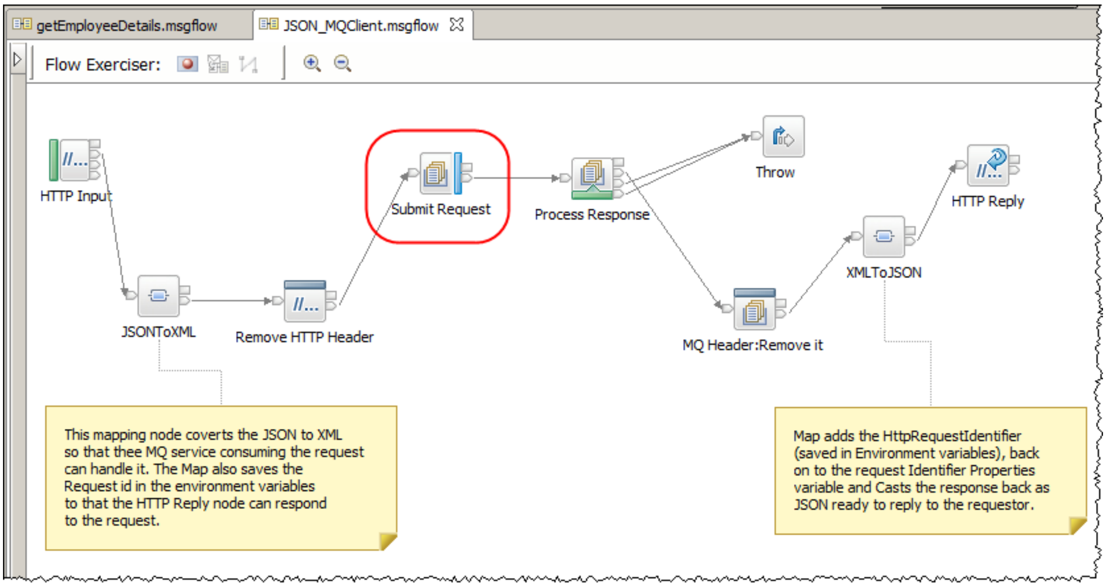
3.1 Review the MQ Client Application

1. In the Integration Toolkit, expand the EmployeeService_JSON_MQClient application.

   Open the JSON_MQClient message flow:



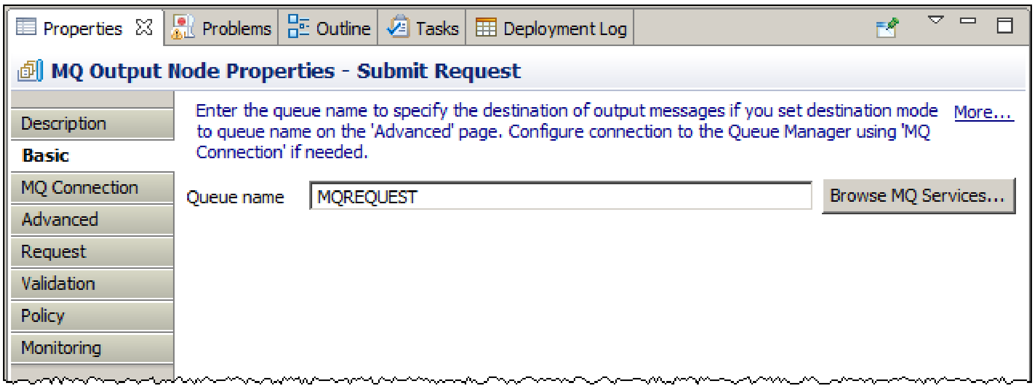
2. In the message flow, click the "Submit Request" MQOutput node to show the properties of the node:



This mapping node coverts the JSON to XML so that thee MQ service consuming the request can handle it. The Map also saves the Request id in the environment variables to that the HTTP Reply node can respond to the request.

Map adds the HttpRequestIdentifier (saved in Environment variables), back on to the request Identifier Properties variable and Casts the response back as JSON ready to reply to the requestor.

3. In IIB V10 the Basic tab now contains only the queue name to be used by the MQOutput node:

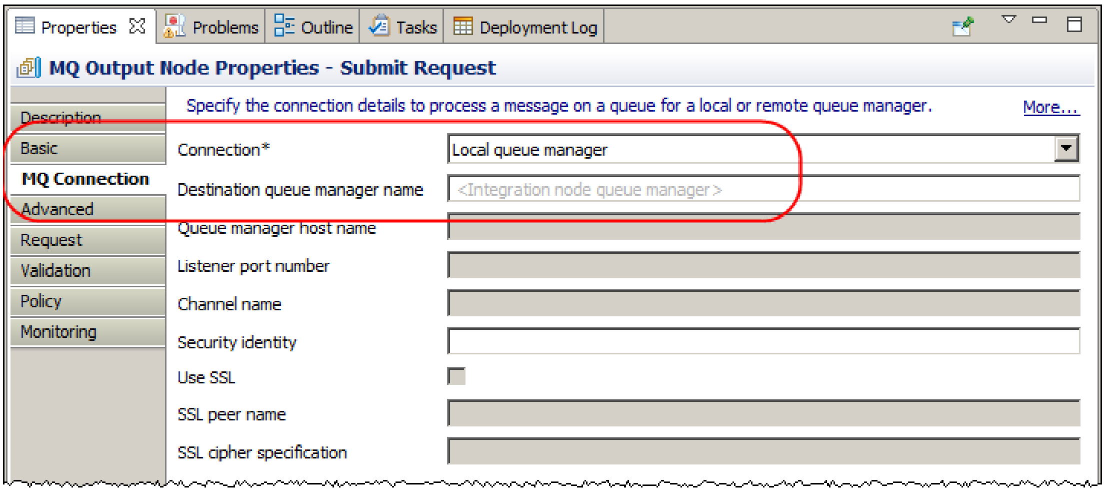


MQ Output Node Properties - Submit Request

Enter the queue name to specify the destination of output messages if you set destination mode to queue name on the 'Advanced' page. Configure connection to the Queue Manager using 'MQ Connection' if needed.

Queue name: MQREQUEST

4. The queue manager connection details are now specified on the MQ Connection tab:



MQ Output Node Properties - Submit Request

Specify the connection details to process a message on a queue for a local or remote queue manager.

Connection*: Local queue manager

Destination queue manager name: <Integration node queue manager>

Queue manager host name:

Listener port number:

Channel name:

Security identity:

Use SSL: □

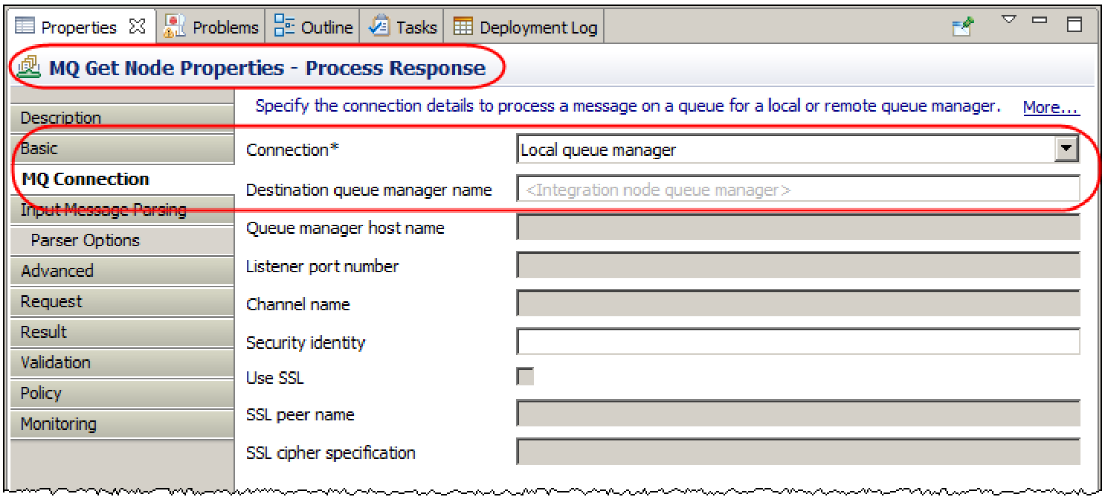
SSL peer name:

SSL cipher specification:

Ensure the same settings are specified that were configured for the MQInput node in the getEmployeeDetails message flow in the EmployeeMQProvider Application (i.e. ensure there is nothing specified in the "Destination queue manager name" and specify "Local queue manager" as the Connection).

5. Click the MQGet node called "*Process Response*" to show its properties.

   (*Note that the MQGet node has exactly the same MQ Connection options available and is configured to use the queue manager defined on the Integration Node*):



MQ Get Node Properties - Process Response

Specify the connection details to process a message on a queue for a local or remote queue manager. More...

Connection*	Local queue manager
Destination queue manager name	<Integration node queue manager>
Queue manager host name	
Listener port number	
Channel name	
Security identity	
Use SSL	□
SSL peer name	
SSL cipher specification	

3.2 Deploy and test using the Flow Exerciser

You will now use the Flow Exerciser to test the applications on TESTNODE_iibuser.

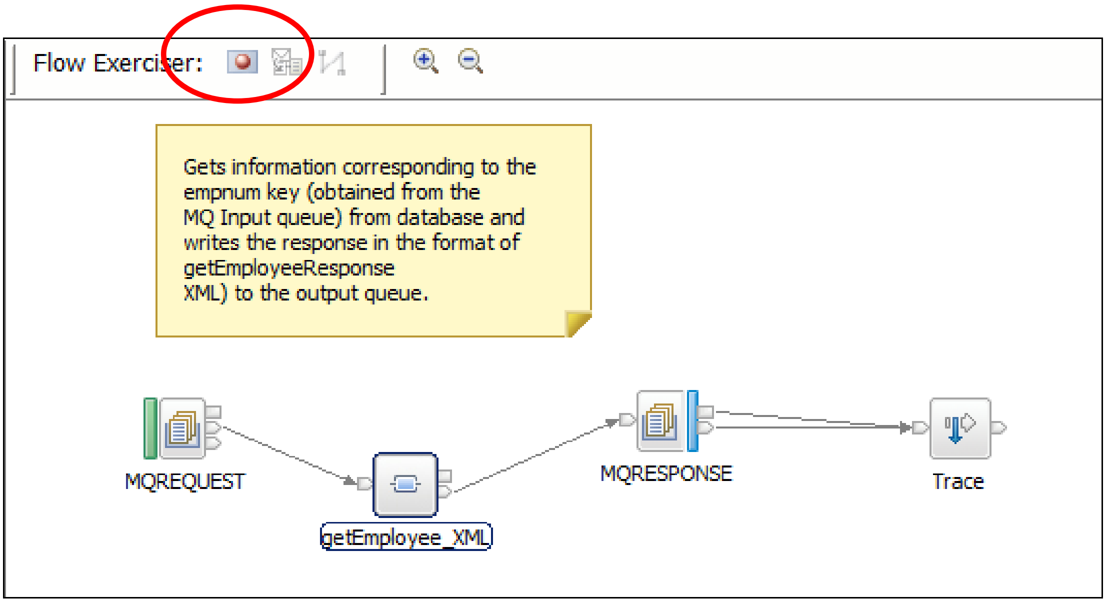
Earlier in the lab, you configured TESTNODE_iibuser to use **IB10QMGR** as its local queue manager in the same way that previous releases of IIB were configured. You will see how MQ flow nodes are now configured to use this configuration in IIB V10.

The queues used by the MQOutput and the MQGet node are already defined on queue manager **IB10QMGR**. However in order to function correctly the EmployeeMQProvider must be deployed.

You will use the Flow Exerciser to test this scenario. The Flow Exerciser automatically deploys applications and any pre-requisite Shared Libraries needed by the applications.

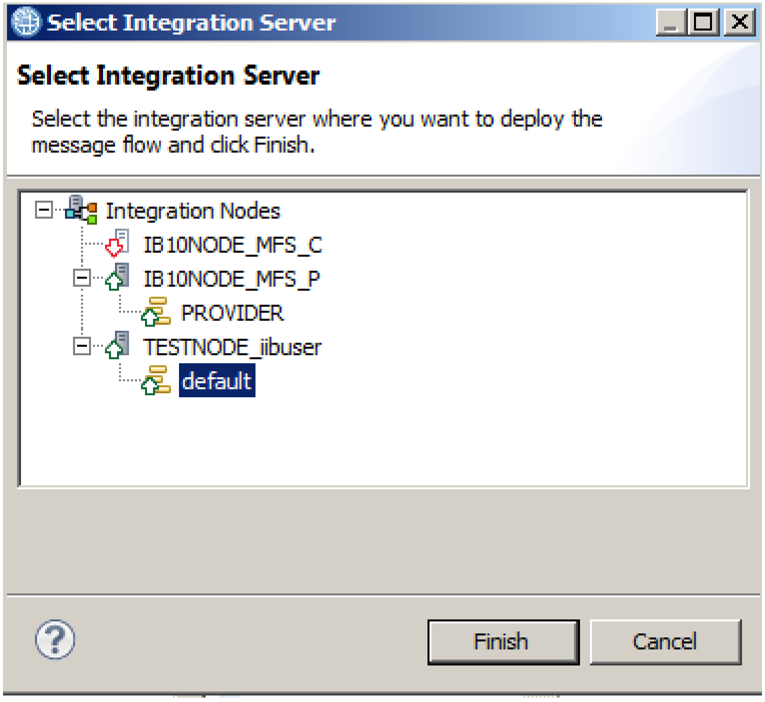
1. Delete all flows and resources from TESTNODE_iibuser/default.

2. Return to the open **getEmployeeDetails.msgflow** and click the record button:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

3. If you have more than one Integration Node defined and running, select "**default**" on "**TESTNODE_iibuser**" in the "Select Integration Server" dialog:

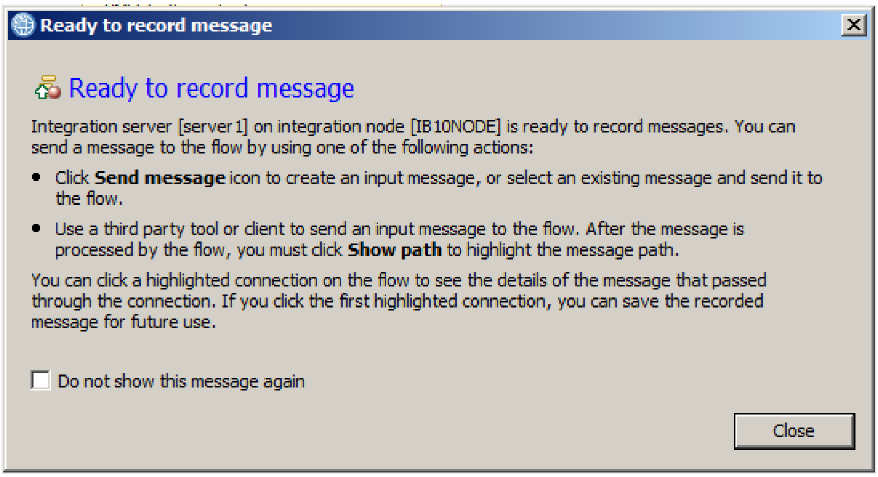


Select Integration Server

Select the integration server where you want to deploy the message flow and click Finish.

The Flow Exerciser will build and deploy a test bar file with the required resoures for the application to run.

4. If you are prompted with the "Ready to record message", click close:



Ready to record message

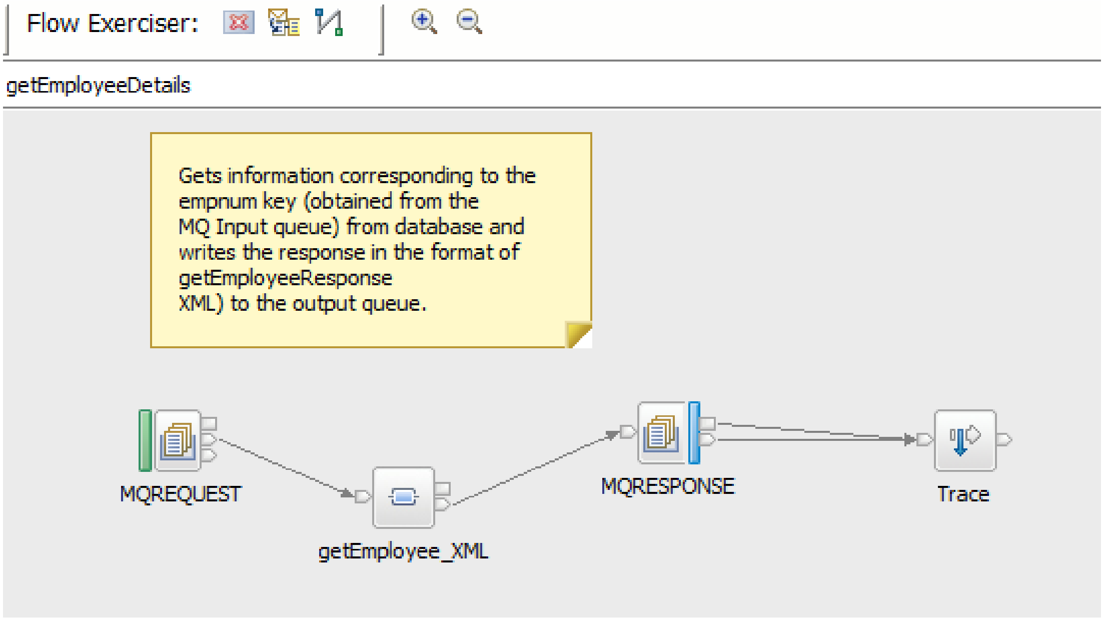
Integration server [server1] on integration node [IB10NODE] is ready to record messages. You can send a message to the flow by using one of the following actions:

- Click **Send message** icon to create an input message, or select an existing message and send it to the flow.
- Use a third party tool or client to send an input message to the flow. After the message is processed by the flow, you must click **Show path** to highlight the message path.

You can click a highlighted connection on the flow to see the details of the message that passed through the connection. If you click the first highlighted connection, you can save the recorded message for future use.

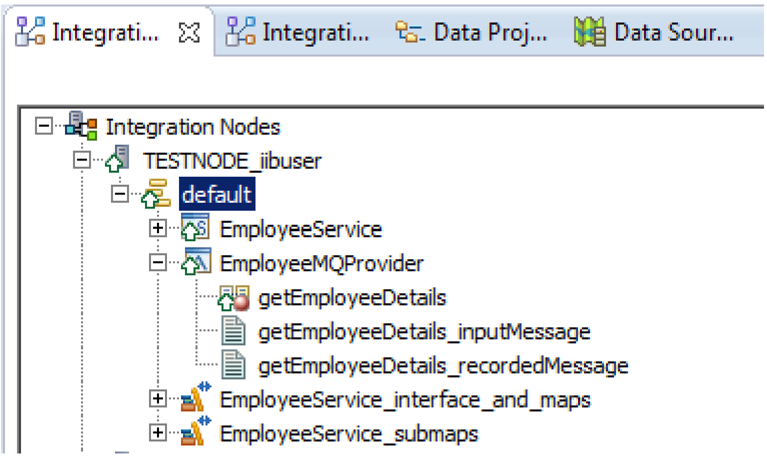
☐ Do not show this message again

5. The canvas background in the message flow editor will grey out, indicating that the message flow is ready to record data sent through it:



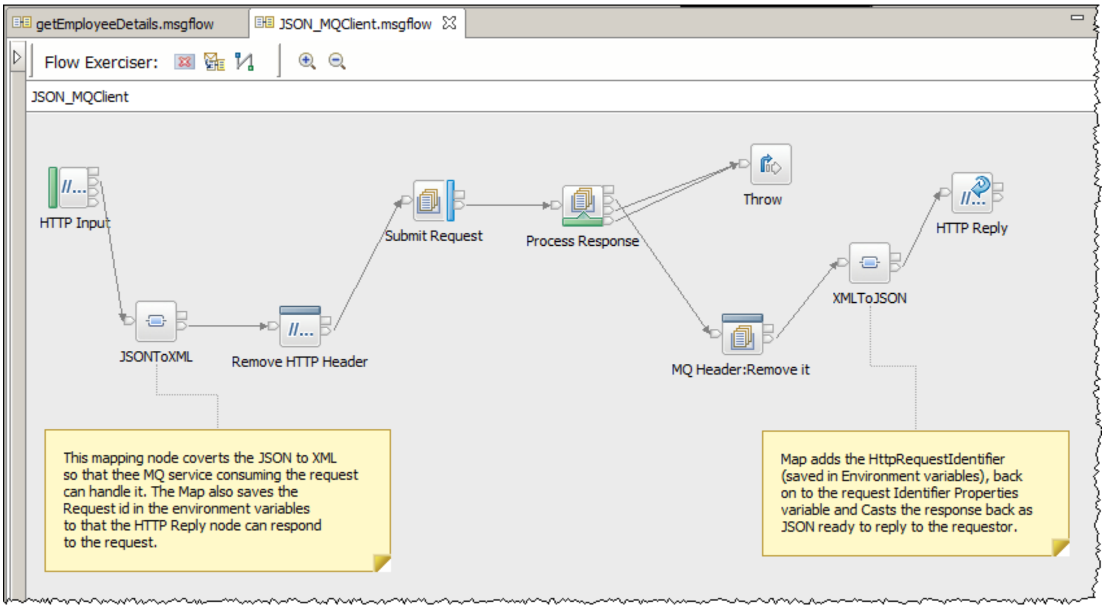
Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

6. Note a record icon now appears on the getEmployeeDetails message flow in the Integration Nodes view :



- Integration Nodes
  - TESTNODE_iibuser
    - default
      - EmployeeService
      - EmployeeMQProvider
        - getEmployeeDetails
        - getEmployeeDetails_inputMessage
        - getEmployeeDetails_recordedMessage
      - EmployeeService_interface_and_maps
      - EmployeeService_submaps

7. Now switch to the open message flow JSON_MQClient.msgflow and repeat the steps above until the message flow is in record mode:

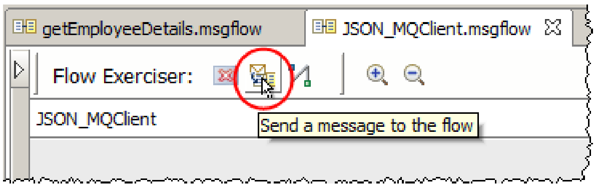


This mapping node coverts the JSON to XML so that thee MQ service consuming the request can handle it. The Map also saves the Request id in the environment variables to that the HTTP Reply node can respond to the request.

Map adds the HttpRequestIdentifier (saved in Environment variables), back on to the request Identifier Properties variable and Casts the response back as JSON ready to reply to the requestor.

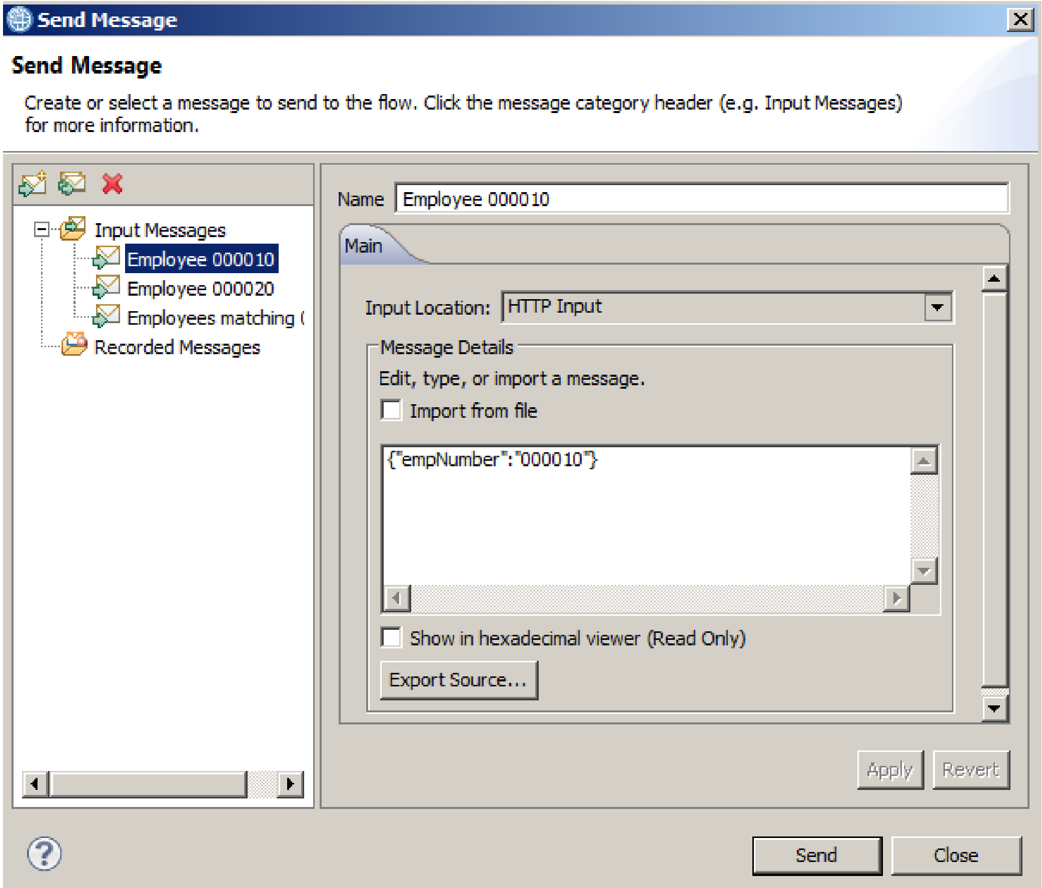
Both message flows are now ready to record activity.

8. In the JSON_MQClient.msgflow, click the “Send a message to the flow” icon:



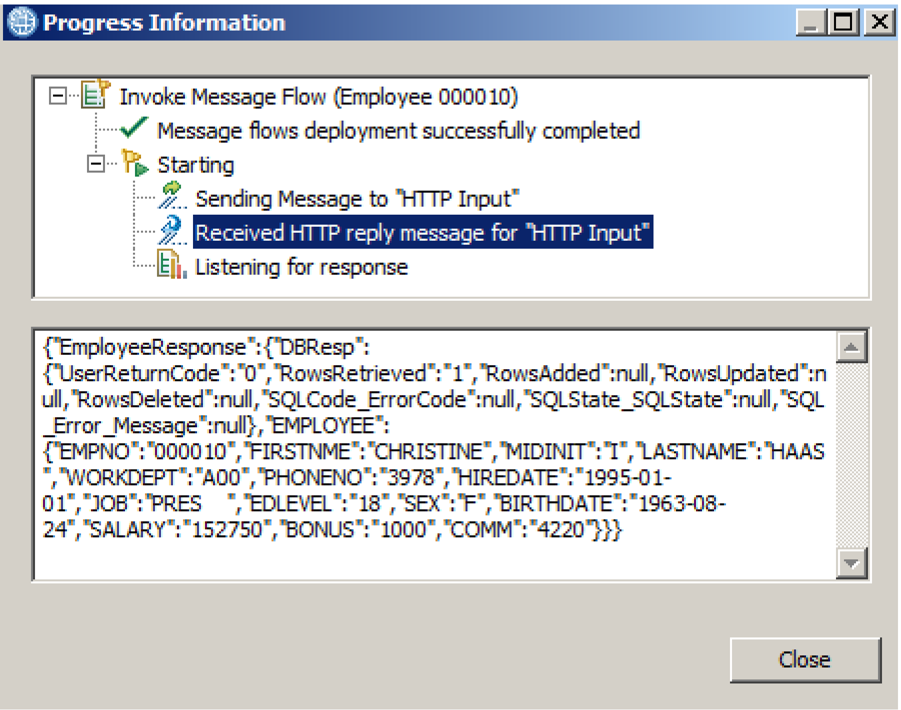
9. In the Send Message window, expand “Input Messages”.

   Highlight “Employee 000010” and click Send:



```
{"empNumber":"000010"}
```

10. The “Progress Information” window will open.
    When the line “Received HTTP reply message for HTTP Input” appears, highlight the line to see the data retrieved from the database:



Progress Information
Invoke Message Flow (Employee 000010)
Message flows deployment successfully completed
Starting
Sending Message to "HTTP Input"
Received HTTP reply message for "HTTP Input"
Listening for response
Close

```
{"EmployeeResponse":{"DBResp":
{"UserReturnCode":"0","RowsRetrieved":"1","RowsAdded":null,"RowsUpdated":n
ull,"RowsDeleted":null,"SQLCode_ErrorCode":null,"SQLState_SQLState":null,"SQL
_Error_Message":null},"EMPLOYEE":
{"EMPNO":"000010","FIRSTNME":"CHRISTINE","MIDINIT":"I","LASTNAME":"HAAS
","WORKDEPT":"A00","PHONENO":"3978","HIREDATE":"1995-01-
01","JOB":"PRES    ","EDLEVEL":"18","SEX":"F","BIRTHDATE":"1963-08-
24","SALARY":"152750","BONUS":"1000","COMM":"4220"}}}
```

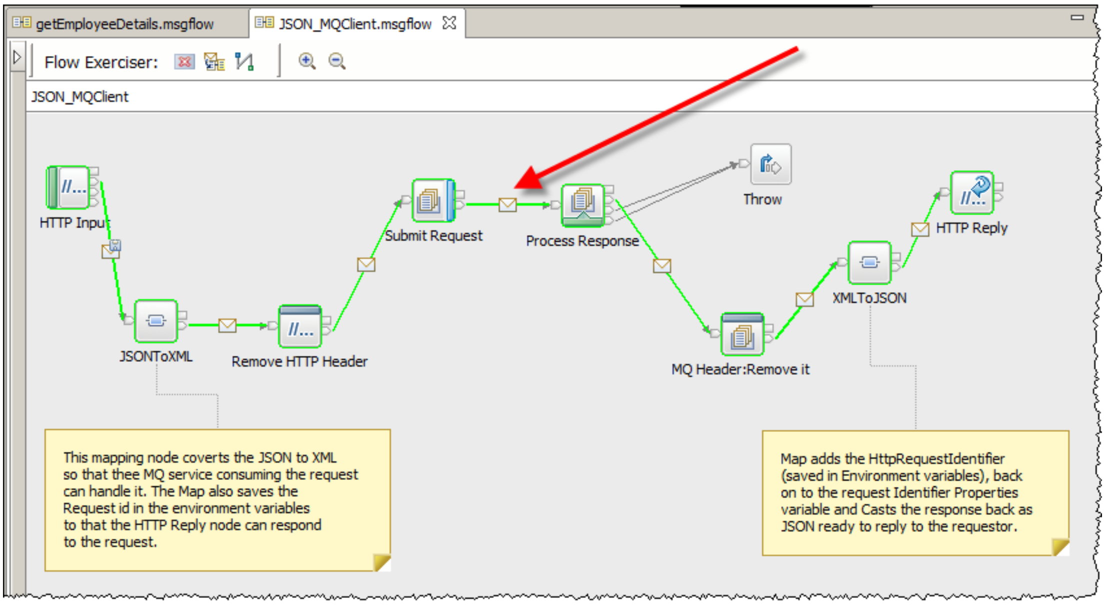
3.2.1 Verify the connection to the queue manager

1. Close the Progress Information window.

   The message flow will now show the route the message took through the message flow (the connectors will be green).

   Each connection between the nodes will display an envelope icon. Clicking this icon will display the contents of the recorded message at that point in the flow.

   Click the envelope between the MQOutput node (“Submit Request”) and MQGet nodes (“Process Response”):



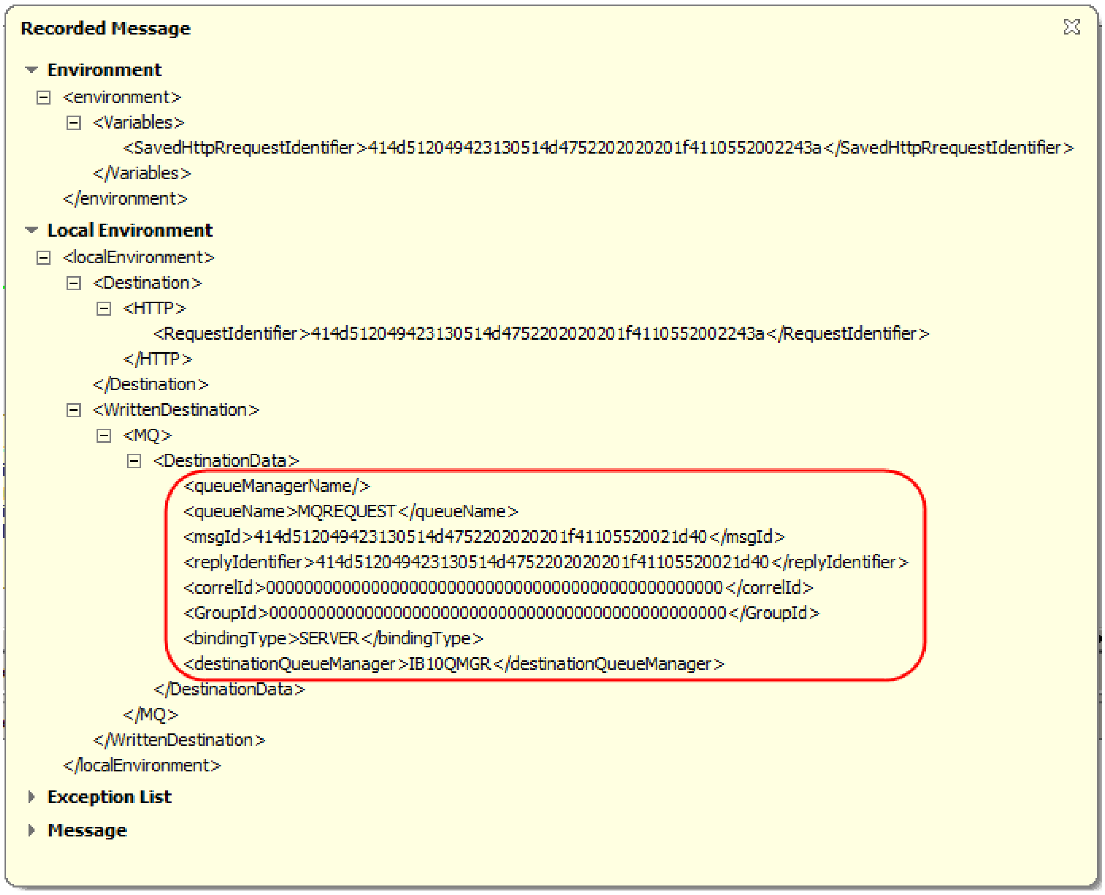
This mapping node coverts the JSON to XML so that thee MQ service consuming the request can handle it. The Map also saves the Request id in the environment variables to that the HTTP Reply node can respond to the request.

Map adds the HttpRequestIdentifier (saved in Environment variables), back on to the request Identifier Properties variable and Casts the response back as JSON ready to reply to the requestor.

2. A yellow popup window will open with the title “Recorded Message”.

   Expand **Environment** and **Local Environment.**

   Note the details in “DestinationData” reflect the details used by the MQOutput node:

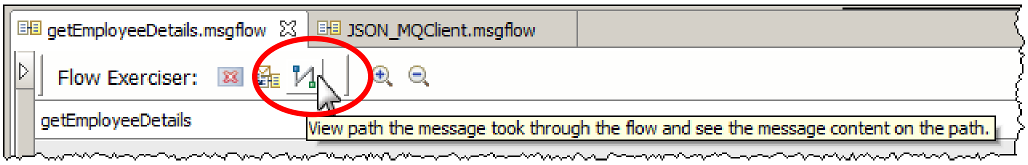


```
Recorded Message

Environment
<environment>
  <Variables>
    <SavedHttpRrequestIdentifier>414d5120494231305 14d4752202020201f4110552002243a</SavedHttpRrequestIdentifier>
  </Variables>
</environment>
Local Environment
<localEnvironment>
  <Destination>
    <HTTP>
      <RequestIdentifier>414d512049423130514d475220202020 1f4110552002243a</RequestIdentifier>
    </HTTP>
  </Destination>
  <WrittenDestination>
    <MQ>
      <DestinationData>
        <queueManagerName/>
        <queueName>MQREQUEST</queueName>
        <msgId>414d512049423130514d4752202020201f41105520021d40</msgId>
        <replyIdentifier>414d512049423130514d4752202020201f41105520021d40</replyIdentifier>
        <correlId>000000000000000000000000000000000000000000000000</correlId>
        <GroupId>000000000000000000000000000000000000000000000000</GroupId>
        <bindingType>SERVER</bindingType>
        <destinationQueueManager>IB10QMGR</destinationQueueManager>
      </DestinationData>
    </MQ>
  </WrittenDestination>
</localEnvironment>
Exception List
Message
```

3. Close the Recorded Message window.

4. Switch to the getEmployeeDetails.msgflow (the MQ Provider).

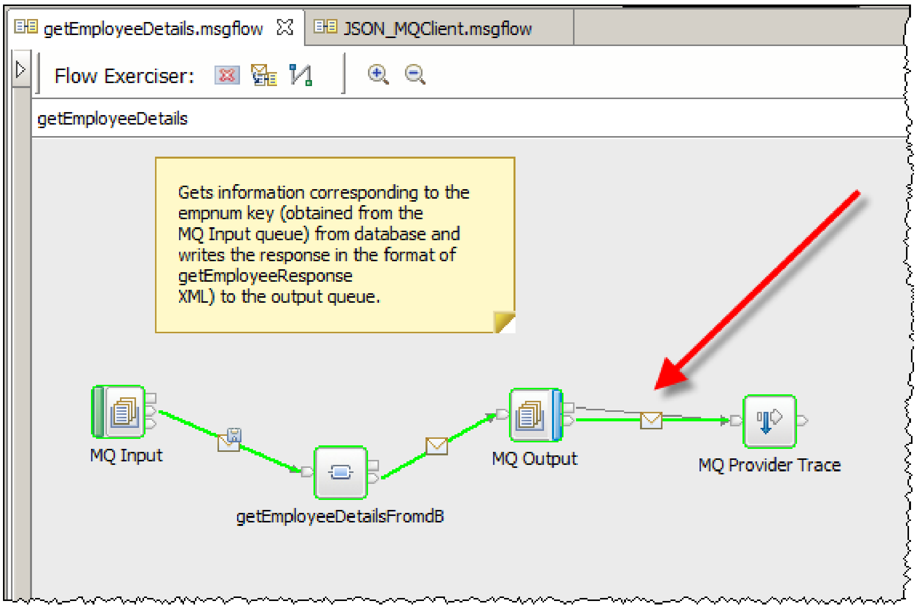
   Click the icon to view the path the message took through this flow:



View path the message took through the flow and see the message content on the path.

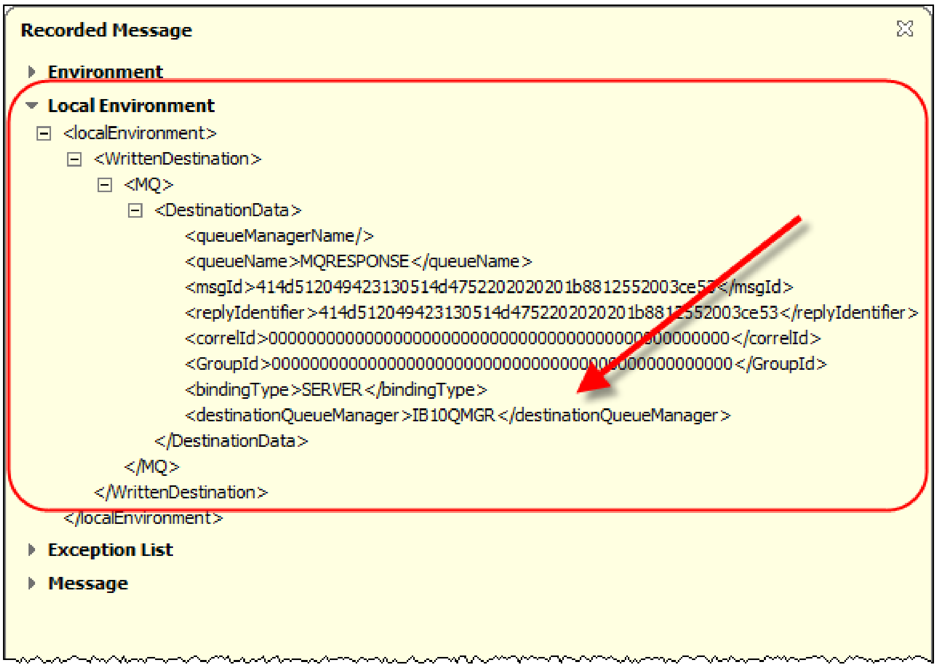
5. Note the green line highlighting the path the message took through the message flow.

   Click on the envelope icon on the connector between the MQOutput node and the MQ Provider Trace Node:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

6. In the Recorded Message popup window expand Local Environment to verify the DestinationData. Note the **bindingtype, DestinationQueueManager** details reflect those defined on the MQ Connections tab for the MQOutput node:



Recorded Message

- Environment
- Local Environment

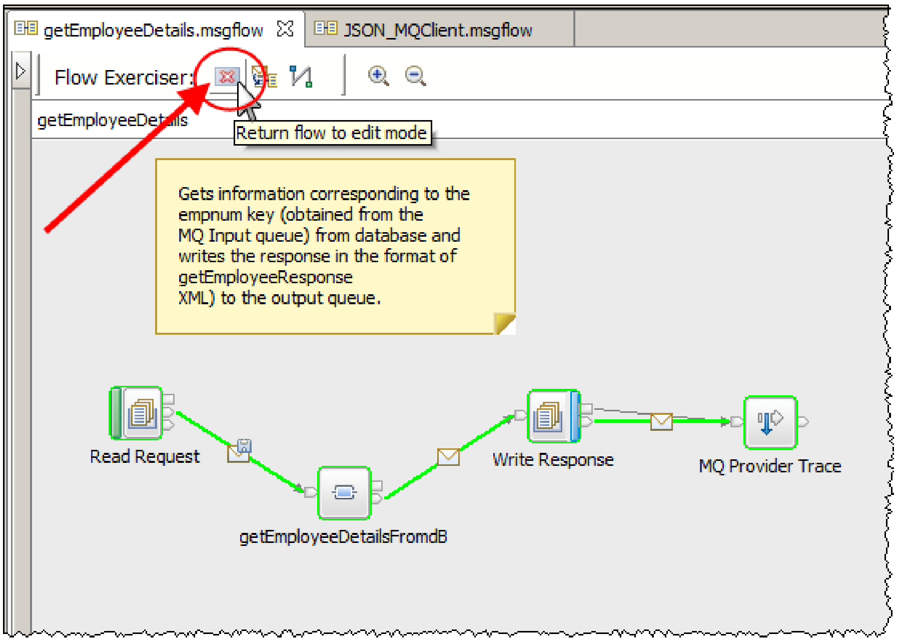
```
<localEnvironment>
  <WrittenDestination>
    <MQ>
      <DestinationData>
        <queueManagerName/>
        <queueName>MQRESPONSE</queueName>
        <msgId>414d5120494231305l4d475220202020201b8812552003ce53</msgId>
        <replyIdentifier>414d51204942313051 4d475220202020201b8812552003ce53</replyIdentifier>
        <correlId>000000000000000000000000000000000000000000000000</correlId>
        <GroupId>000000000000000000000000000000000000000000000000</GroupId>
        <bindingType>SERVER</bindingType>
        <destinationQueueManager>IB10QMGR</destinationQueueManager>
      </DestinationData>
    </MQ>
  </WrittenDestination>
</localEnvironment>
```

- Exception List
- Message

We will revisit this area when we change the MQ Connection details.

7. Close the Recorded Message popup.

8. Return the message flow to edit mode by clicking the “Return flow to edit mode” button:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

9. If you get one, dismiss the warning that the action will clear all recorded messages from the message flow.

The message flow will return to a white background.

10. Repeat the above step on JSON_MQClient.msgflow to return that flow to edit mode.

4. Preparing to run MQ applications on IIB nodes with no associated queue manager

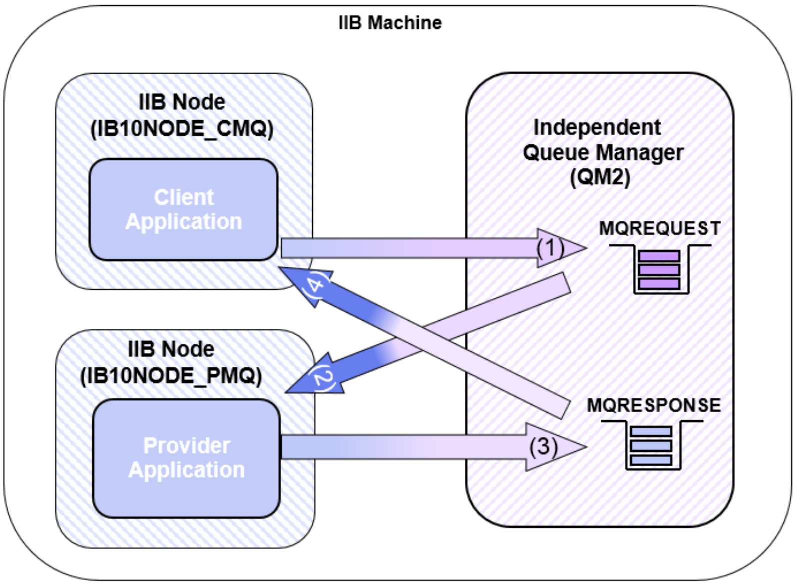
MQ Flexibility in IIB V10 enables MQ Applications to run on IIB nodes which do not have a queue manager associated with the IIB node.

You will now configure the applications provided with this lab guide to run on separate IIB nodes. Both IIB nodes will be created **without specifying a queue manager (**no "–q" option).

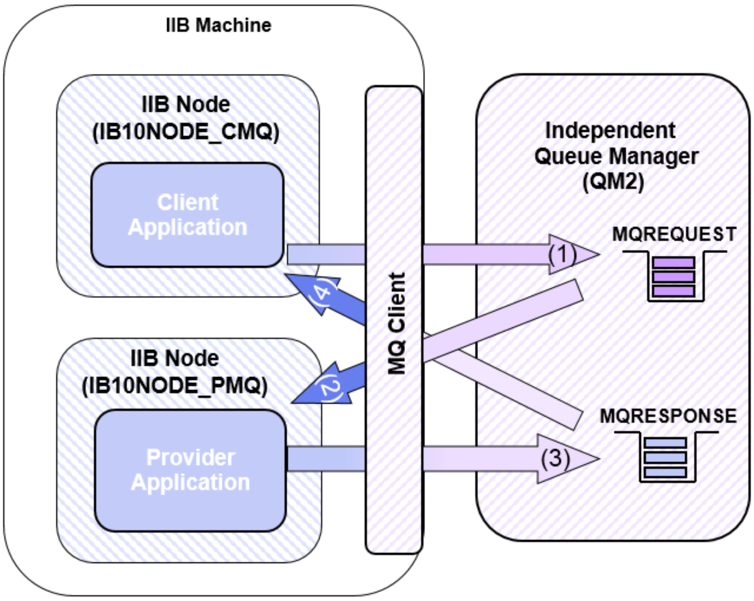
The MQREQUEST and MQRESPONSE queues will be defined on a queue manager which is independent of the two IIB nodes.

The MQ applications will be configured to access these queues by configuring the queue manager (on which they are defined) as:

1) A local queue manager configuration specified on the MQ Connection tab for the MQ nodes:



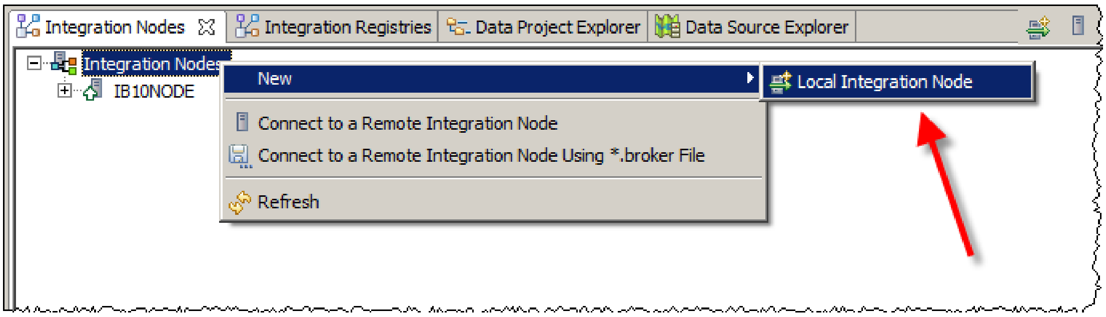
2) A (remote) MQ client configuration specified on the MQ Connection tab for the MQ nodes



4.1 Create the IIB Nodes

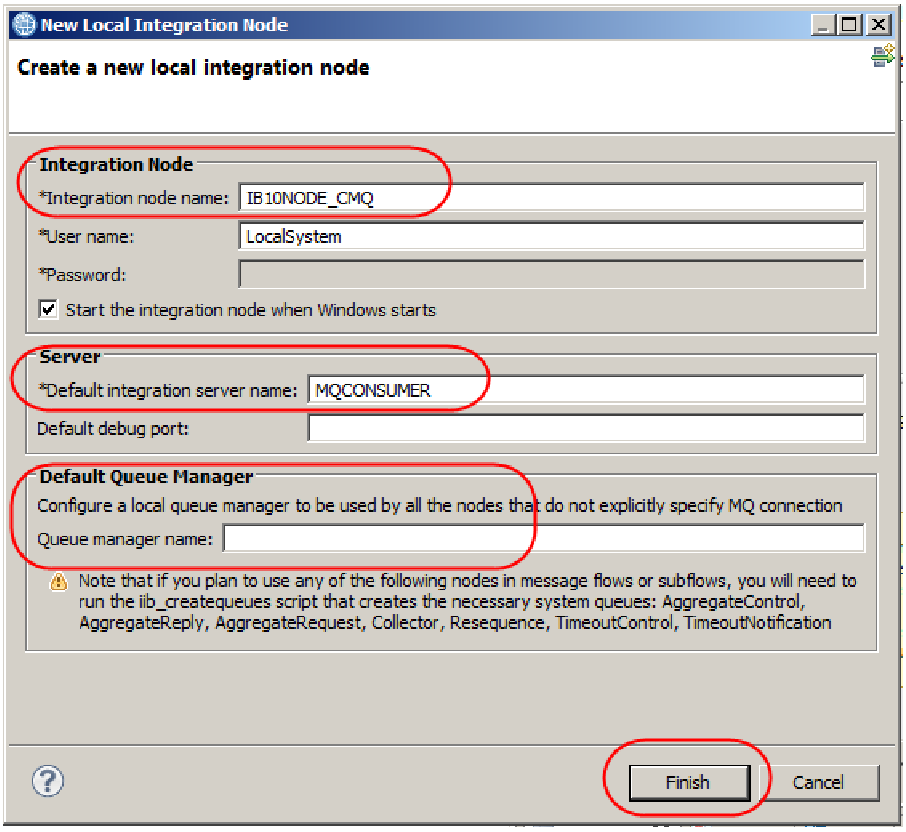
In this section, you will define the IIB nodes you will use throughout the remainder of the Lab guide.

1. In the Integration Nodes view, right click on "Integration Nodes" and select New > Local Integration Node:



2. In the Create a new local Integration Node window specify the following and click finish: (*Note: **do not** specify a default queue manager*)

- Integration node name: **`IB10NODE_CMQ`**
- Default integration server name: **`MQCONSUMER`**
- Default queue manager: *<leave blank >*



New Local Integration Node

Create a new local integration node

Integration Node

*Integration node name: IB10NODE_CMQ

*User name: LocalSystem

*Password:

☑ Start the integration node when Windows starts

Server

*Default integration server name: MQCONSUMER

Default debug port:

Default Queue Manager

Configure a local queue manager to be used by all the nodes that do not explicitly specify MQ connection

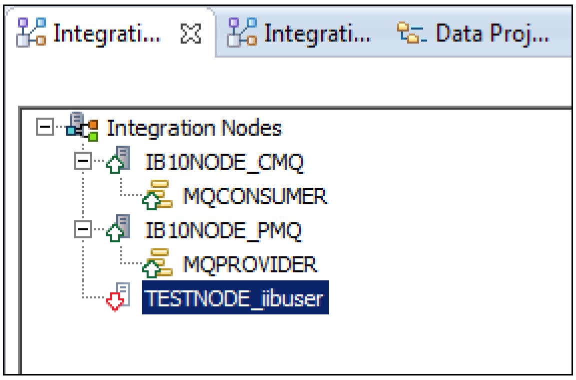
Queue manager name:

Note that if you plan to use any of the following nodes in message flows or subflows, you will need to run the iib_createqueues script that creates the necessary system queues: AggregateControl, AggregateReply, AggregateRequest, Collector, Resequence, TimeoutControl, TimeoutNotification

3. Repeat the above process and create a local Integration Node called **IB10NODE_PMQ** with server **MQPROVIDER.**

4. Stop the **TESTNODE_iibuser** node (right click on the node and select stop).

5. The list of nodes in the Integration Nodes view should now look like this:



6. Open an **Integration Console** (from the Start menu) and navigate to

```
C:\student10\MQ_Topology\commands
```

7. Run the command

```
00ConfigureMQIIBNodes.cmd
```

Accept the defaults to start the script.

This script configures the environment for the MQ applications to run successfully by configuring the JDBC, ODBC and security for the Provider node to access the EMPLOYEE table in the HRDB database.

The Integration Nodes will be restarted as part of the script.

When the script has successfully completed, the two Integration Nodes you created earlier:

1) "**IB10NODE_PMQ**" (*with Integration Server called "**MQPROVIDER**"*)

and

2) "**IB10NODE_CMQ**" (*with Integration Server called "**MQCONSUMER**"*)

should still be running (*right click on the node and click refresh to update the Integration Nodes view*).

5. Scenario 2: IIB node with no associated queue manager: Using an MQ SERVER connection

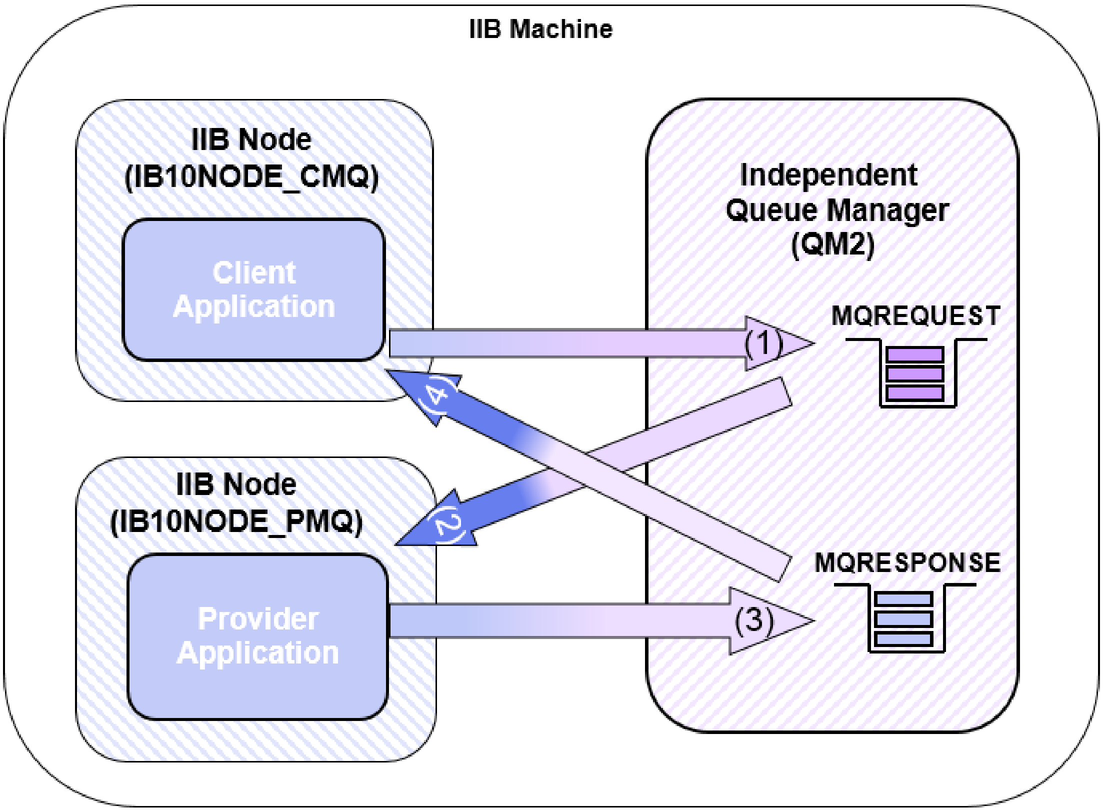
5.1 Configure the MQ nodes to use a different LOCAL queue Manager

You will now configure the MQ applications to use queues on a different queue manager (QM2). Both the queue manager and queues have already been defined for you.

You will configure the Provider application to obtain requests from MQREQUEST and write Responses to MQRESPONSE on the queue manager “QM2” (QM2 is independent of the IIB nodes you will deploy the applications to, however it is defined on the same machine enabling the MQ nodes in the applications to communicate with the queue manager using Server bindings mode).

Similarly the MQ application acting as a Client in this scenario will be configured to write requests to MQREQUEST on QM2 and read the responses from MQRESPONSE on QM2, using a server bindings mode connection to the queue manager.

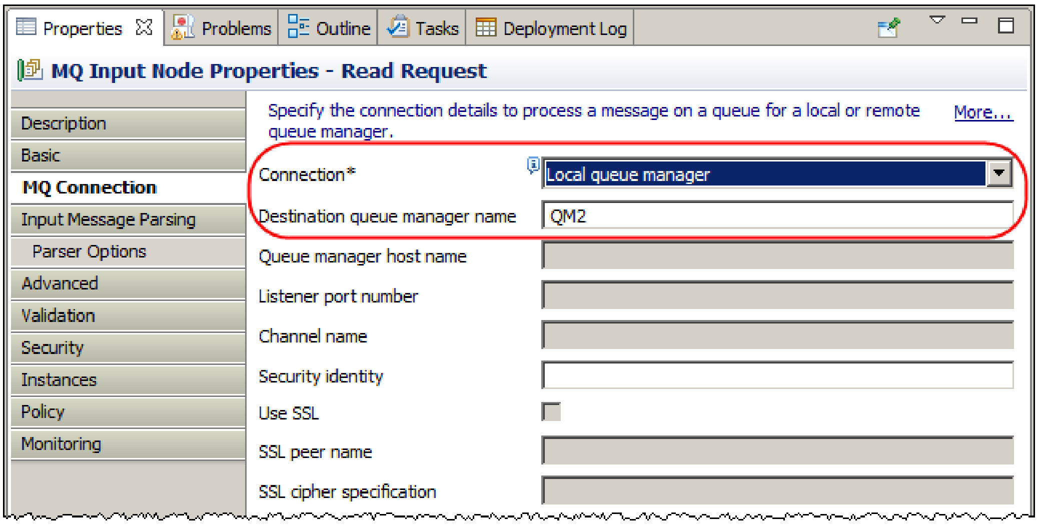
The topology you will set up will look this:



5.1.1 Configure the MQ Provider application

1. In the getEmployeeDetails message flow, click the MQInput node ("Read Request") to show the node properties.

   On the MQ Connection tab, specify **QM2** in the "**Destination queue manager name**" field:



MQ Input Node Properties - Read Request

Specify the connection details to process a message on a queue for a local or remote queue manager.

Connection*: Local queue manager

Destination queue manager name: QM2

Queue manager host name:

Listener port number:

Channel name:

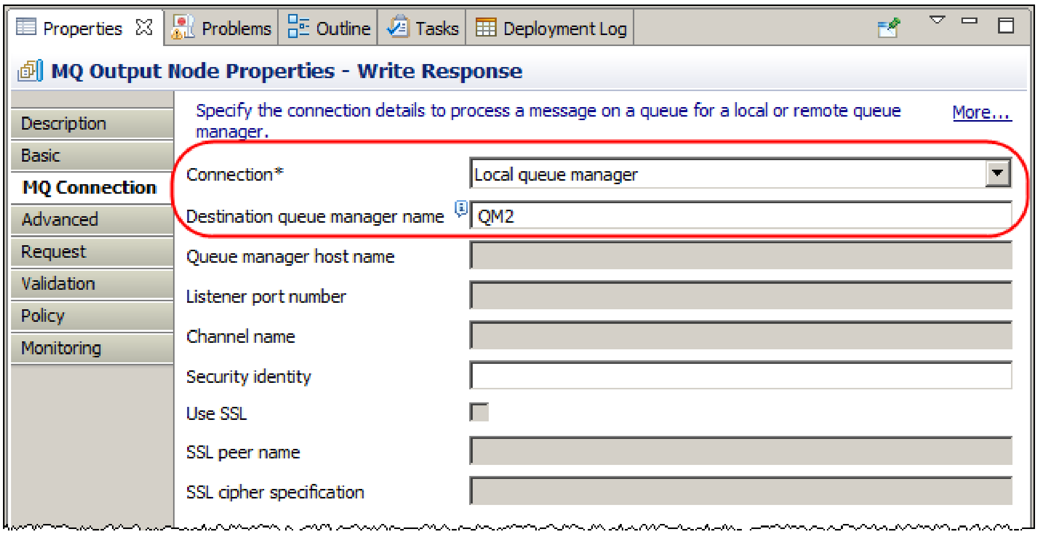
Security identity:

Use SSL: □

SSL peer name:

SSL cipher specification:

2. **Repeat** the previous step for the MQOutput node called "Write Response":



MQ Output Node Properties - Write Response

Specify the connection details to process a message on a queue for a local or remote queue manager.

Connection*: Local queue manager

Destination queue manager name: QM2

Queue manager host name:

Listener port number:

Channel name:

Security identity:

Use SSL: □

SSL peer name:

SSL cipher specification:

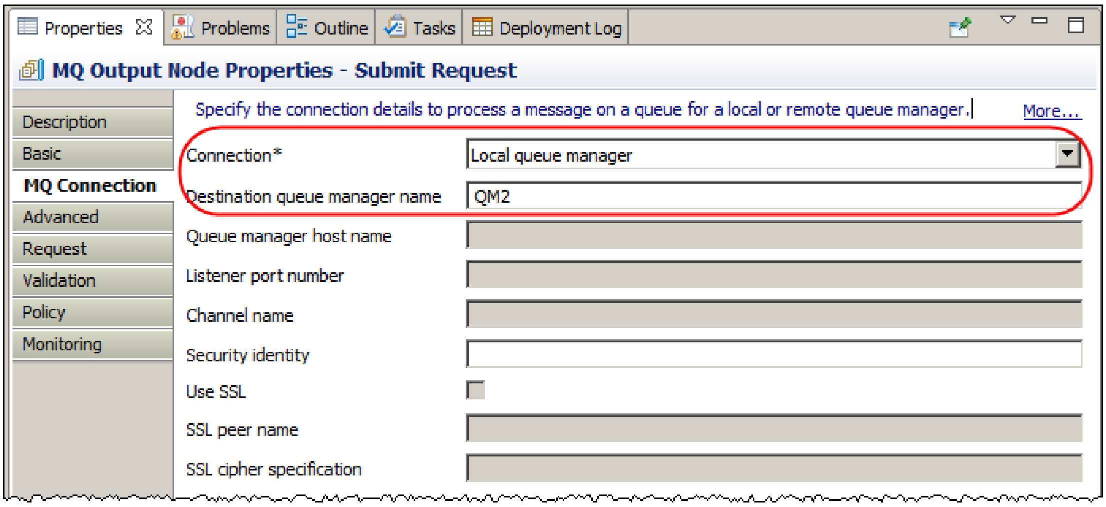
3. Save the message flow (Ctrl-s).

5.1.2 Configure the MQ Client application

You will now configure the MQ Client application to use queues (already defined) on a queue manager (QM2) defined locally.

1. In the JSON_MQClient message flow, click the "Submit Request" MQOutput node to show the properties tab.

   In the MQ Connection Tab specify that the queue exists on a local queue manager called QM2:



MQ Output Node Properties - Submit Request

Specify the connection details to process a message on a queue for a local or remote queue manager. More...

Connection* Local queue manager

Destination queue manager name QM2

Queue manager host name

Listener port number

Channel name

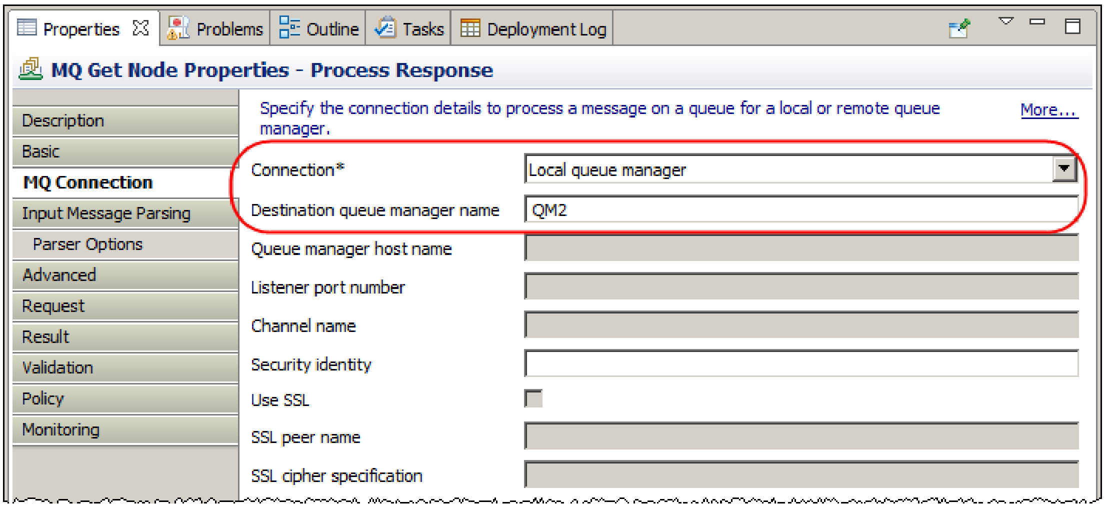
Security identity

Use SSL □

SSL peer name

SSL cipher specification

2. Click the "Process Response" MQGet node and perform the same change:



MQ Get Node Properties - Process Response

Specify the connection details to process a message on a queue for a local or remote queue manager. More...

Connection* Local queue manager

Destination queue manager name QM2

Queue manager host name

Listener port number

Channel name

Security identity

Use SSL □

SSL peer name

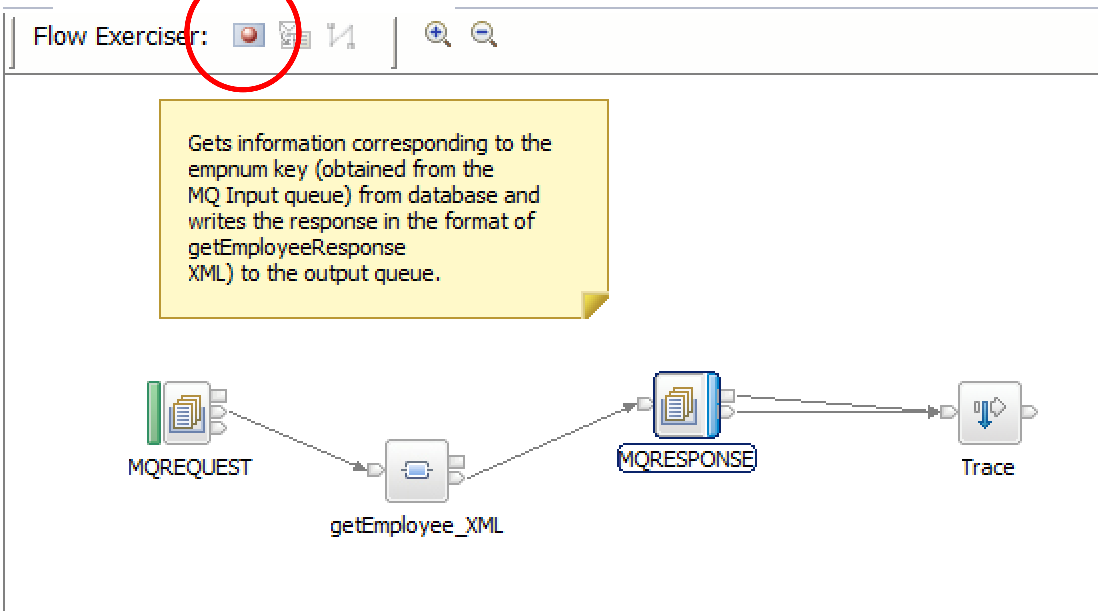
SSL cipher specification

3. Save the message flow (Ctrl-s).

5.2 Test the scenario using the Flow Exerciser

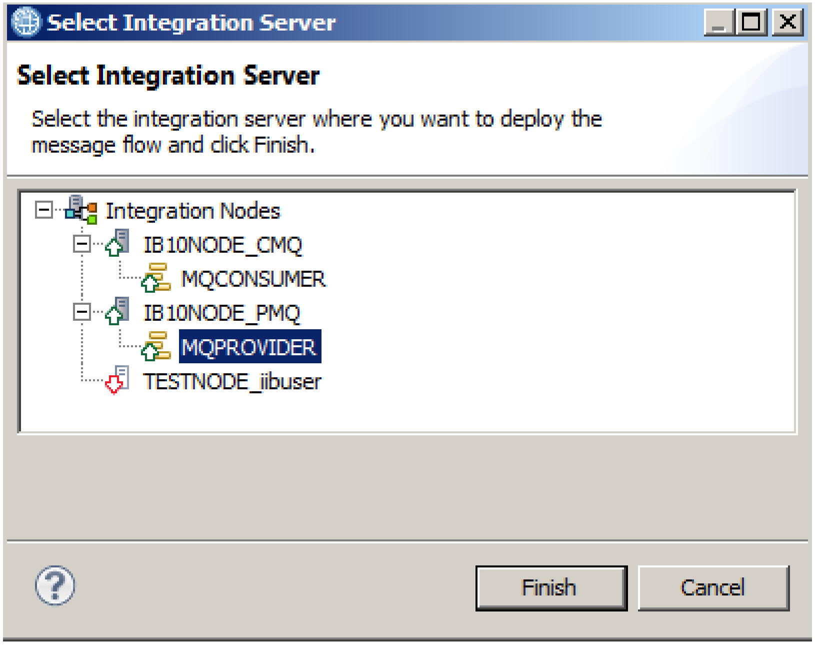
You will now test the applications and verify that the queue being used is defined on QM2.

1. Return to the open **getEmployeeDetails.msgflow** and start the Flow Exerciser:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

2. When prompted where to deploy the application, chose **MQPROVIDER** on **IB10NODE_PMQ** and click Finish:

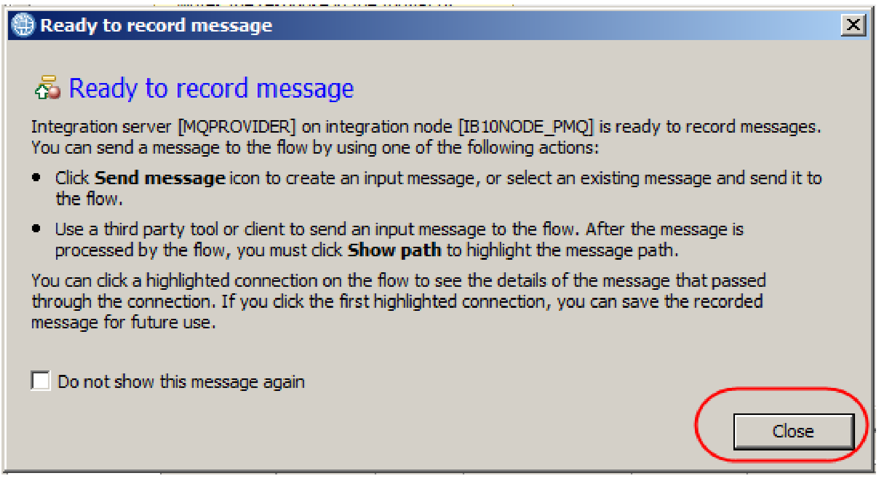


Select Integration Server

Select Integration Server

Select the integration server where you want to deploy the message flow and click Finish.

3. If the message appears, dismiss the “Ready to record message”:



Ready to record message

Ready to record message

Integration server [MQPROVIDER] on integration node [IB10NODE_PMQ] is ready to record messages. You can send a message to the flow by using one of the following actions:

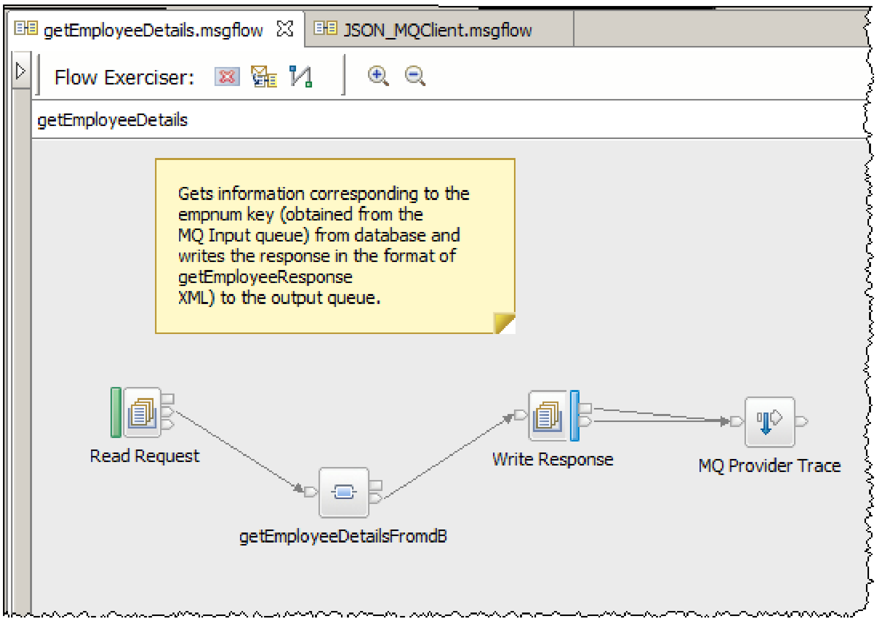
- Click **Send message** icon to create an input message, or select an existing message and send it to the flow.
- Use a third party tool or client to send an input message to the flow. After the message is processed by the flow, you must click **Show path** to highlight the message path.

You can click a highlighted connection on the flow to see the details of the message that passed through the connection. If you click the first highlighted connection, you can save the recorded message for future use.

☐ Do not show this message again

Close

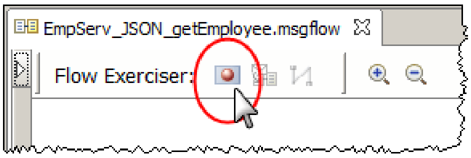
4. The canvas background in the message flow editor will grey out, indicating that the message flow is ready to record data sent through it:



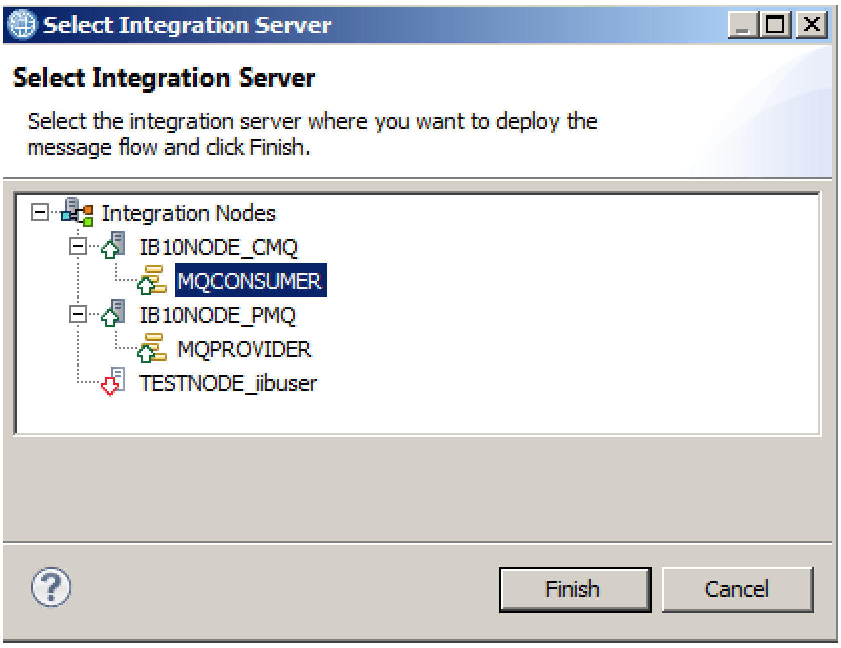
getEmployeeDetails.msgflow
JSON_MQClient.msgflow
Flow Exerciser:
getEmployeeDetails
Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.
Read Request
getEmployeeDetailsFromdB
Write Response
MQ Provider Trace

5. Switch to the open message flow JSON_MQClient.msgflow.

Start the flow Exerciser.



6. When prompted where to deploy the message flow, select **MQCONSUMER** on **IB10NODE_CMQ**:



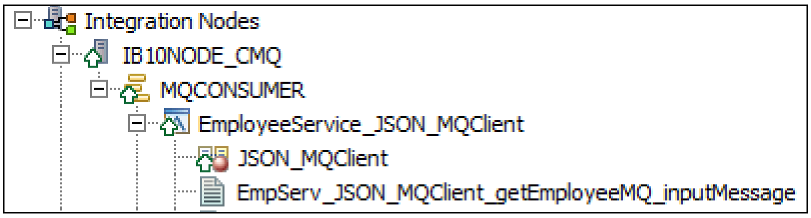
Select Integration Server

Select the integration server where you want to deploy the message flow and click Finish.

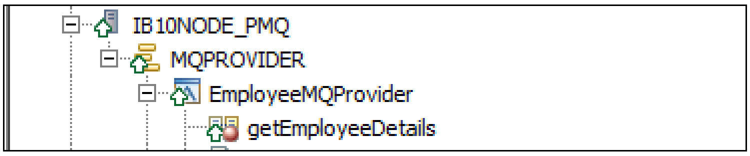
- Integration Nodes
  - IB10NODE_CMQ
    - MQCONSUMER
  - IB10NODE_PMQ
    - MQPROVIDER
  - TESTNODE_iibuser

7. If the "Ready to record message" appears, dismiss it.

8. Note that **both** message flows (in both IIB nodes) will be now be shown to be in record mode in the Integration Nodes view:

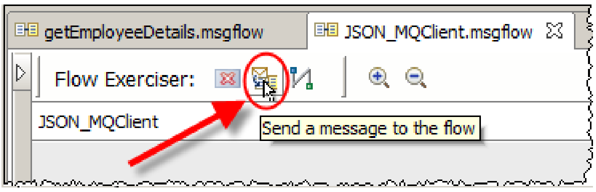


- Integration Nodes
  - IB10NODE_CMQ
    - MQCONSUMER
      - EmployeeService_JSON_MQClient
        - JSON_MQClient
        - EmpServ_JSON_MQClient_getEmployeeMQ_inputMessage

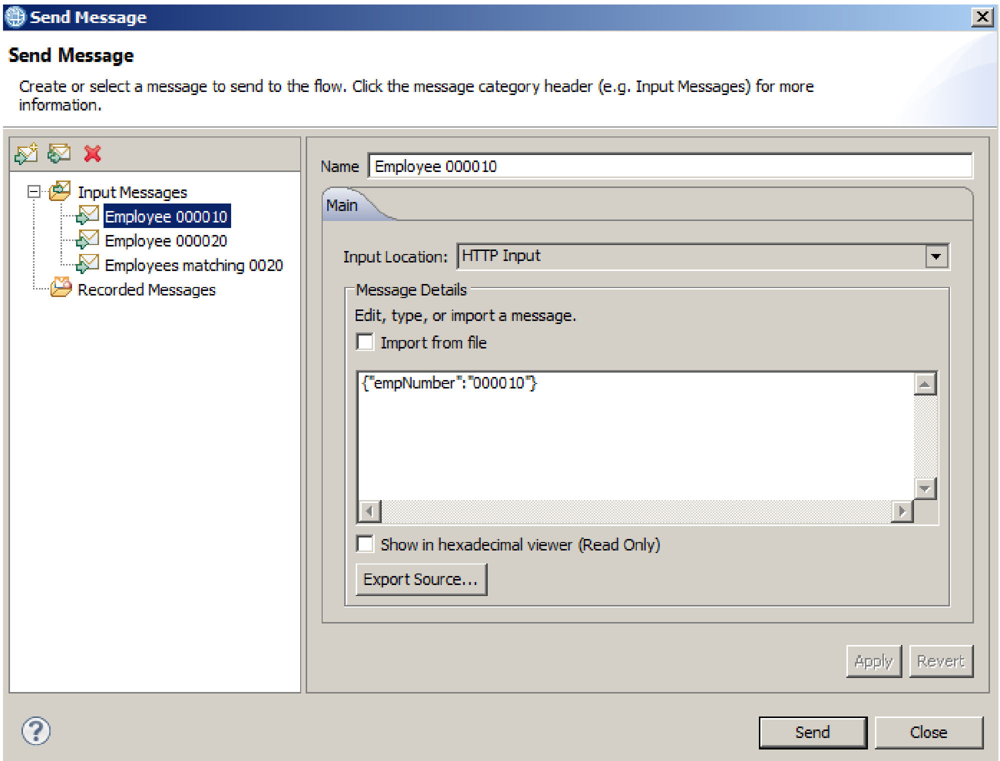


- IB10NODE_PMQ
  - MQPROVIDER
    - EmployeeMQProvider
      - getEmployeeDetails

9. In the JSON_MQClient.msgflow, click the "Send a message to the flow":



10. In the Send Message window, expand Input Messages and highlight “User 000010” and click Send. (This will inject a message into the message flow exercising the nodes in the flow. Note the message details contains the JSON to find user with a key of “000010”):



Send Message

Create or select a message to send to the flow. Click the message category header (e.g. Input Messages) for more information.

Input Messages
- Employee 000010
- Employee 000020
- Employees matching 0020

Recorded Messages

Name: Employee 000010

Input Location: HTTP Input

Message Details: Edit, type, or import a message.

☐ Import from file

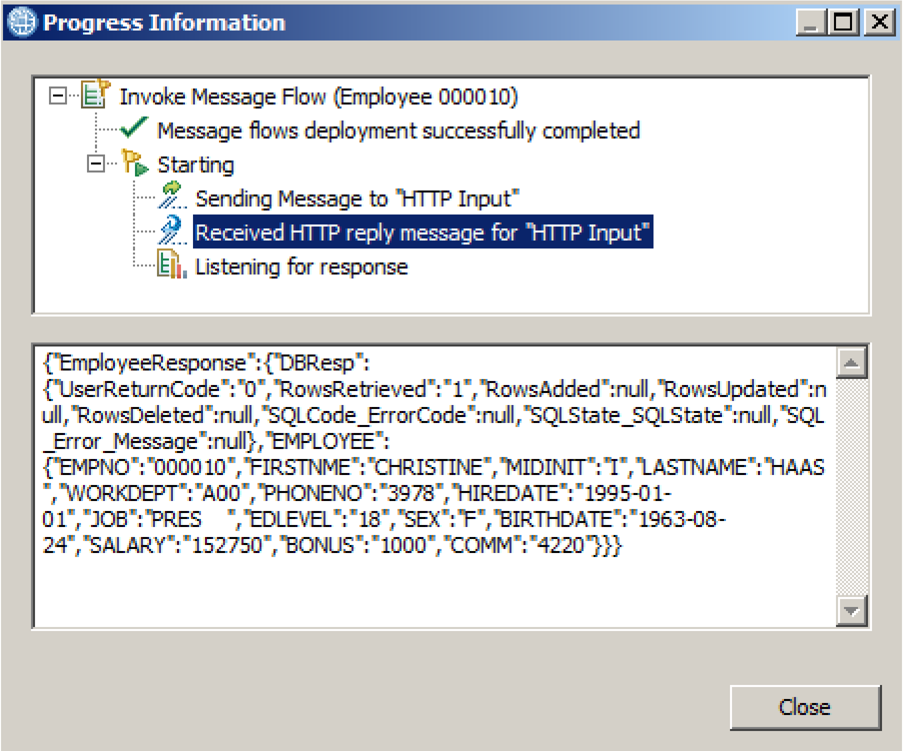
```
{"empNumber":"000010"}
```

☐ Show in hexadecimal viewer (Read Only)

11. The “Progress Information” window will open. When the line “Received HTTP reply message for HTTP Input” appears, the flow has completed.

Highlight the line to see the received data.

You will see the response from the database look up using the key “000010”:



Invoke Message Flow (Employee 000010)
- Message flows deployment successfully completed
- Starting
  - Sending Message to "HTTP Input"
  - Received HTTP reply message for "HTTP Input"
  - Listening for response

```
{"EmployeeResponse":{"DBResp":
{"UserReturnCode":"0","RowsRetrieved":"1","RowsAdded":null,"RowsUpdated":null,"RowsDeleted":null,"SQLCode_ErrorCode":null,"SQLState_SQLState":null,"SQL_Error_Message":null},"EMPLOYEE":
{"EMPNO":"000010","FIRSTNME":"CHRISTINE","MIDINIT":"I","LASTNAME":"HAAS","WORKDEPT":"A00","PHONENO":"3978","HIREDATE":"1995-01-01","JOB":"PRES    ","EDLEVEL":"18","SEX":"F","BIRTHDATE":"1963-08-24","SALARY":"152750","BONUS":"1000","COMM":"4220"}}}
```

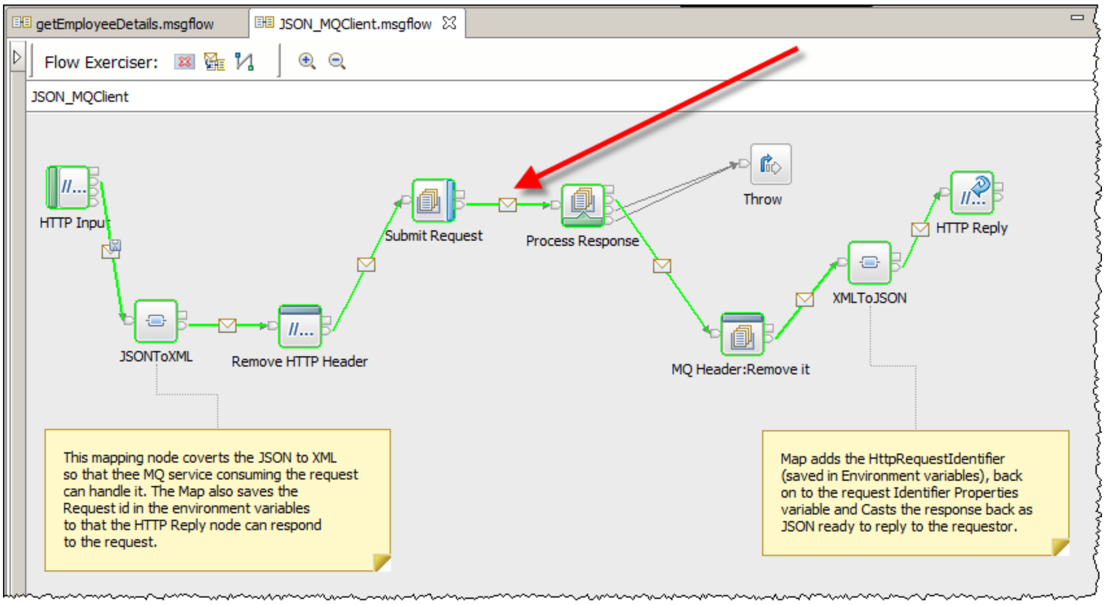
5.3 Verify the connection to the queue manager (server connection)

You will now use the Flow Exerciser to verify that the queue being used is defined on QM2.

1. Close the Progress Information window.

   The message flow will now show the route the message took through the message flow (a line in green).

   Click the envelope icon between the MQOutput node “Submit Request” and the MQGet node “Process Response”:

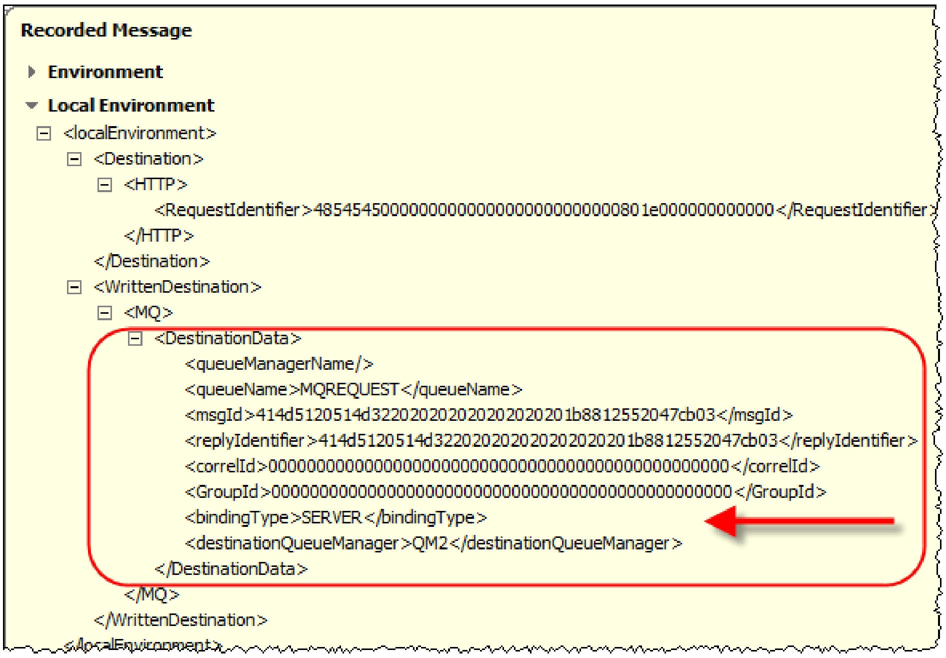


This mapping node coverts the JSON to XML so that thee MQ service consuming the request can handle it. The Map also saves the Request id in the environment variables to that the HTTP Reply node can respond to the request.

Map adds the HttpRequestIdentifier (saved in Environment variables), back on to the request Identifier Properties variable and Casts the response back as JSON ready to reply to the requestor.

2. When the “Recorded Message” (yellow background) popup appears, expand **Local Environment.**

   Note the details in “DestinationData” reflect the properties saved with the MQOutput node “Submit Response” :



Recorded Message

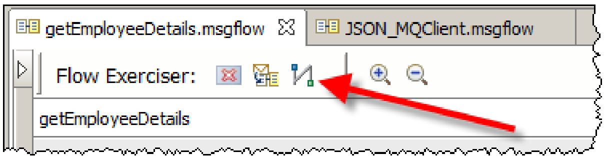
Environment

Local Environment

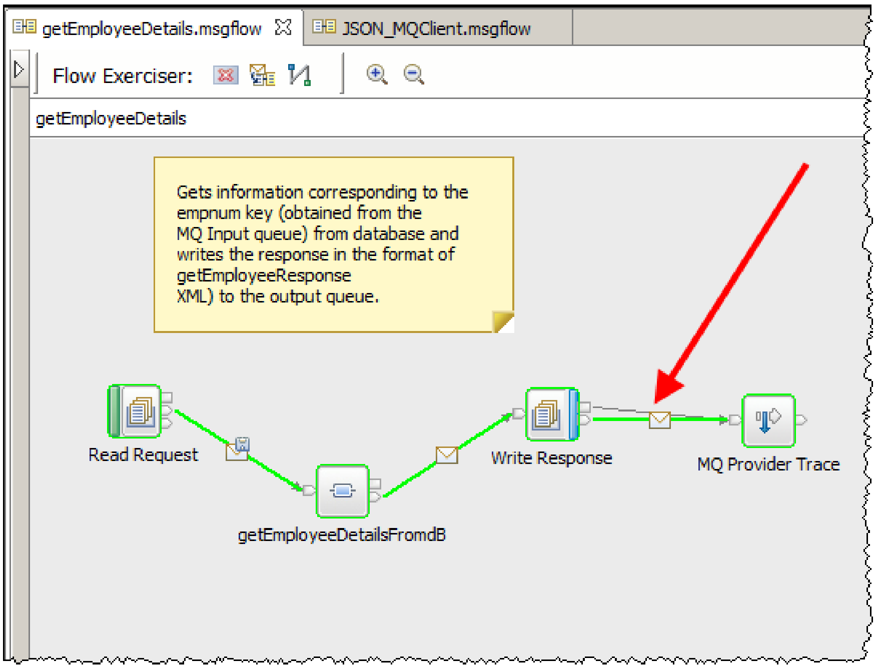
```
<localEnvironment>
  <Destination>
    <HTTP>
      <RequestIdentifier>48545450000000000000000000000000801e000000000000</RequestIdentifier>
    </HTTP>
  </Destination>
  <WrittenDestination>
    <MQ>
      <DestinationData>
        <queueManagerName/>
        <queueName>MQREQUEST</queueName>
        <msgId>414d5120514d322020202020202020201b8812552047cb03</msgId>
        <replyIdentifier>414d5120514d322020202020202020201b8812552047cb03</replyIdentifier>
        <correlId>000000000000000000000000000000000000000000000000</correlId>
        <GroupId>000000000000000000000000000000000000000000000000</GroupId>
        <bindingType>SERVER</bindingType>
        <destinationQueueManager>QM2</destinationQueueManager>
      </DestinationData>
    </MQ>
  </WrittenDestination>
</localEnvironment>
```

   The MQ Client application has written the data to the MQREQUEST queue using a **SERVER** (Bindings) connection to **QM2**.

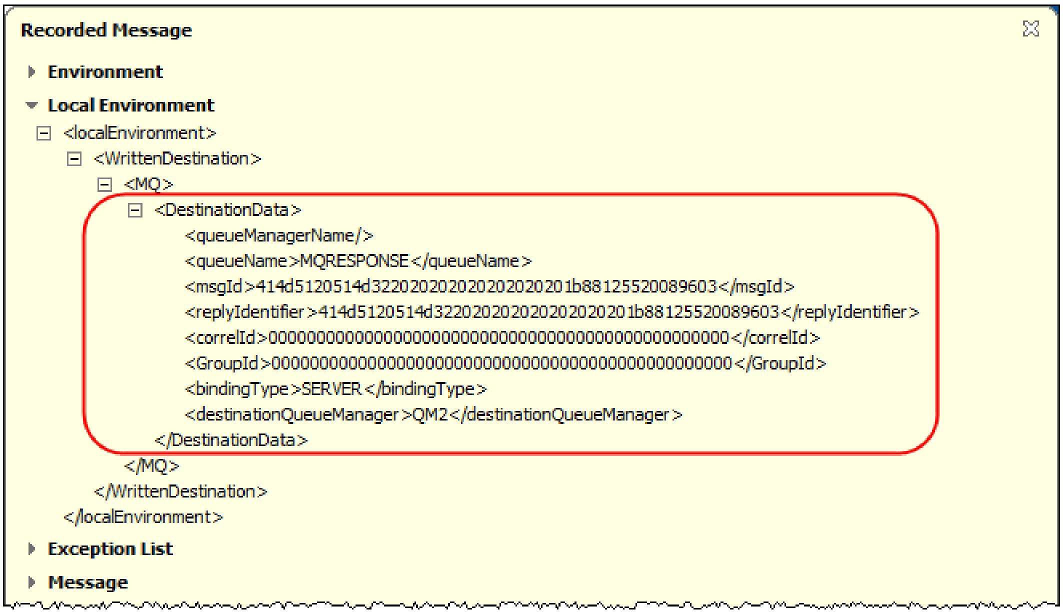
3. Switch to the getEmployeeDetails message flow and click the “**Show message path through the message flow**” button:



4. The Flow Exerciser will show the path that the message took through this flow in green. Click the envelope between the MQOutput node and the Trace node.


getEmployeeDetails.msgflow
JSON_MQClient.msgflow
Flow Exerciser:
getEmployeeDetails
Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.
Read Request
getEmployeeDetailsFromdB
Write Response
MQ Provider Trace

5. In the Recorded Message window, expand "Local Environment" to see the details of where the application wrote the MQ details to:



Recorded Message

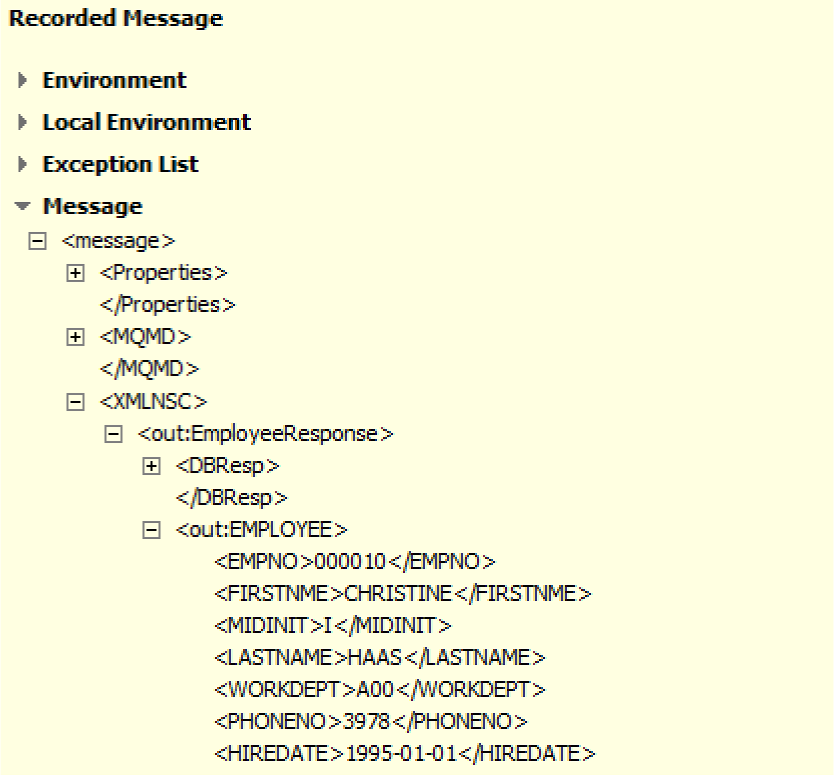
- **Environment**
- **Local Environment**

```
<localEnvironment>
  <WrittenDestination>
    <MQ>
      <DestinationData>
        <queueManagerName/>
        <queueName>MQRESPONSE</queueName>
        <msgId>414d5120514d322020202020202020201b88125520089603</msgId>
        <replyIdentifier>414d5120514d322020202020202020201b88125520089603</replyIdentifier>
        <correlId>000000000000000000000000000000000000000000000000</correlId>
        <GroupId>000000000000000000000000000000000000000000000000</GroupId>
        <bindingType>SERVER</bindingType>
        <destinationQueueManager>QM2</destinationQueueManager>
      </DestinationData>
    </MQ>
  </WrittenDestination>
</localEnvironment>
```

- **Exception List**
- **Message**

As with the JSON_MQClient flow, the MQProvider application also wrote directly to the MQRESPONSE queue on **QM2**, using **SERVER** bindings.

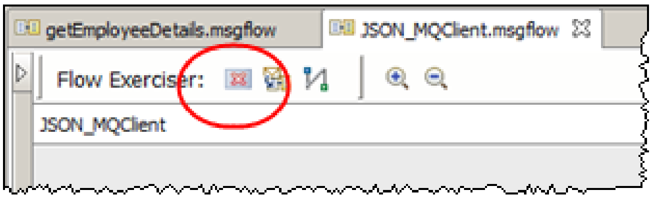
6. Expand the Message section (to XMLNSC) to show the details returned from the getEmployee_XML map.



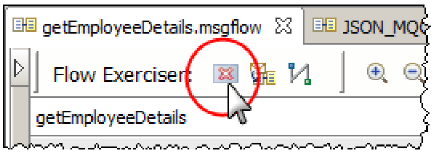
```
Recorded Message

▸ Environment
▸ Local Environment
▸ Exception List
▾ Message
  ⊟ <message>
      ⊞ <Properties>
        </Properties>
      ⊞ <MQMD>
        </MQMD>
      ⊟ <XMLNSC>
          ⊟ <out:EmployeeResponse>
              ⊞ <DBResp>
                </DBResp>
              ⊟ <out:EMPLOYEE>
                  <EMPNO>000010</EMPNO>
                  <FIRSTNME>CHRISTINE</FIRSTNME>
                  <MIDINIT>I</MIDINIT>
                  <LASTNAME>HAAS</LASTNAME>
                  <WORKDEPT>A00</WORKDEPT>
                  <PHONENO>3978</PHONENO>
                  <HIREDATE>1995-01-01</HIREDATE>
```

7. If the Flow Exerciser is recording messages, it is not possible to edit a message flow. In the next section you will re-configure the MQ nodes in the message flow.

8. Stop the Flow Exerciser recording messages in the JSON_MQClient.msgflow:



9. Stop the Flow Exerciser recording messages in getEmployeeDetails.msgflow



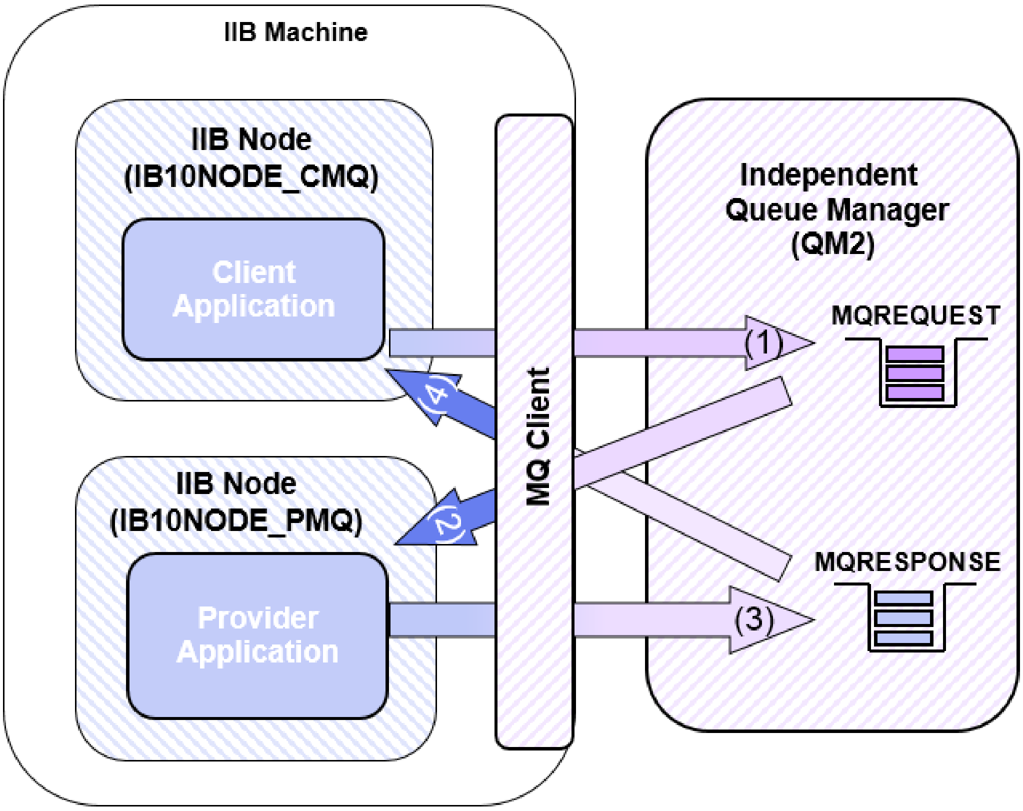
6. Scenario 3: IIB node with no associated queue manager: Using an MQ CLIENT connection

You will now configure the MQ applications to access the queues on queue manager (QM2) using a Client Connection.

An MQ Client "server conn" definition and a TCP listener (running on port 1442) have already been defined on QM2.

The configuration for this scenario is similar to the previous scenario, however you will configure the MQ Connection properties for both the applications to use an MQ Client connection. The MQ Client is already installed on your prebuilt environment.

You will configure the environment to simulate the following (*note the prebuilt Lab environment all IIB nodes and queue managers are located in a single virtual machine*):



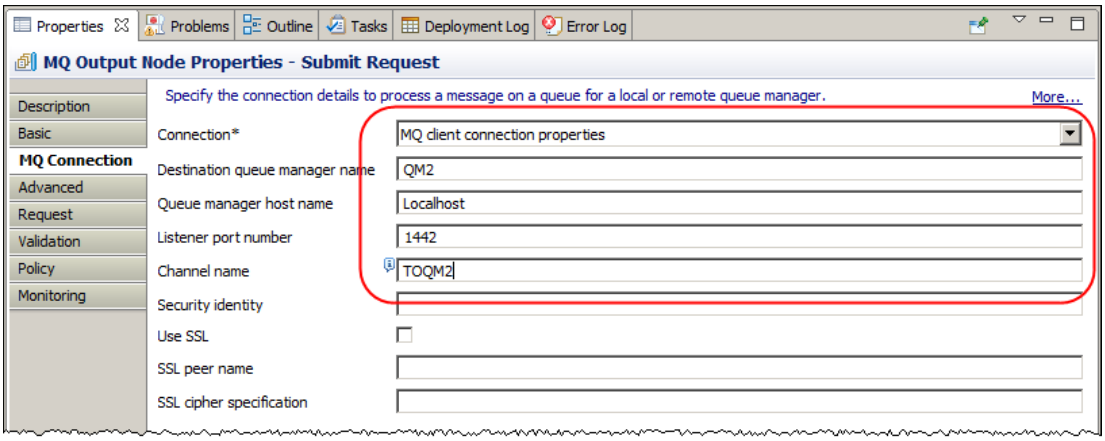
6.1 Configure the MQ nodes to use an MQ client connection

6.1.1 Configure the MQ Client application

1. In the JSON_MQClient message flow, click the **MQOutput node** ("Submit Request") to show the node properties in the properties tab.

   Specify the following connections properties for the node:

• Connection*	: **`MQ client connection properties`**
• Destination queue manager name	: **`QM2`**
• Queue manager host name	: **`localhost`**
• Listener port number	: **`1442`**
• Channel name	: **`TOQM2`**



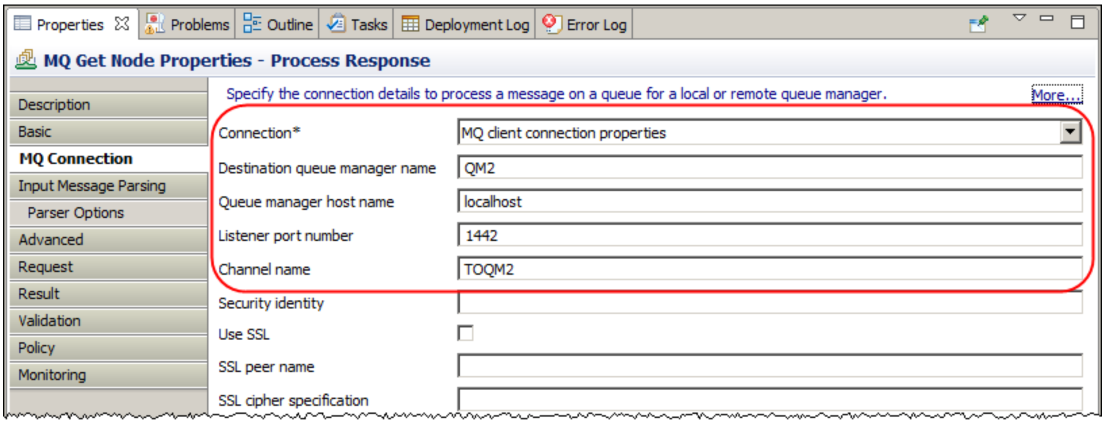
MQ Output Node Properties - Submit Request

Specify the connection details to process a message on a queue for a local or remote queue manager.

Connection*	MQ client connection properties
Destination queue manager name	QM2
Queue manager host name	Localhost
Listener port number	1442
Channel name	TOQM2
Security identity	
Use SSL	□
SSL peer name	
SSL cipher specification	

2. Repeat the above step for the **MQGet node** called "Process Response".

   The properties tab should look like this (the same as above):



MQ Get Node Properties - Process Response

Specify the connection details to process a message on a queue for a local or remote queue manager.

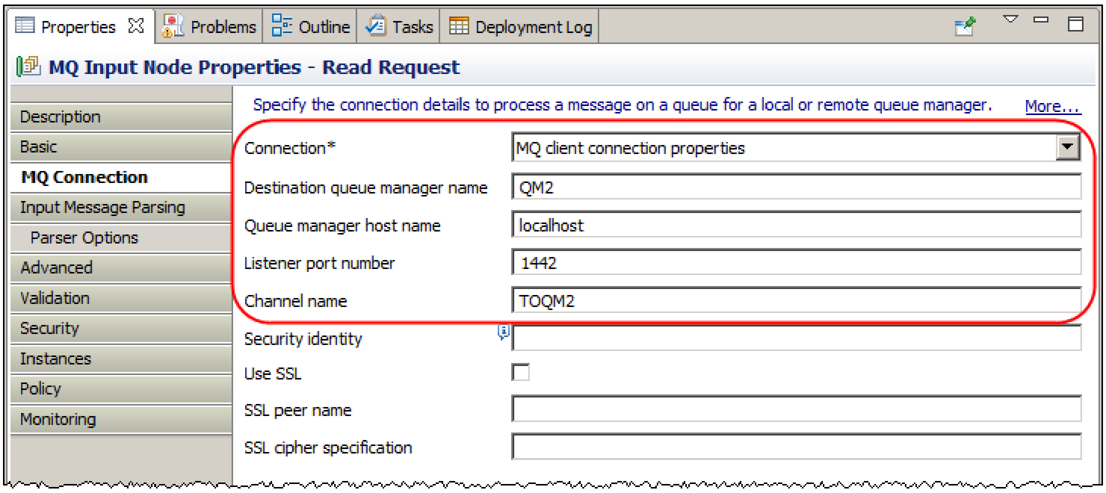
Connection*	MQ client connection properties
Destination queue manager name	QM2
Queue manager host name	localhost
Listener port number	1442
Channel name	TOQM2
Security identity	
Use SSL	□
SSL peer name	
SSL cipher specification	

3. Save the message flow (ctrl s)

6.1.2 Configure the MQ Provider application

1. In the getEmployeeDetails message flow, click the “Read Request” MQInput node to show the nodes properties in the properties tab.

   Specify the following client connection properties in the MQ Connection tab:

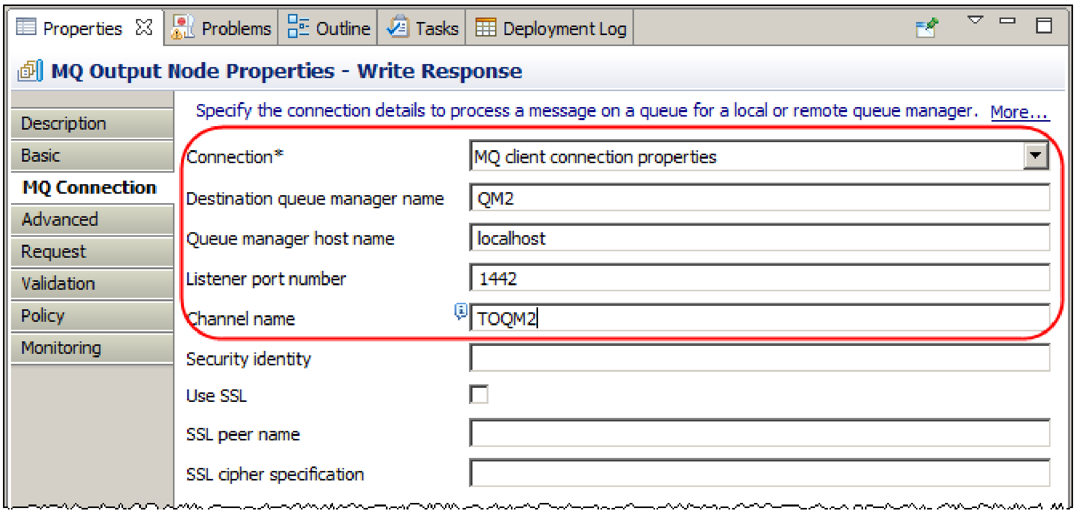


MQ Input Node Properties - Read Request

Specify the connection details to process a message on a queue for a local or remote queue manager.

Connection* MQ client connection properties
Destination queue manager name QM2
Queue manager host name localhost
Listener port number 1442
Channel name TOQM2
Security identity
Use SSL □
SSL peer name
SSL cipher specification

2. Click the MQOutput node “Write Response” and configure the MQ Connection properties as above:



MQ Output Node Properties - Write Response

Specify the connection details to process a message on a queue for a local or remote queue manager.

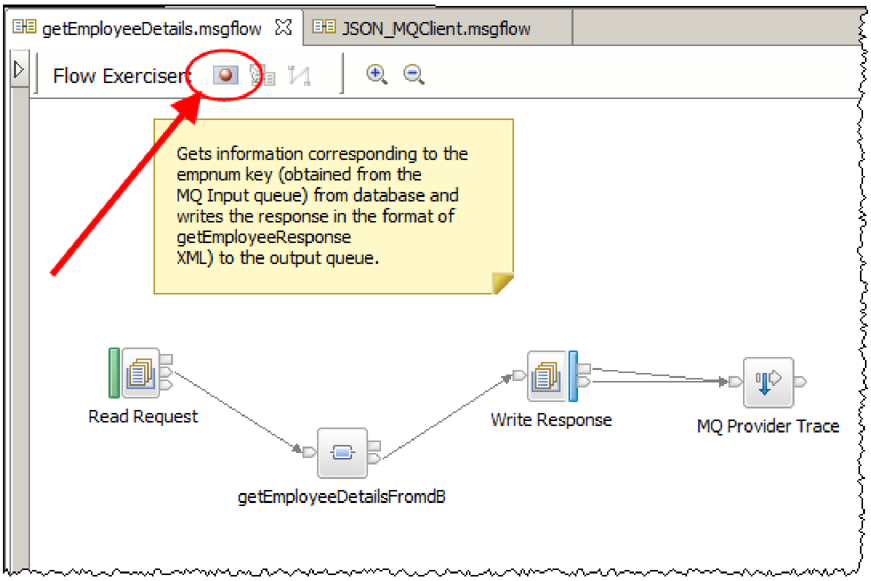
Connection* MQ client connection properties
Destination queue manager name QM2
Queue manager host name localhost
Listener port number 1442
Channel name TOQM2
Security identity
Use SSL □
SSL peer name
SSL cipher specification

3. Save the message flow (ctrl s)

6.2 Test the scenario using the Flow Exerciser

You will now test the applications and verify the connection the type of connection to the queue manager QM2.

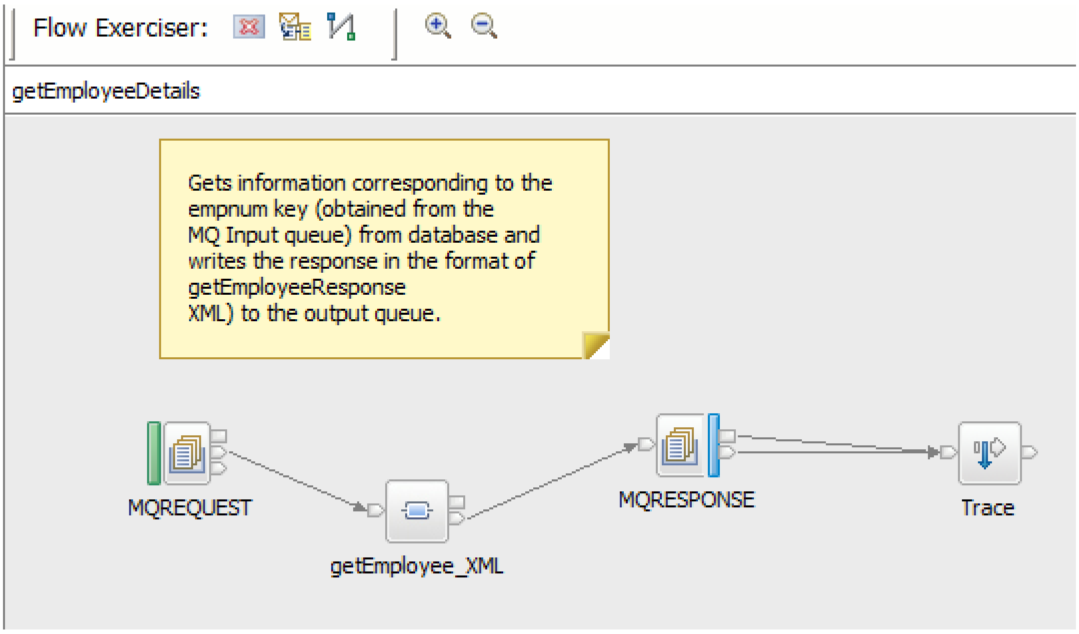
1. In the **getEmployeeDetails.msgflow**, click the Flow Exerciser button:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

2. If prompted, deploy to **MQPROVIDER** on **IB10NODE_PMQ**:

3. The canvas background in the message flow editor will grey out, indicating that the message flow is ready to record data sent through it:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

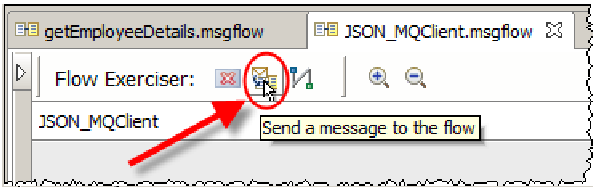
4. Now switch to the message flow JSON_MQClient.msgflow.

   Click the record button.

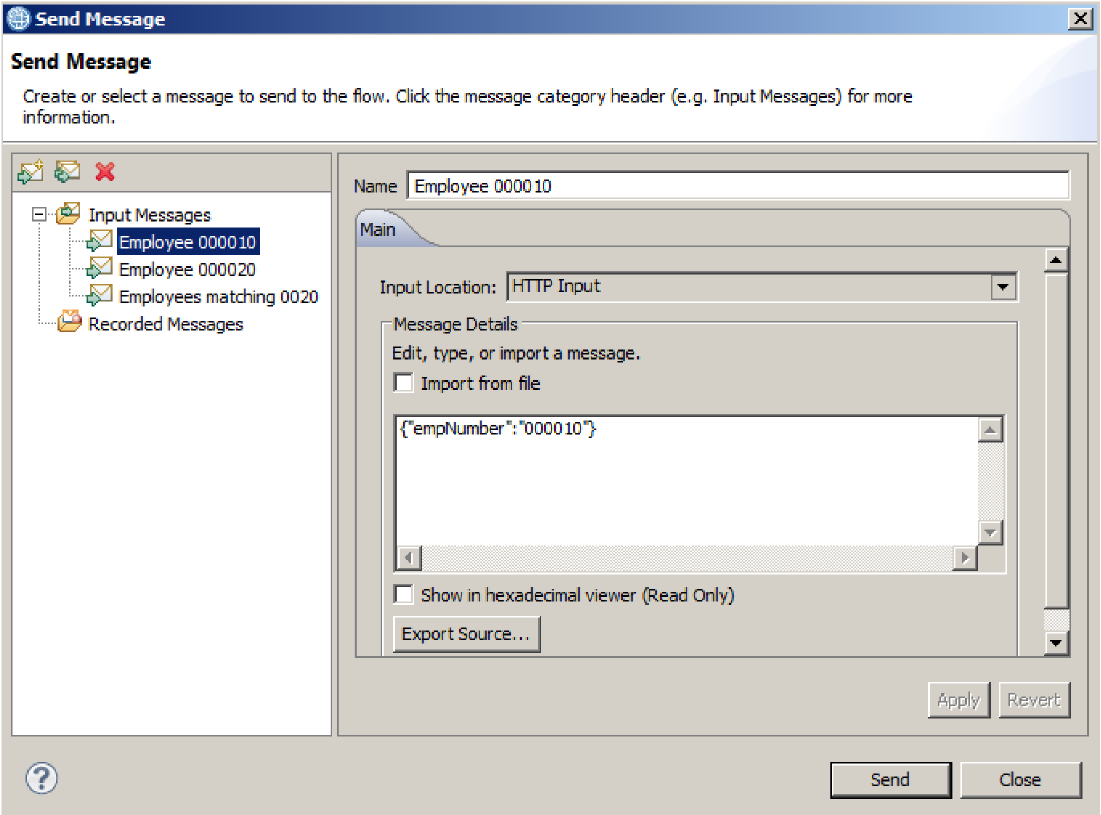
   Deploy the updated application to **MQCONSUMER** on **IB10NODE_CMQ.**

5. If the “Ready to record message” appears, dismiss it.

6. In the JSON_MQClient.msgflow, click the “Send a message to the flow”:



7. In the Send Message window, expand Input Messages and highlight “User 000010” and click Send:


Send Message

Send Message

Create or select a message to send to the flow. Click the message category header (e.g. Input Messages) for more information.

Name: Employee 000010

Input Location: HTTP Input

Message Details

Edit, type, or import a message.

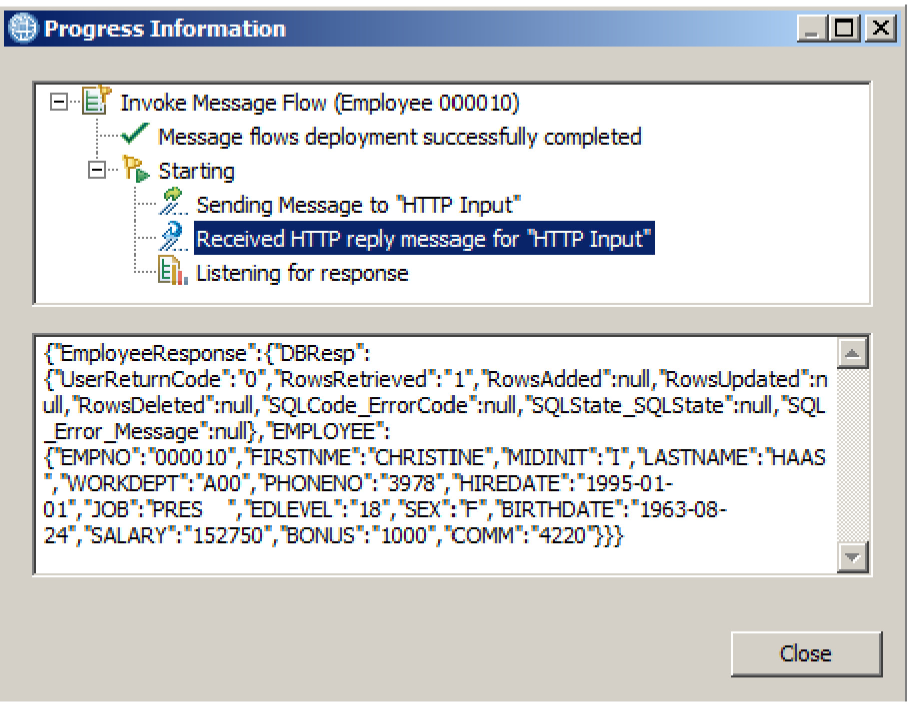
☐ Import from file

```
{"empNumber":"000010"}
```

☐ Show in hexadecimal viewer (Read Only)

8. The “Progress Information” window will open.

   When the line “Received HTTP reply message for HTTP Input” appears, highlight the line to see the received data. You will see the response from the database look up using the key “000010”:



```
Invoke Message Flow (Employee 000010)
    Message flows deployment successfully completed
    Starting
        Sending Message to "HTTP Input"
        Received HTTP reply message for "HTTP Input"
        Listening for response
```

```
{"EmployeeResponse":{"DBResp":
{"UserReturnCode":"0","RowsRetrieved":"1","RowsAdded":null,"RowsUpdated":n
ull,"RowsDeleted":null,"SQLCode_ErrorCode":null,"SQLState_SQLState":null,"SQL
_Error_Message":null},"EMPLOYEE":
{"EMPNO":"000010","FIRSTNME":"CHRISTINE","MIDINIT":"I","LASTNAME":"HAAS
","WORKDEPT":"A00","PHONENO":"3978","HIREDATE":"1995-01-
01","JOB":"PRES    ","EDLEVEL":"18","SEX":"F","BIRTHDATE":"1963-08-
24","SALARY":"152750","BONUS":"1000","COMM":"4220"}}}
```

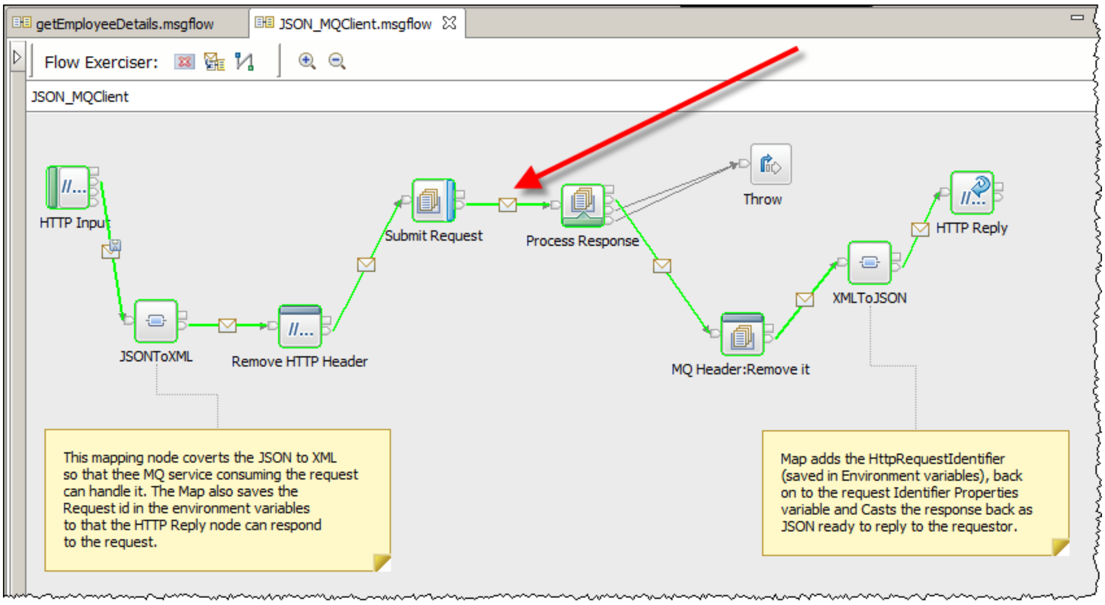
6.3 Verify the queue manager connection (client connection)

You will now use the Flow Exerciser to verify that the connection details used by the runtime environment.

1. Close the Progress Information window.

   The message flow will show the route the message took through the message flow (a line in green).

   Click the envelope icon between the "Submit Request" MQOutput node and the MQGet node called "Process Response":

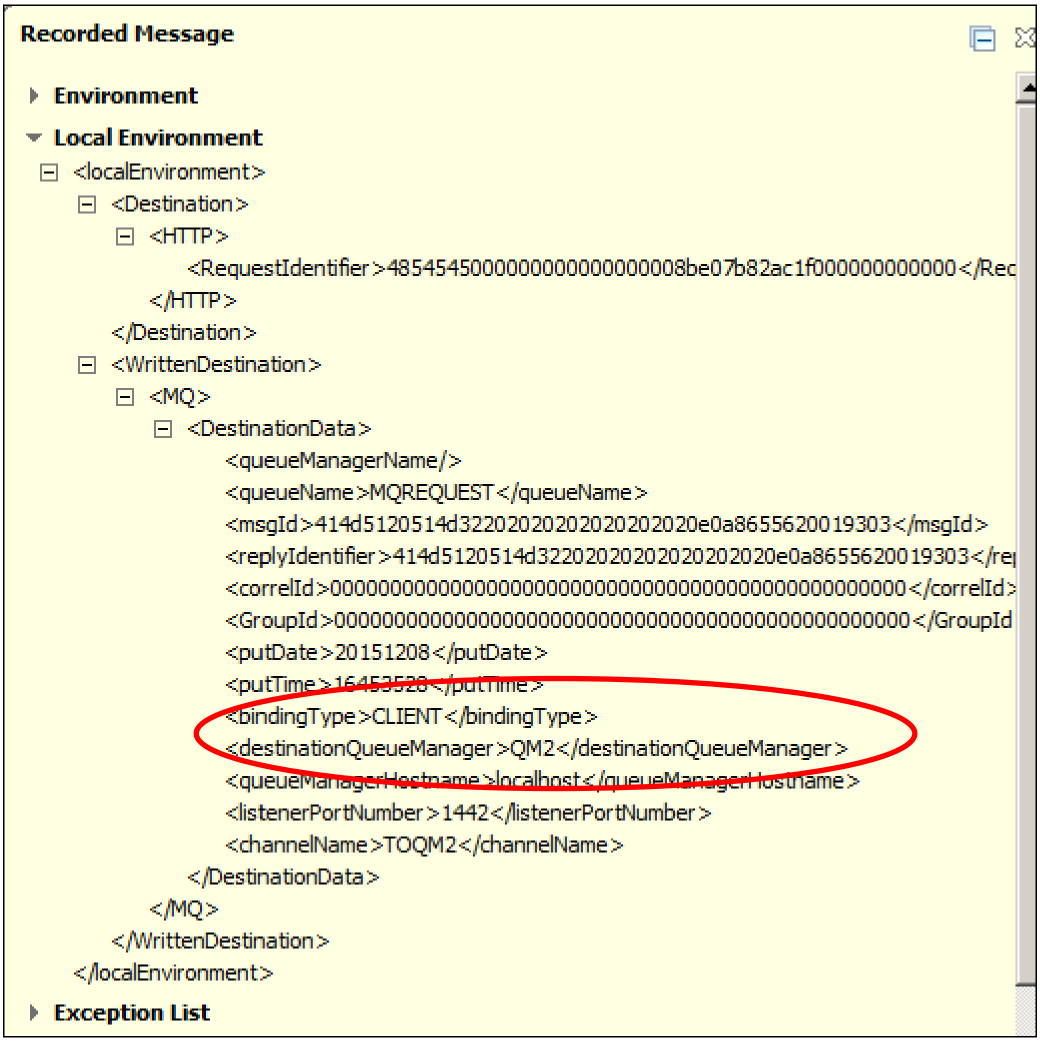


This mapping node coverts the JSON to XML so that thee MQ service consuming the request can handle it. The Map also saves the Request id in the environment variables to that the HTTP Reply node can respond to the request.

Map adds the HttpRequestIdentifier (saved in Environment variables), back on to the request Identifier Properties variable and Casts the response back as JSON ready to reply to the requestor.

2. When the “Recorded Message” (yellow background) popup appears, expand **Local Environment.**

   Note the details in “DestinationData” reflect the properties saved with the MQOutput node “Submit Response”:



Recorded Message

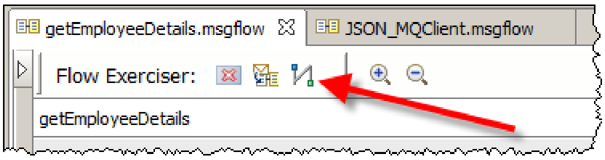
- Environment
- Local Environment

```
<localEnvironment>
  <Destination>
    <HTTP>
      <RequestIdentifier>4854545000000000000000008be07b82ac1f000000000000</Rec
    </HTTP>
  </Destination>
  <WrittenDestination>
    <MQ>
      <DestinationData>
        <queueManagerName/>
        <queueName>MQREQUEST</queueName>
        <msgId>414d5120514d322020202020202020202020e0a8655620019303</msgId>
        <replyIdentifier>414d5120514d322020202020202020202020e0a8655620019303</re
        <correlId>000000000000000000000000000000000000000000000000</correlId>
        <GroupId>000000000000000000000000000000000000000000000000</GroupId
        <putDate>20151208</putDate>
        <putTime>16453528</putTime>
        <bindingType>CLIENT</bindingType>
        <destinationQueueManager>QM2</destinationQueueManager>
        <queueManagerHostname>localhost</queueManagerHostname>
        <listenerPortNumber>1442</listenerPortNumber>
        <channelName>TOQM2</channelName>
      </DestinationData>
    </MQ>
  </WrittenDestination>
</localEnvironment>
```

- Exception List

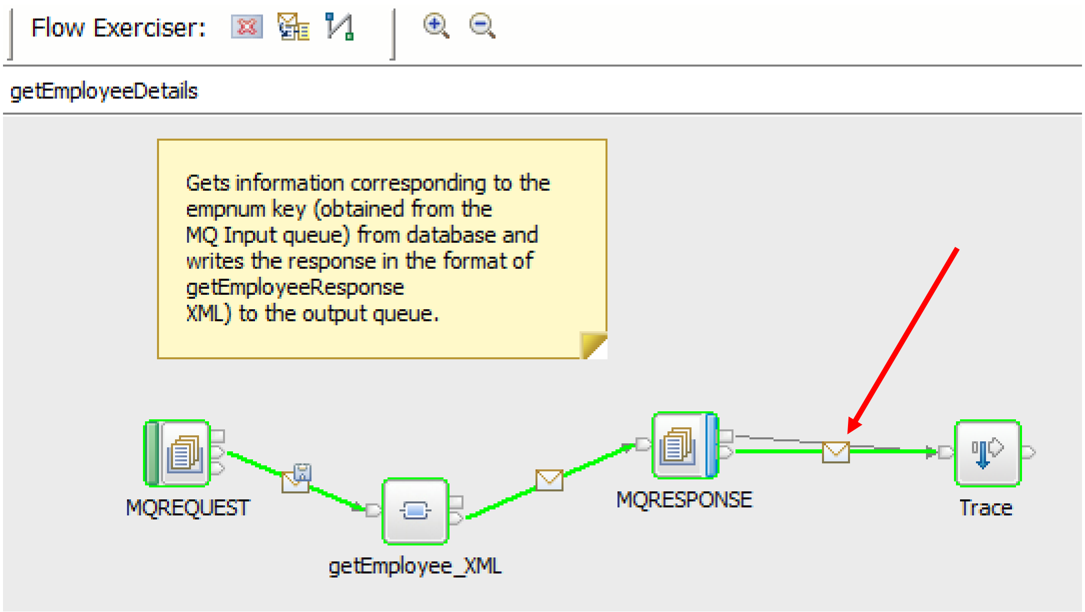
The MQ Client application has written the data to the MQREQUEST queue using a **CLIENT** connection to **QM2**.

3. Switch to the getEmployeeDetails message flow and click the show message path through the message flow button:



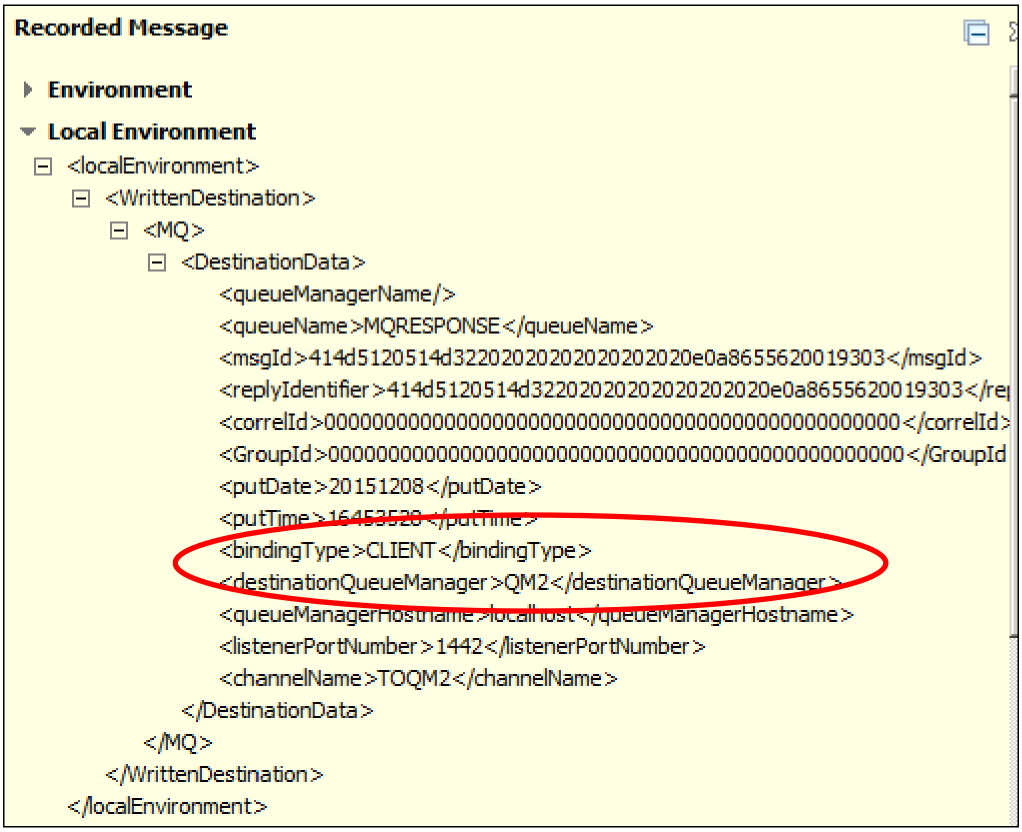
4. The Flow Exerciser will show the path that the message took through this flow in green.

   Click on the envelope between the MQOutput node and the MQ Provider trace:



Gets information corresponding to the empnum key (obtained from the MQ Input queue) from database and writes the response in the format of getEmployeeResponse XML) to the output queue.

5. In the Recorded Message window, expand "Local Environment" to see the details of where the application wrote the MQ details to:



Recorded Message

- Environment
- Local Environment

```
<localEnvironment>
  <WrittenDestination>
    <MQ>
      <DestinationData>
        <queueManagerName/>
        <queueName>MQRESPONSE</queueName>
        <msgId>414d5120514d3220202020202020202020e0a8655620019303</msgId>
        <replyIdentifier>414d5120514d3220202020202020202020e0a8655620019303</re
        <correlId>000000000000000000000000000000000000000000000000</correlId>
        <GroupId>000000000000000000000000000000000000000000000000</GroupId
        <putDate>20151208</putDate>
        <putTime>16453528</putTime>
        <bindingType>CLIENT</bindingType>
        <destinationQueueManager>QM2</destinationQueueManager>
        <queueManagerHostname>localhost</queueManagerHostname>
        <listenerPortNumber>1442</listenerPortNumber>
        <channelName>TOQM2</channelName>
      </DestinationData>
    </MQ>
  </WrittenDestination>
</localEnvironment>
```

The MQProvider application also wrote the message to the MQRESPONSE queue using a client connection to QM2.

END OF LAB GUIDE