16L42

IBM Integration Bus

A JSON client Application invoking an Integration Service

Featuring:

Graphical Data Mapper with schemaless mapping JSON input/output
Mapping Node - element concatenation and Custom XPath
Mapping Node - accessing the Environment tree
Mapping Node - copying / pasting complex elements
Mapping Node - accessing IIB node runtime information
Unit Test with the Flow Exerciser
Invoking web services asynchronously

January 2016

Hands-on lab built at product Version 10.0.0.3

1. INTRODUCTION 3

1.1 SCENARIO 3
1.2 PREPARE THE IIB NODE 4
1.3 OPEN THE WINDOWS LOG MONITOR FOR IIB 4
1.4 IMPORT AND DEPLOY EMPLOYEESERVICE 5

2. CREATE THE CLIENT APPLICATION 6

2.1 IMPLEMENT THE MAPPING NODES 12
2.1.1 Implement the JSON_to_SOAP map 12
2.1.2 Implement the XML_to_JSON map 18
2.2 COMPLETE THE MESSAGE FLOW 25

3. TEST THE EMPLOYEESERVICE_JSONCLIENT APPLICATION 27

3.1 TEST USING THE FLOW EXERCISER 27
3.2 TEST USING SOAPUI AND THE FLOW EXERCISER 36

4. ADDING DATA TO THE ENVIRONMENT TREE (OPTIONAL) 41

4.1 CONFIGURE THE FIRST MAPPING NODE 41
4.2 ADD AND CONFIGURE NEW NODES 45
4.2.1 Configure the Route node 46
4.2.2 Configure the new Mapping node 47
4.2.3 Create the mapping transforms 50
4.2.4 Configure the File Output node 52
4.3 DEPLOY AND RETEST 54

5. ADD FAULT AND FAILURE HANDLING (OPTIONAL) 55

6. INVOKING THE EMPLOYEESERVICE INTEGRATION SERVICE ASYNCHRONOUSLY (OPTIONAL) 58

END OF LAB GUIDE 63

1. Introduction

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 Scenario

This lab guide shows you how to do the following tasks:

1. Use a web service connection to invoke an existing Integration Service (EmployeeService).
2. Use the Graphical Data Mapper to demonstrate schemaless mapping, mapping JSON input and output data.
3. Use the Graphical Data Mapper to write and read elements to the Environment tree.
4. Use the Graphical Data Mapper to retrieve information about the runtime environment, including the application and flow names.

The EmployeeService web service provider is already available for you, so the first task will be to import this prebuilt solution of EmployeeService, and deploy it to the broker node.

1.2 Prepare the IIB node

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

If you have already done Lab 1 in this series (create an Integration Service), you can skip straight to section 1.3.

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`

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

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

   `mqsistop TESTNODE_iibuser`

   `mqsistart TESTNODE_iibuser`

1.3 Open the Windows Log Monitor for IIB

A useful tool for IIB development on Windows is the IIB Log Viewer. This tool continuously monitors the Windows Event Log, and all messages from the log are displayed immediately.

From the Start menu, click IIB Event Log Monitor. The Monitor will open; it is useful to have this always open in the background.

1.4 Import and deploy EmployeeService

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_JSON`

2. Import the PI file

   `c:\student10\integration_service\solution\`
   `EmployeeService.10.0.0.3.zip`

   Import all projects in the PI file.

3. In the IIB Toolkit, make sure TESTNODE_iibuser is started (if not, right-click, Start).

   Stop any other active IIB nodes.

4. In the Toolkit navigator, expand the BARs folder, and deploy the following barfile to TESTNODE_iibuser/default:

   `EmployeeService.10.0.0.3.bar`

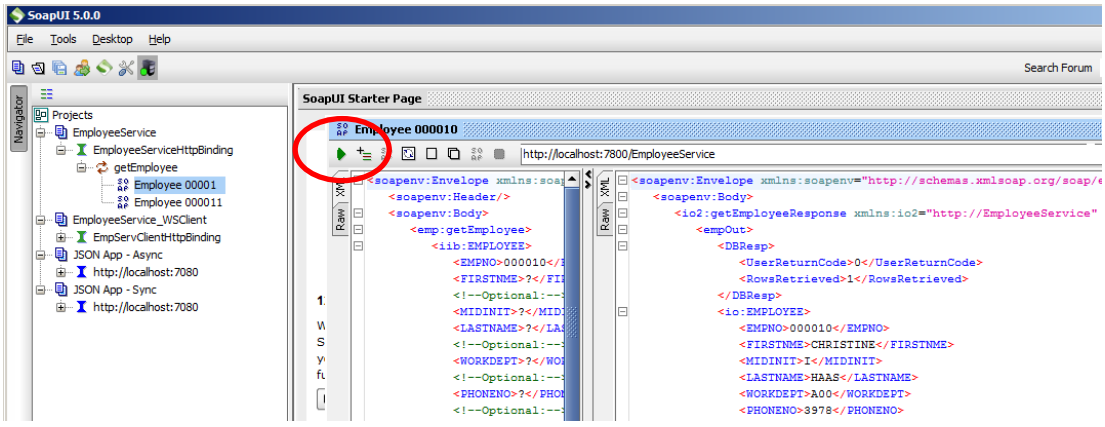
   This will overwrite any existing services of the same name in the default server.

5. As a quick test, to make sure the new EmployeeService is working, use SOAPUI to execute this service.

   In SOAPUI, fully expand the project EmployeeService.

   Open the test for 000010. Make sure the IP address is localhost, and that the port is 7800. Change to these values if necessary.

   Clicking the green arrow should return the following data. Note the number of rows retrieved is 1.



2. Create the Client Application

In this section you will create a new Application that will act as a client of the existing EmployeeService web service.

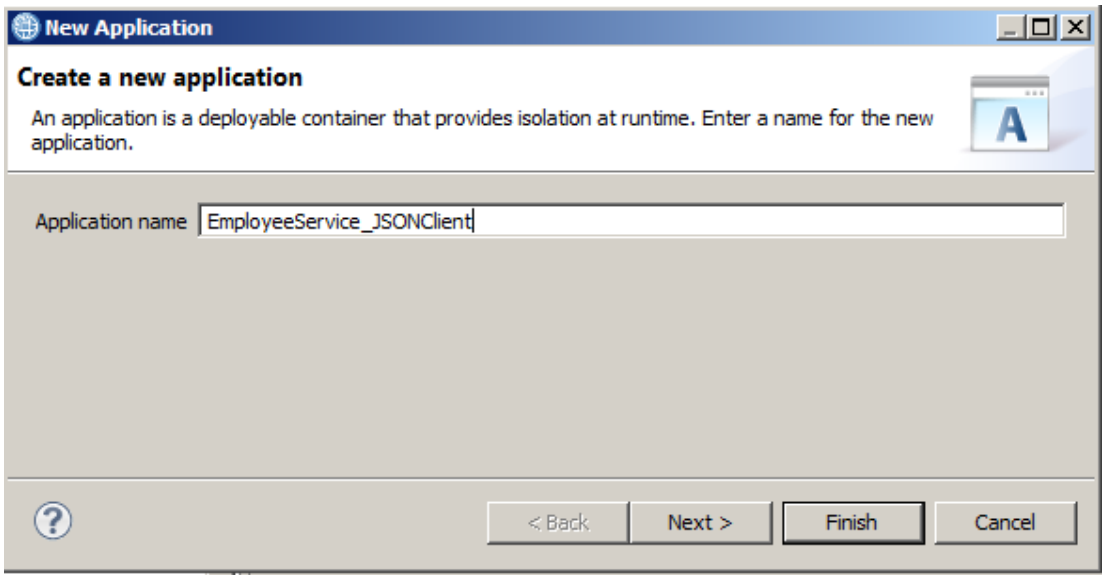
The EmployeeService uses the EMPLOYEE table of the HRDB database, and has implemented the getEmployee operation.

This new application, EmployeeService_JSONClient, will do the following functions:

- Receive a request to retrieve employee details in JSON format over HTTP
- Convert the JSON request to a SOAP message, and use this to retrieve the data by invoking the EmployeeService web service (getEmployee operation).
- Convert the response data from XML to JSON, and send the reply to the original client request.
- If the employee data is not found, write a "not found" record to a log file, in JSON format.

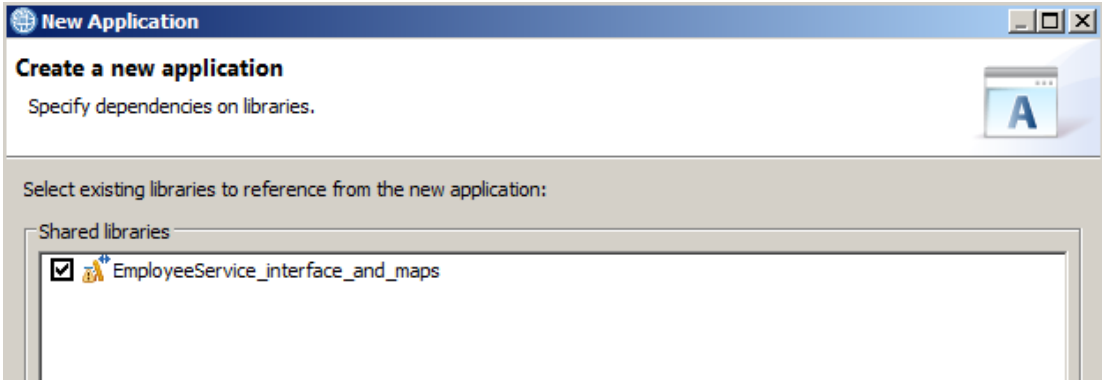
1. In the workspace, create a new Application named EmployeeService_JSONClient.

   Click Next.

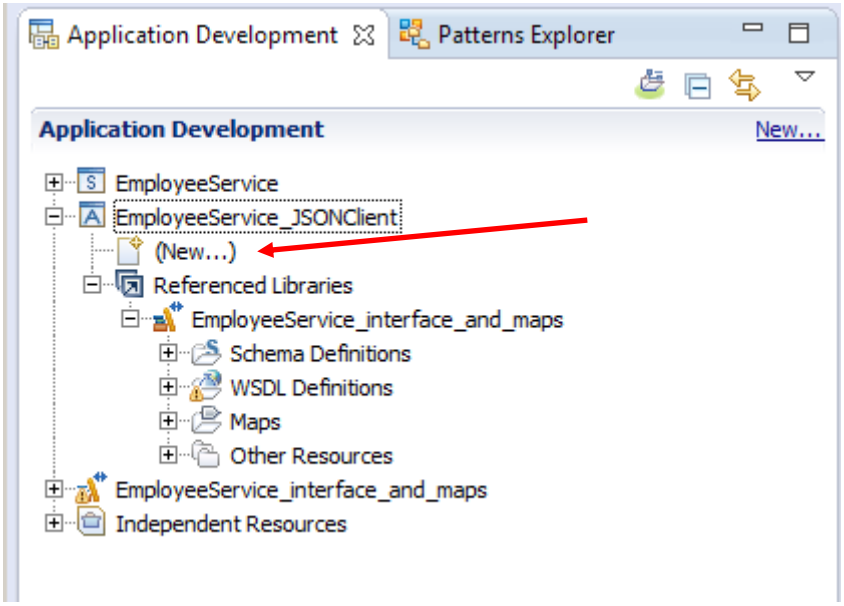


2. Select EmployeeService_interface_and_maps as a referenced Shared Library.

   Click Finish.

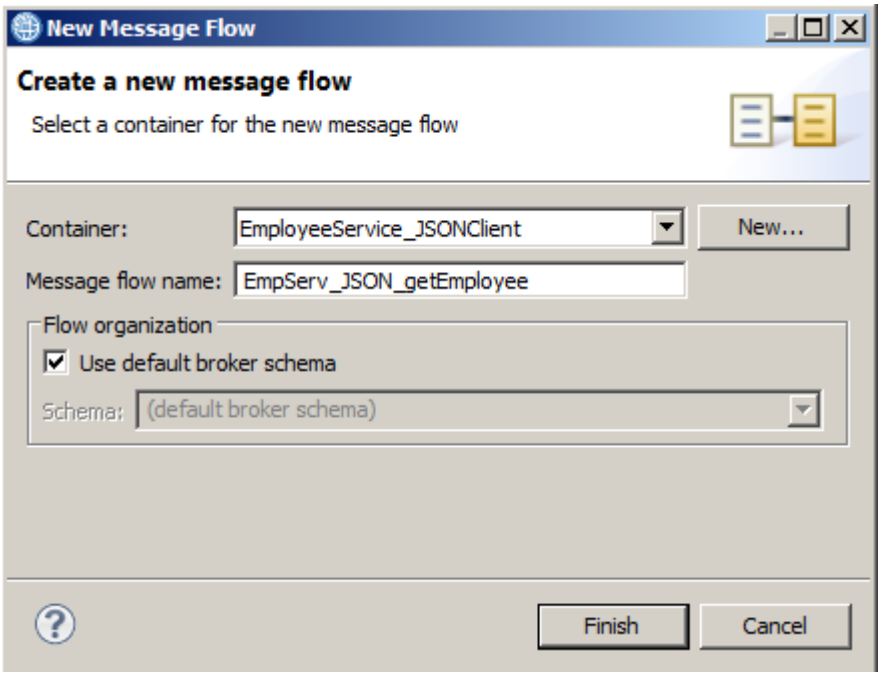


3. Click (New...) to create a new message flow.



4. Name the new flow EmpServ_JSON_getEmployee.

   Click Finish.



5. Drop the following nodes onto the flow editor, and name and connect them as shown.

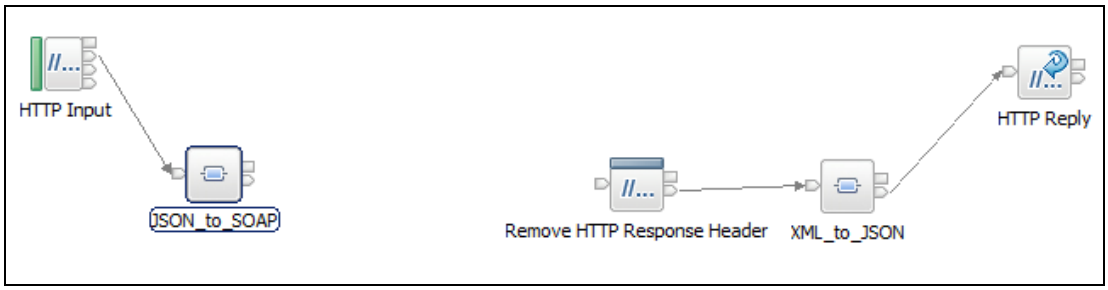
   - HTTP Input
   - Mapping Node - "JSON_to_SOAP"
   - HTTP Header node - "Remove HTTP Response Header"
   - Another Mapping Node - "XML_to_JSON"
   - HTTP Reply

   For the HTTP Input node:

   - Set the Basic URL Path Suffix property to **/empServClient_getEmployee (**the forward slash is important).
   - Set the Input Message Parsing Message Domain to **JSON**.

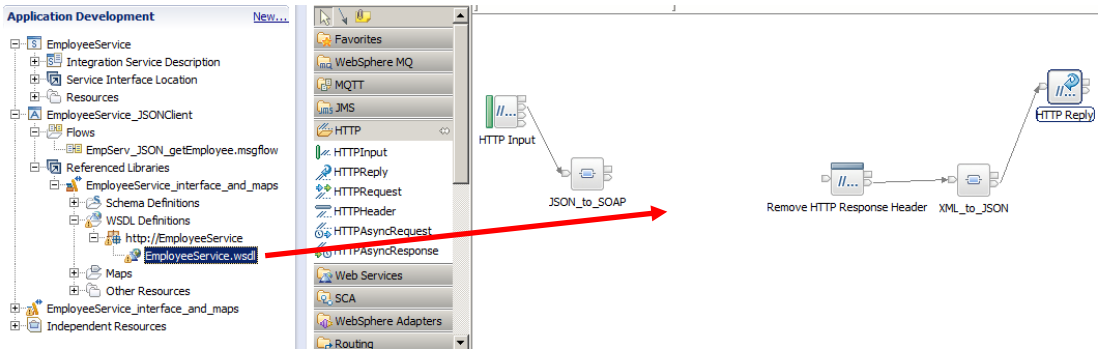
   For the HTTP Header node:

   - In the node properties, select the HTTPResponse tab
   - Select the radio button "Delete header" (this is required because if not deleted, it will override the subsequent setting of the ContentType, which is needed for data in the format "application/json").



6. In the EmployeeService_JSONClient application, expand the Referenced Libraries folder, then the WSDL Definitions folder. You will see that it contains the WSDL of the EmployeeService.

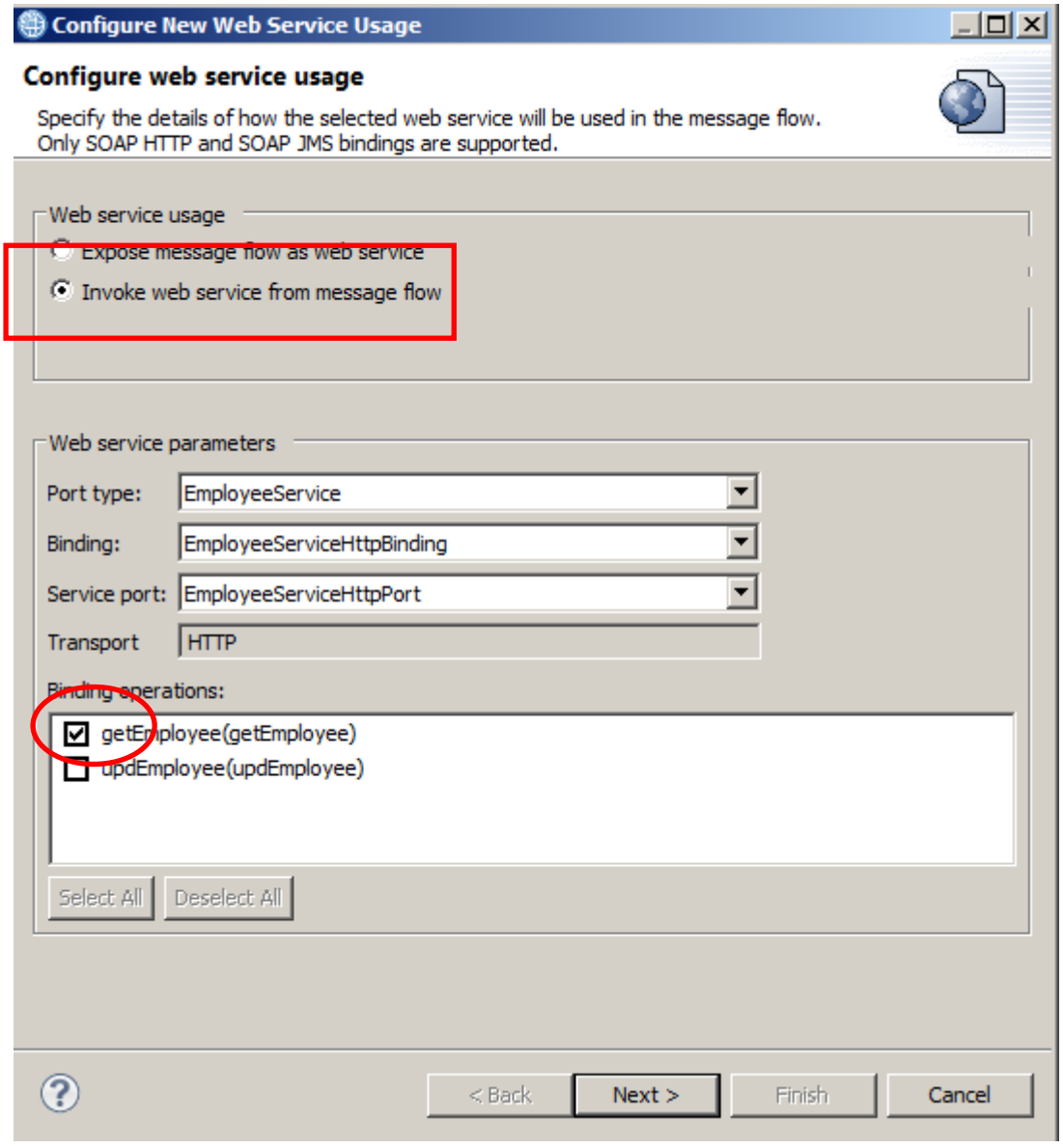
   Drag and drop this wsdl onto the flow editor, after the first mapping node.



7. When you drop the wsdl onto the flow editor, the Configure New Web Service window will open, as shown below.

   The "Expose message flow as web service" choice will be selected by default; this should be changed to "Invoke web service from message flow". No other changes are necessary, so click Finish.

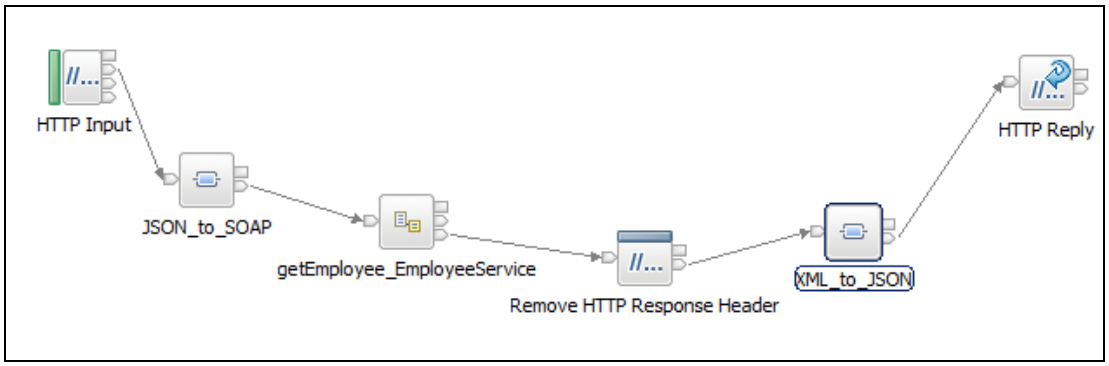
   (Note - if you are using the pre-built solution of EmployeeService, make sure you select the **getEmployee** operation).



8. Connect the flow nodes as shown. Take care to make sure the Output terminals are the ones to be connected (the out terminals will be green when you hover over them).

   Save the flow.

   Note that the Mapping nodes show crosses, indicating that further configuration is required. You will do that in the next section.



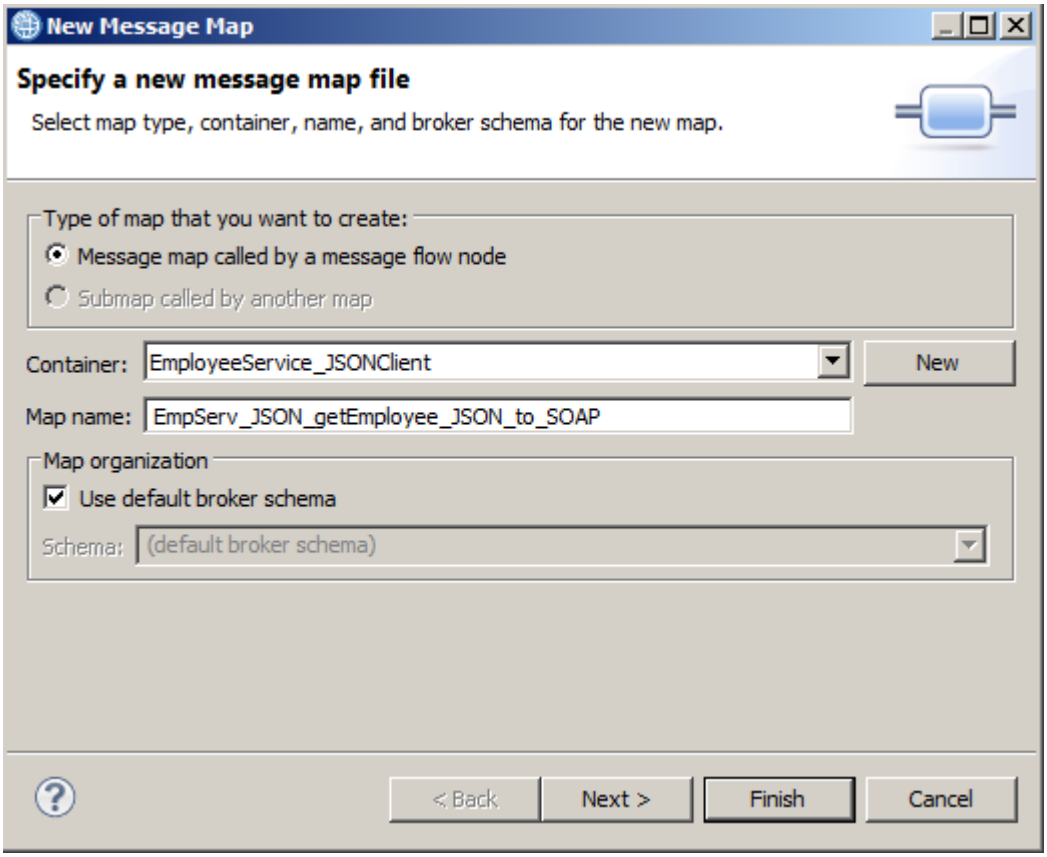
2.1 Implement the Mapping Nodes

This section is going to use the Mapping Nodes with the schemaless mapping support provided in IIB V10. The first mapping node will receive an input message in JSON format, and this will be converted to the SOAP message that is required by EmployeeService.

The second mapping node will perform the reverse operation, converting an XML message back to JSON format, which will be sent back to the original client using the HTTP Reply node. (Note the response message has automatically had the SOAP headers removed, because dropping the wsdl onto the flow editor generates a subflow with a SOAP Extract node).

2.1.1 Implement the JSON_to_SOAP map

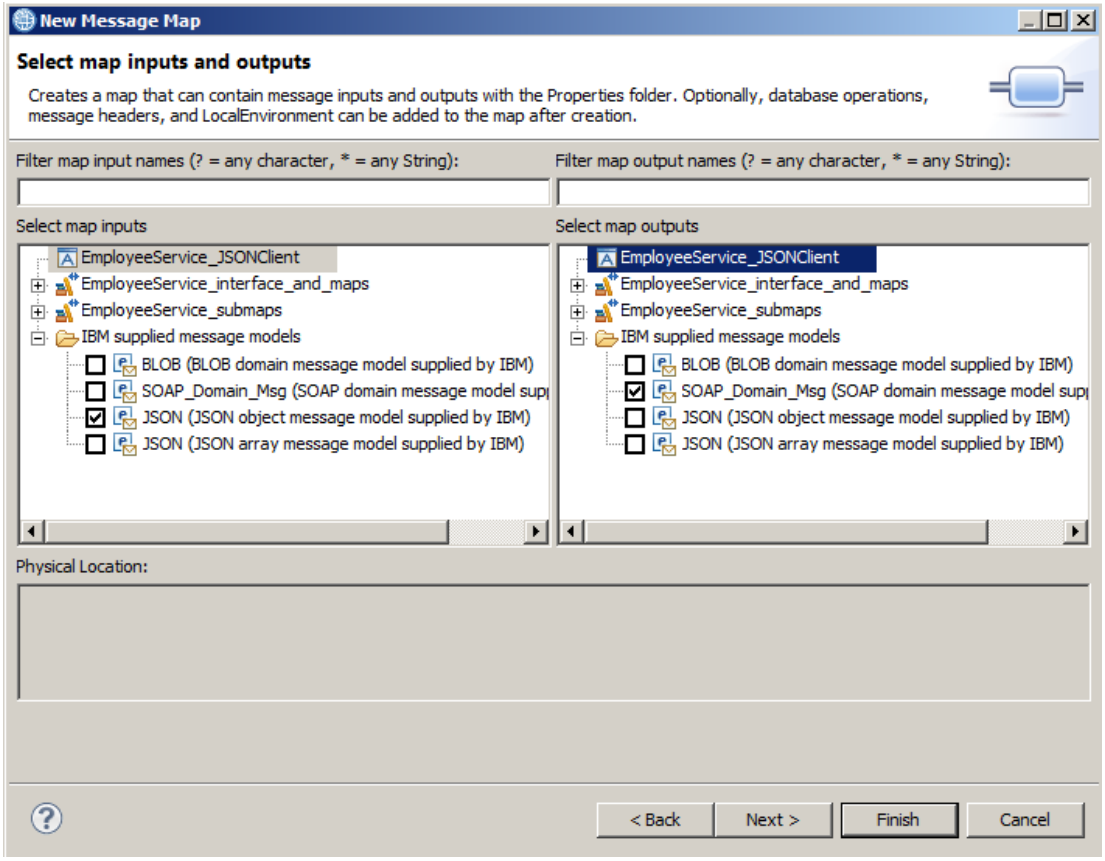
1. Double-click the first Mapping Node, JSON_to_SOAP. At the first pop-up, click **Next**.



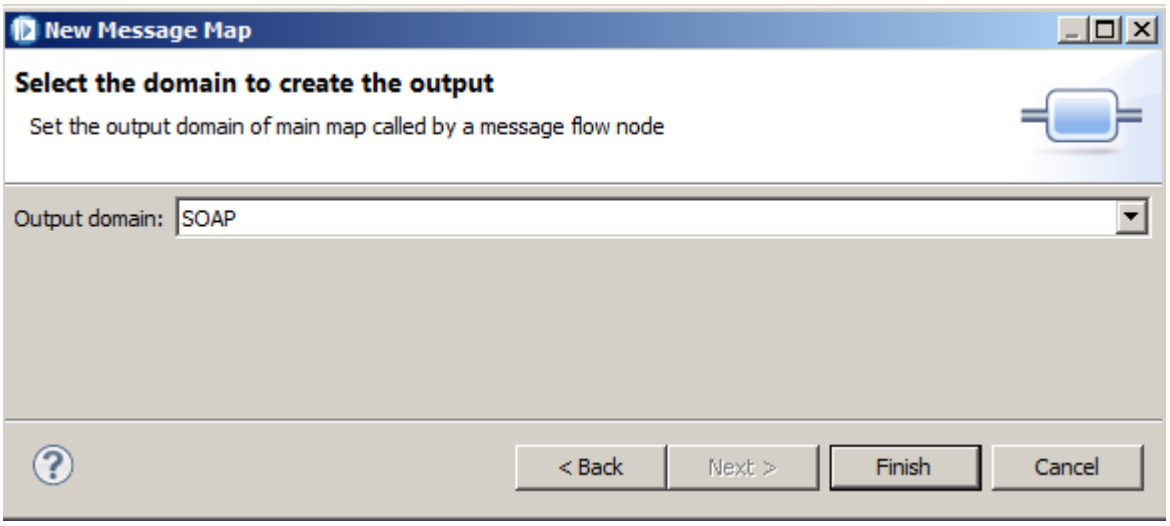
2. At the next window, select the map inputs and outputs.

   - For the input, expand the IBM supplied message models, and choose JSON (JSON object message model).
   - For the output, under the same folder, choose SOAP_Domain_Msg.

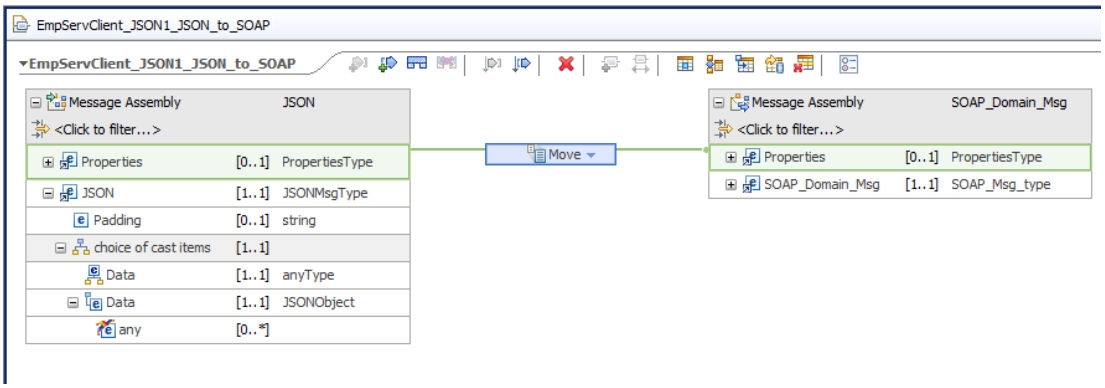
   Click **Next**.



3. Check that the Output domain is SOAP, and click Finish.



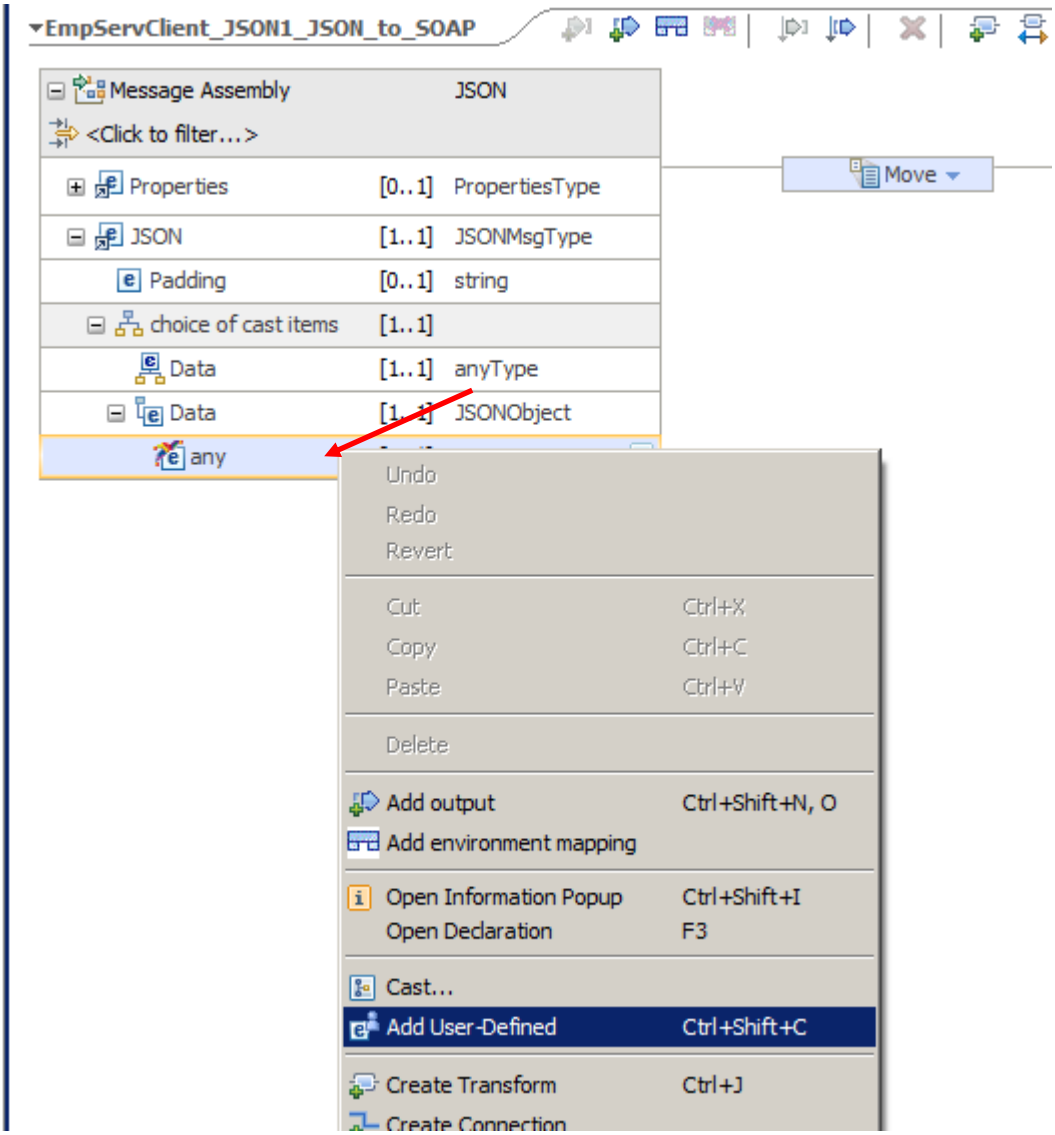
4. The mapping editor will load. Expand the Message Assembly for the input message.



5. You are going to add a user-defined element to the JSON part of the input message assembly.

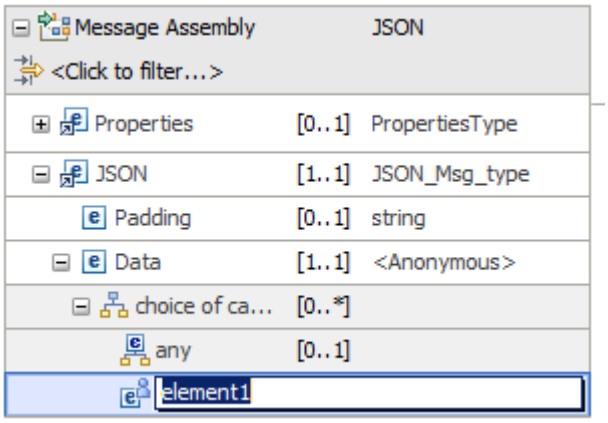
   You will see that the JSON folder has a Data element, and an "any" element.

   Right-click the "any" element, and select "Add User Defined" from the context menu.

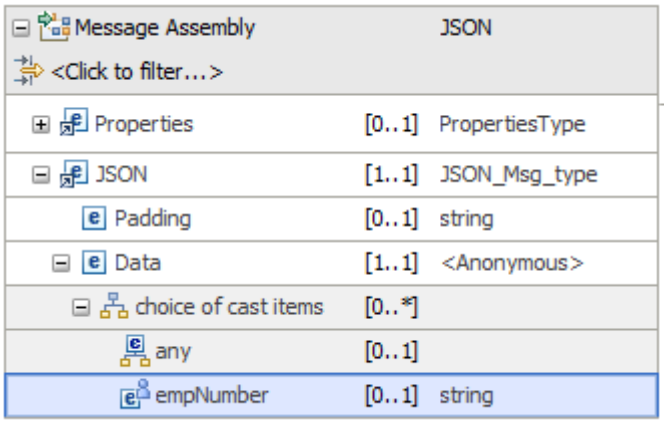


Provided by IBM BetaWorks

6. The word "element1" is added to the JSON message.



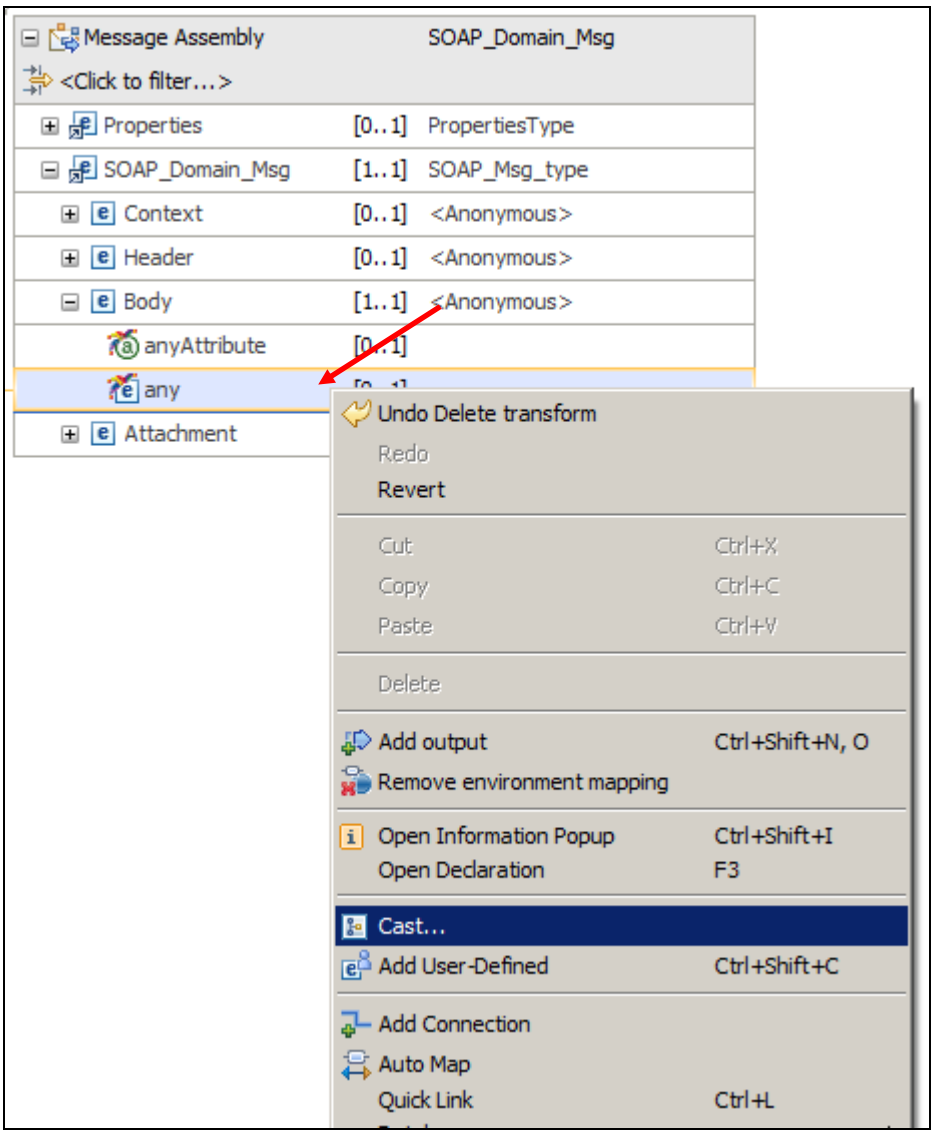
7. Change this to "empNumber" by overtyping "element1". Leave the element type as "string".



8. Now switch to the output message assembly on the right. This message is going to be a SOAP message, so it has to be Cast as a SOAP message, based on the schema that defines the output of the EmployeeService getEmployee operation.

   In the output message assembly, expand the SOAP_Domain_Msg, then Body.

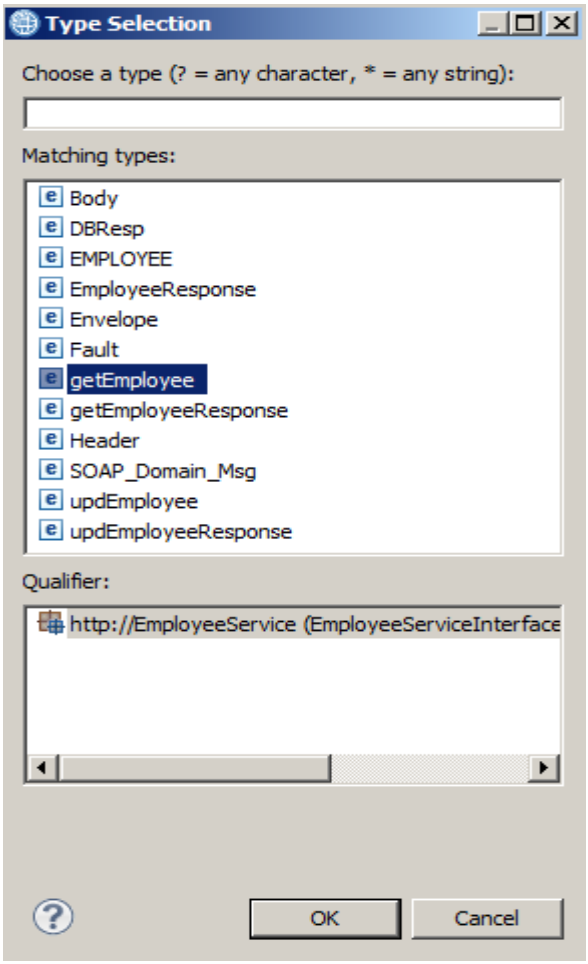
   Right-click the "any" element, and select "Cast" from the context menu.



9. From the Type Selection window, highlight **getEmployee**, and click OK.

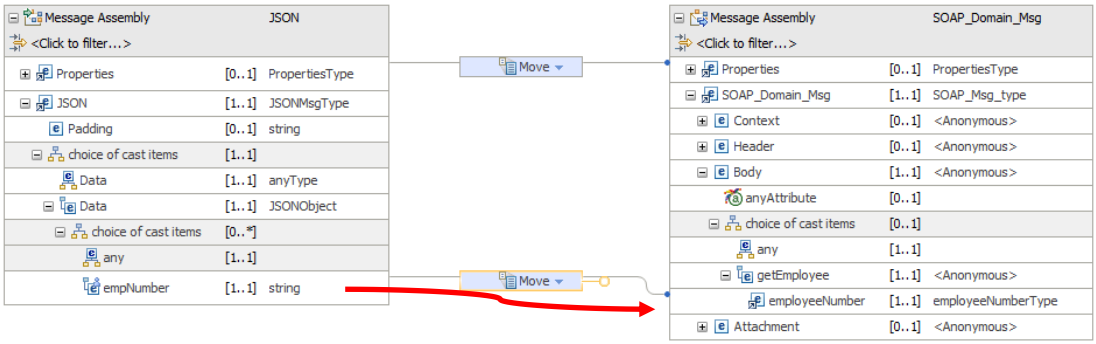
   (getEmployee is the name of the input message for EmployeeService getEmployee operation).

   Note - if you are using the pre-built version of EmployeeService, you will also see other types, as shown here.



10. Expand the getEmployee message, and drag and drop the JSON input **empNumber** to the output **employeeNumber**.

    This will create a Move transform.



Save and close this mapping node.

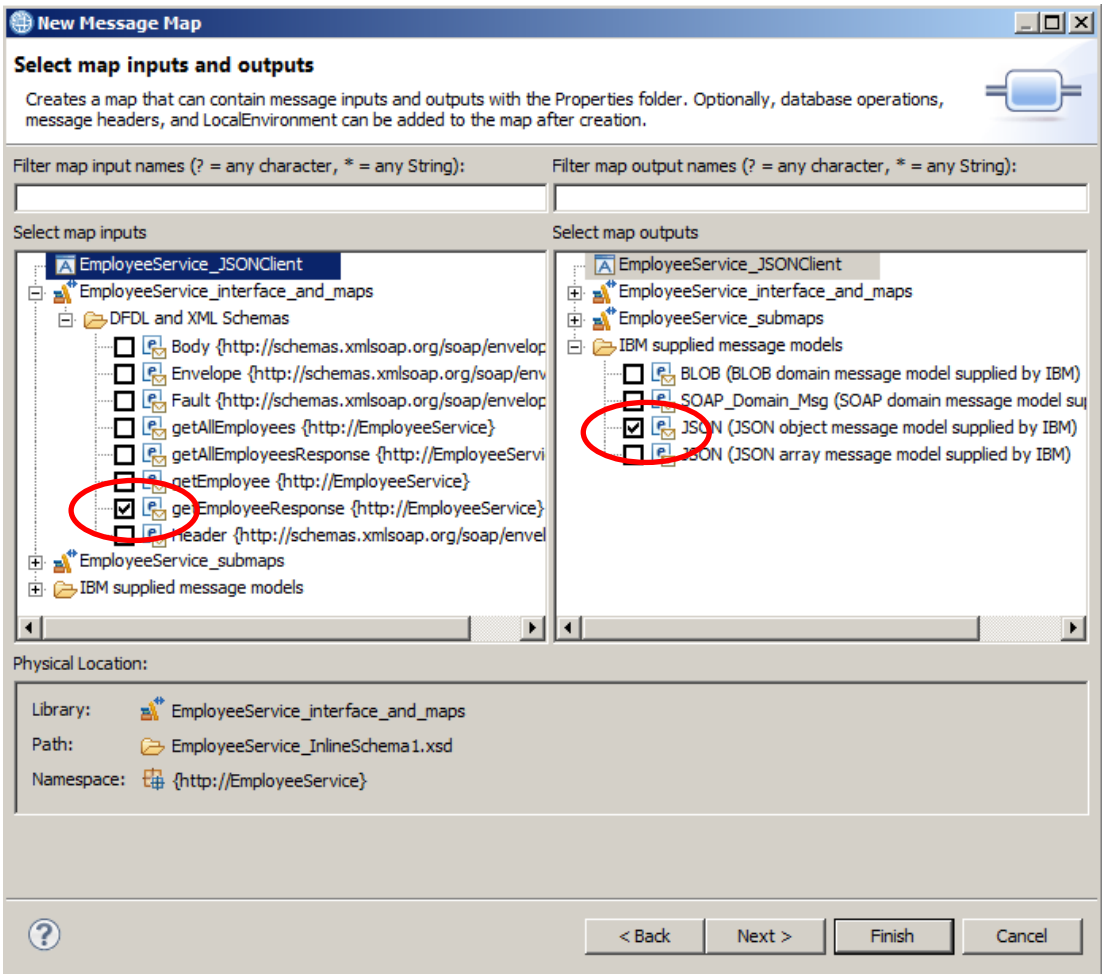
2.1.2 Implement the XML_to_JSON map

1. Double-click the XML_to_JSON map node, click Next, and at the input and output selection, make the following choices:

   - For the input message, because the generated subflow which invokes the web service has included a SOAP Extract node, you do not need to specify a SOAP_Domain message. Instead, expand EmployeeService_submaps/DFDL and XML Schemas, and select the **getEmployeeResponse** message.

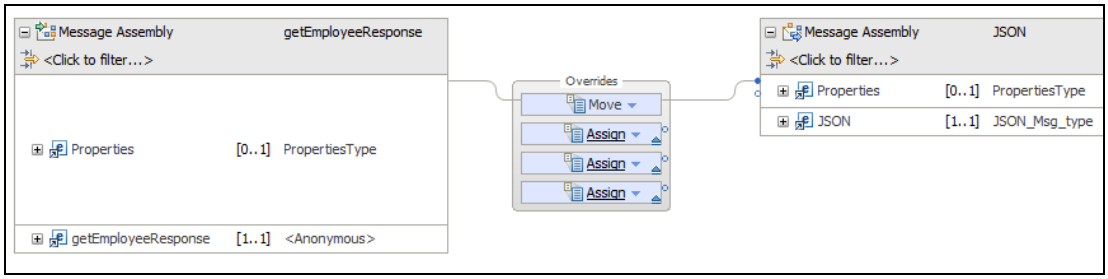
   - For the output message, expand IBM supplied message models, and select JSON (JSON object message).

   Click Finish (or Next to check that the output domain is JSON).



2. The default map will be shown.

   The JSON domain does not require the use of MsgSet, MsgFormat and MsgType. The mapping editor initialises those values for you, so do not change the provided Assign transforms.

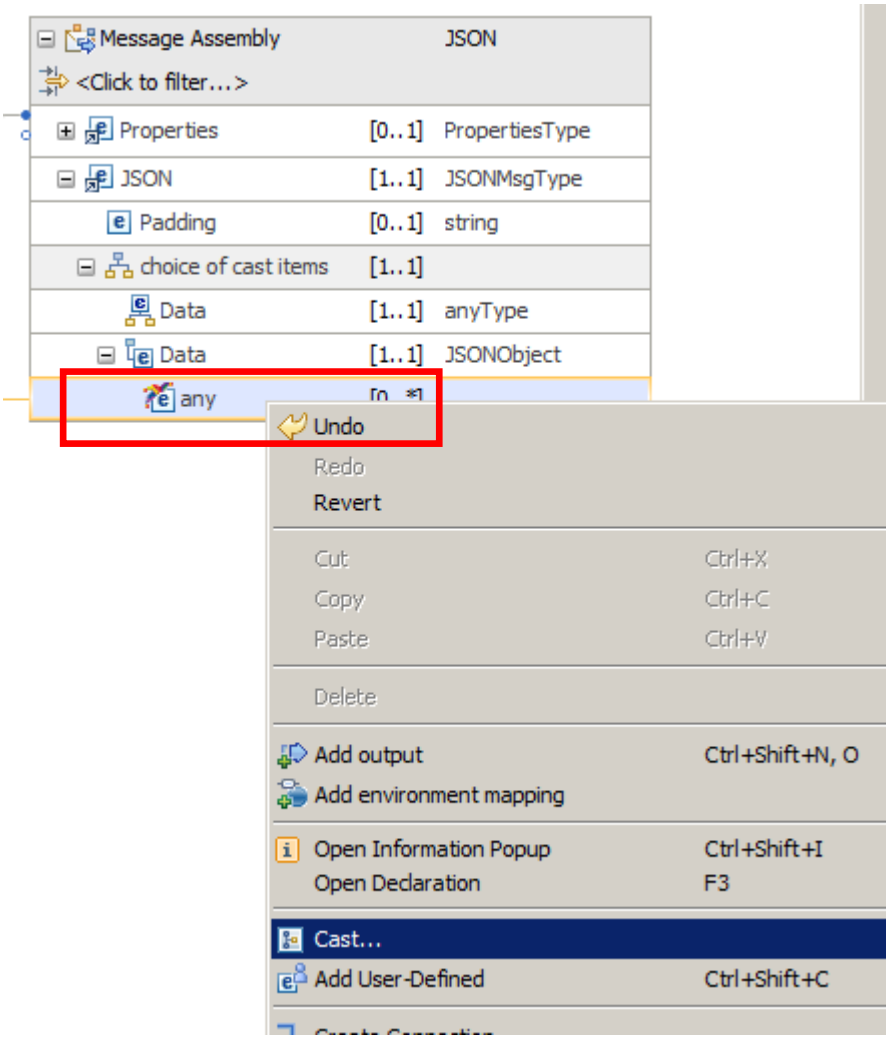


3. Before creating the map transformations, you are going to define the output message as a JSON object message. This will be CAST from the schema type EmployeeResponse message, so will contain the two schema types DBResp and EMPLOYEE.

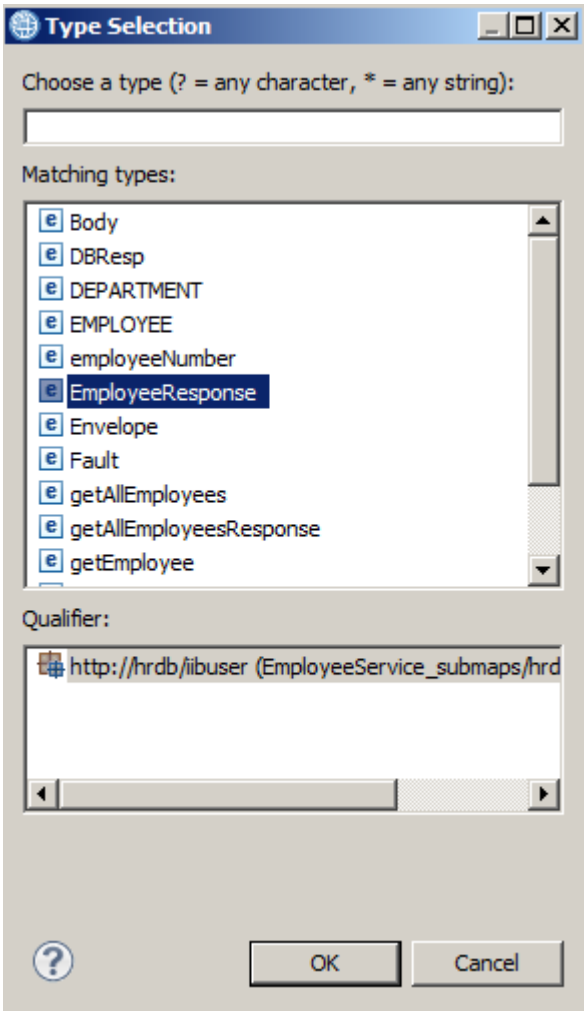
   Expand the output message assembly, JSON element, and "choice of cast items".

   Expand the last Data element to show the "any" element.

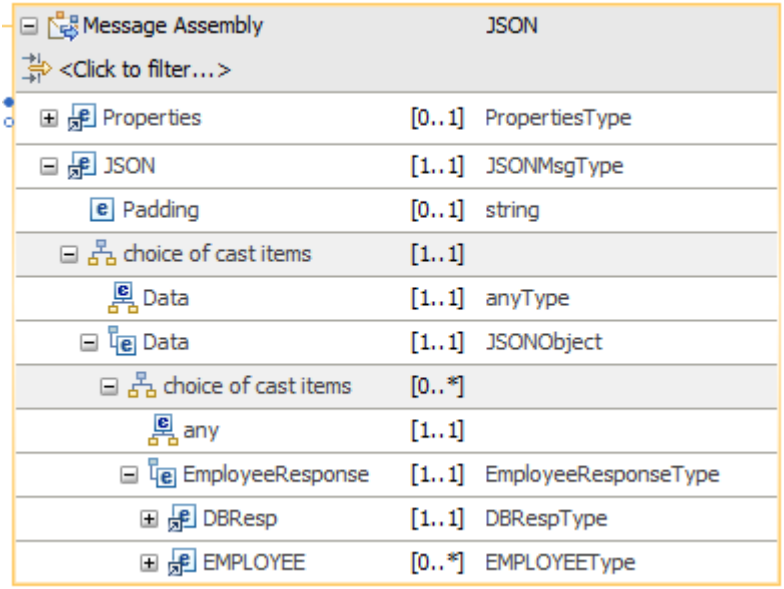
   Right-click the "any" element, and select Cast from the context menu.



4. From the Type Selection, choose EmployeeResponse, click OK.

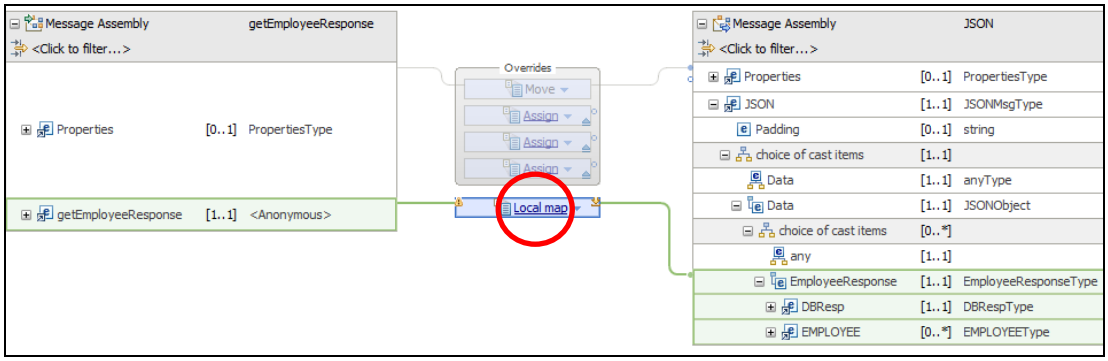


This will include this data type in the output message.



5. Since the transforms will operate on an output containing an array, the initial mapping must be performed at the highest level of the map assembly.

   Connect the input **getEmployeeResponse** to the output **EmployeeResponse**. This will create a Local Map.



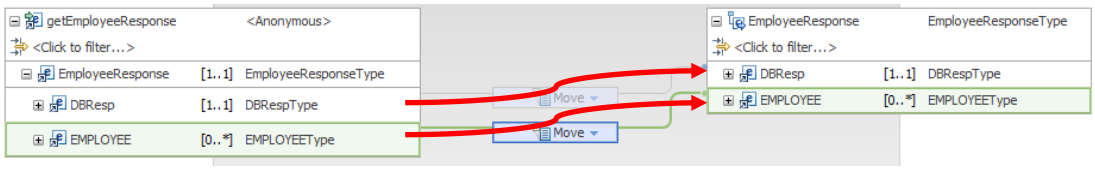
6. Click Local Map to provide the detail element mappings. Initially, no element mappings are done.

7. Connect the input DBResp to the output DBResp. This will create a Move transform.

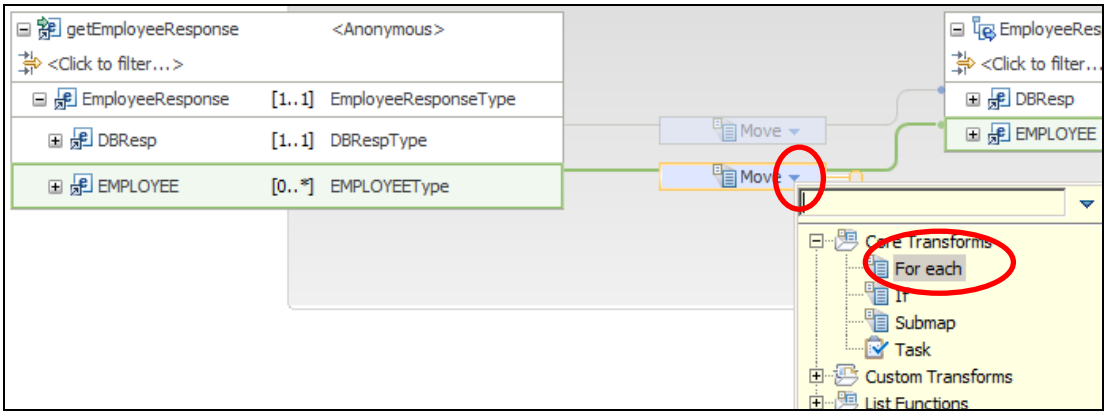
   (If a Local Map transform is created, accept the quickfix (light-bulb) to create the individual mappings.)

   Connect EMPLOYEE to EMPLOYEE. This will also create a Move transform.

   (If a For Each transform is added automatically, bypass the next step).

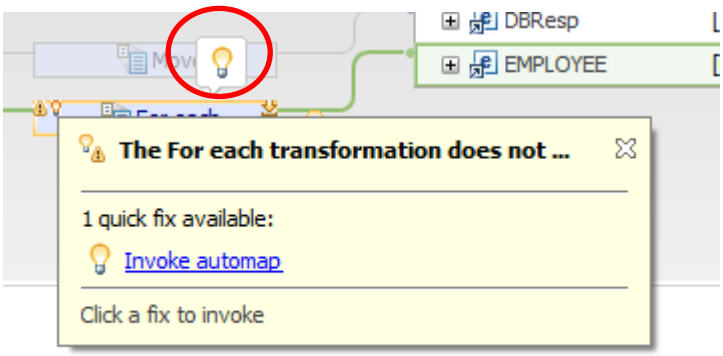


8. For the EMPLOYEE to EMPLOYEE transform, change the Move transform to a "For each".

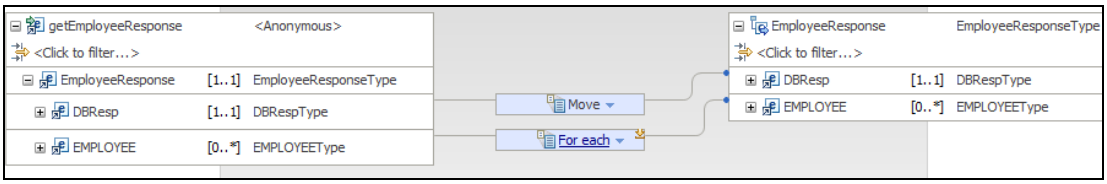


9. The transform will show a warning, indicating no mappings have been done. Click the light-bulb to invoke the quick fix, and invoke the Automap suggestion.

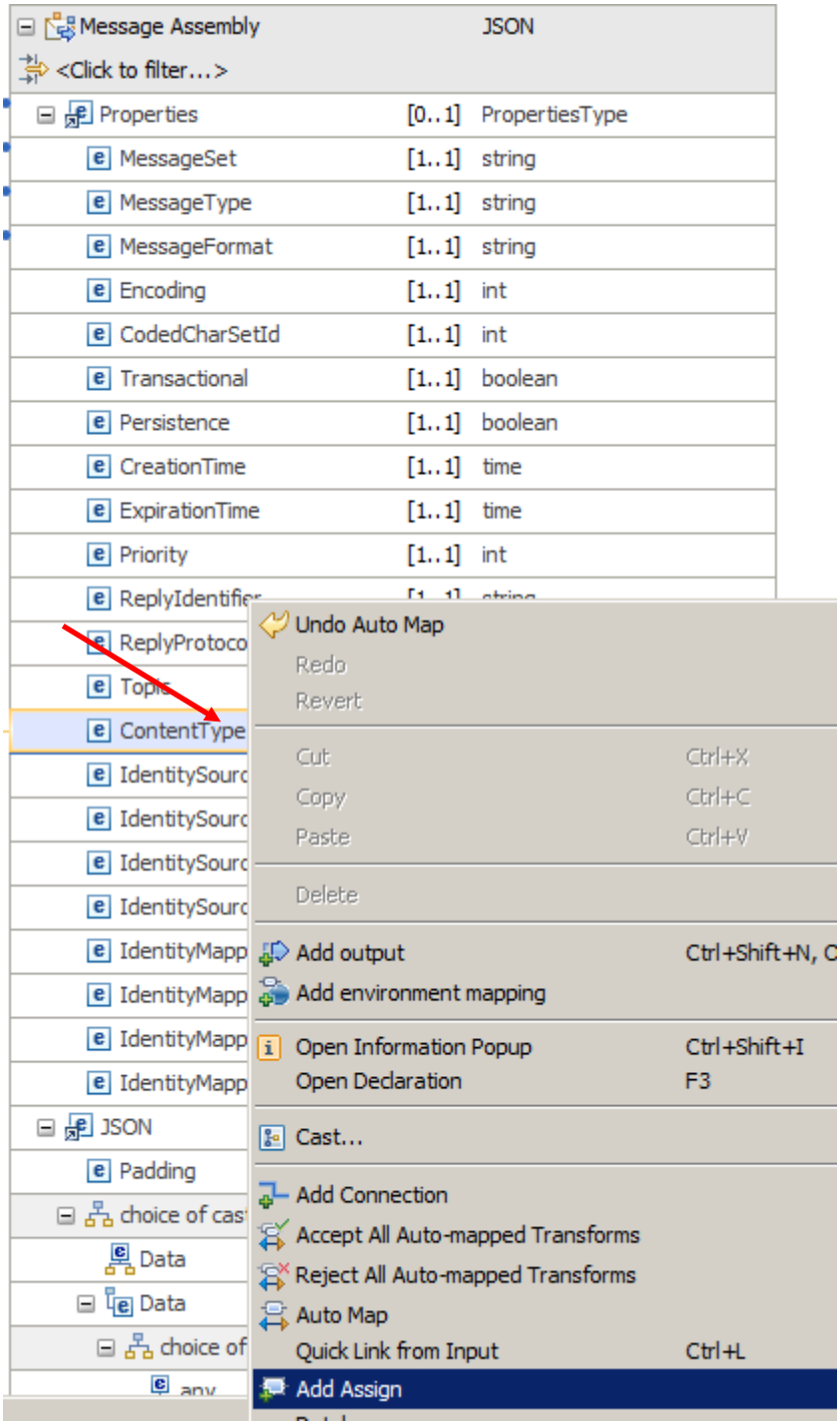
   This will create all the required mappings inside the "For each".



10. This part of the map will now look like this:

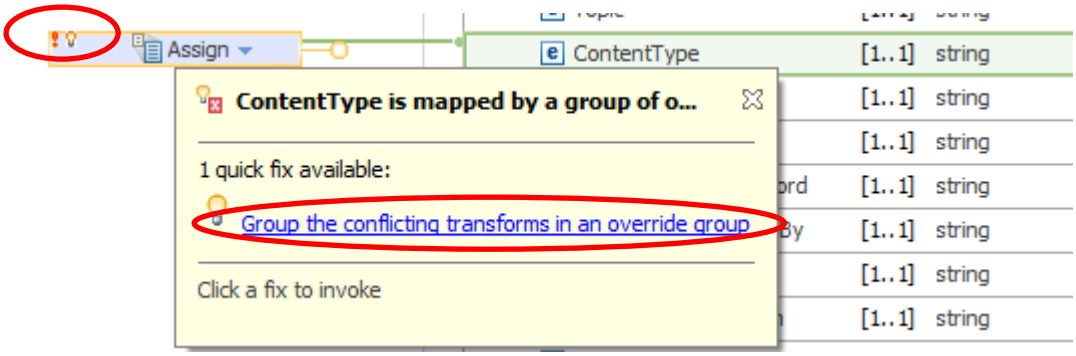


11. Finally, set the Content Type of the output message properties to "application/json". To do this, go to the top level of the map, expand the output Message Assembly Properties, right-click ContentType, and select Add Assign.



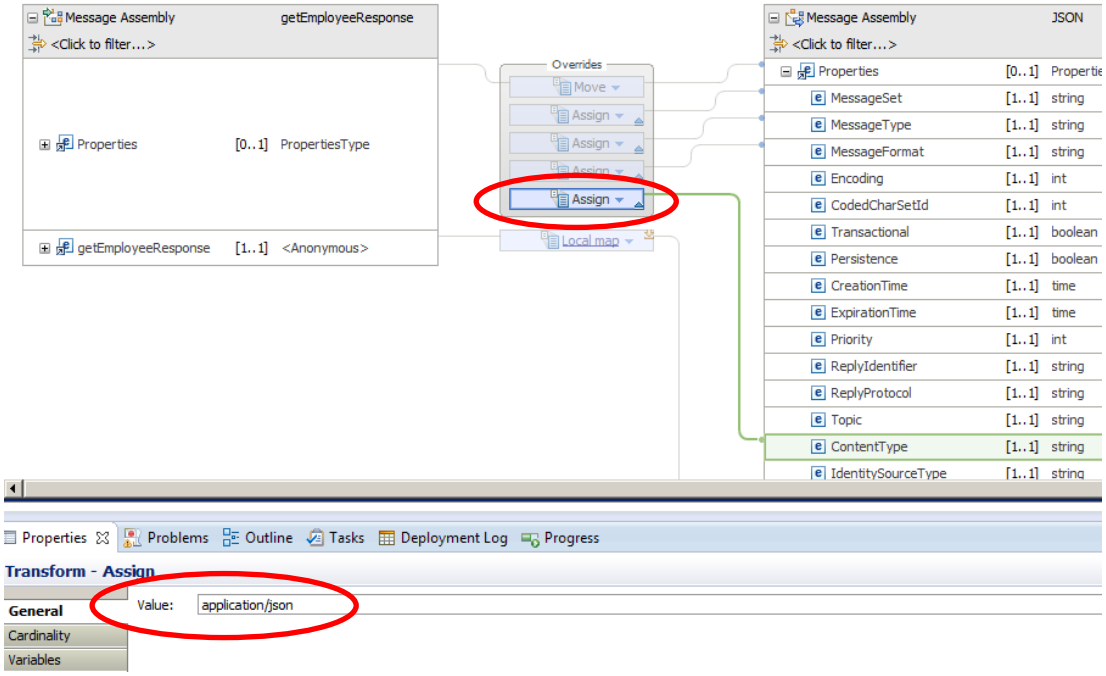
12. This will add the Assign transform, but initially this will show an error. This is because some of the output properties values are already set within an Override Group.

    On the new Assign, click the light-bulb to invoke the quick fix. This will move the Assign to the Override Group.



13. Now highlight the new Assign, which is the last transform within the Override Group.

    In the properties of the transform, in the General tab, set the value to **application/json.**



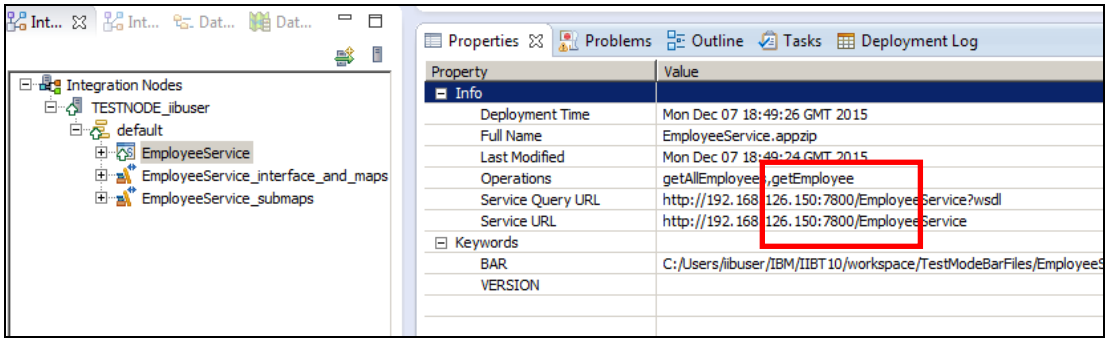
    Save and Close the map.

2.2 Complete the Message Flow

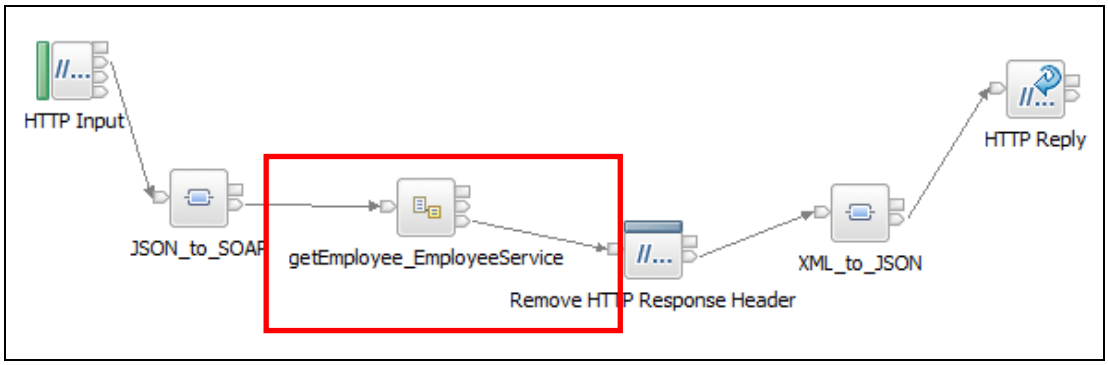
1. One final check is necessary. You should make sure that the new application is configured to use the correct port number of the EmployeeService.

   In the Integration Nodes pane, expand TESTNODE_iibuser and the default server.

   Highlight EmployeeService. In the properties of the deployed service, you will see the Service URL that is active for this service. Note and record the port number that is in use. In the example below, this is 7800, but may be different on your system.



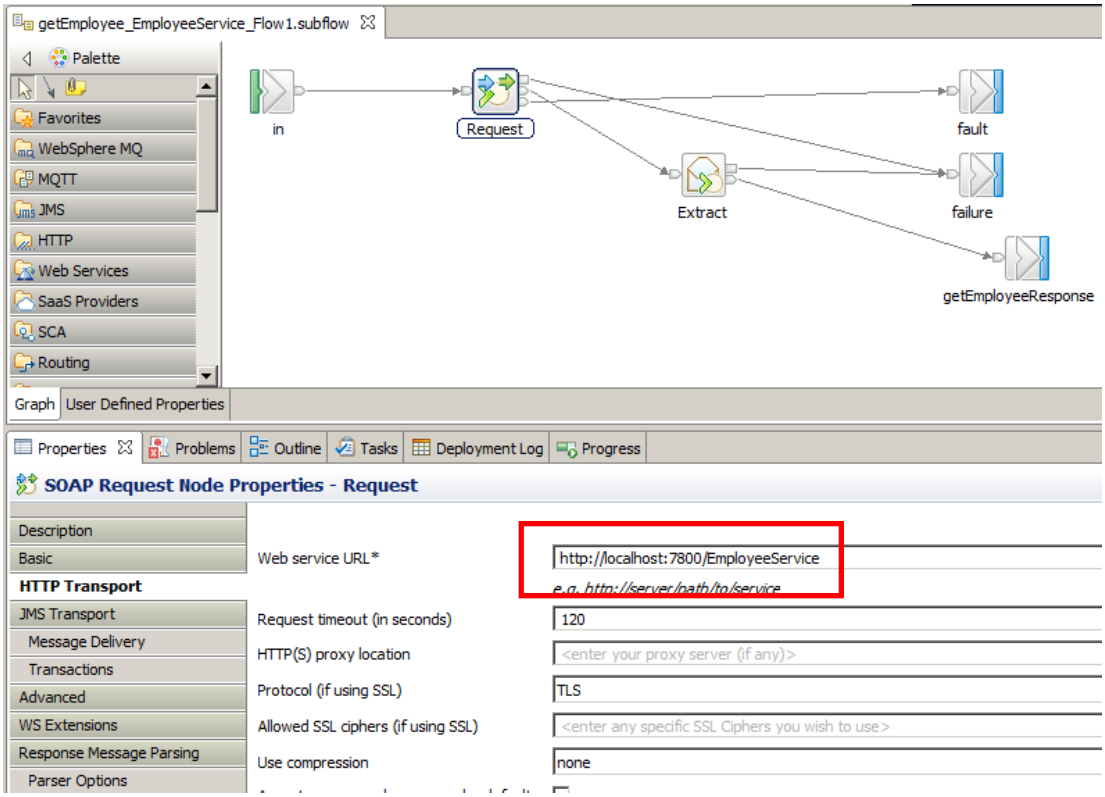
2. In the primary message flow, double-click the generated subflow getEmployee_EmployeeService.



3. In the node properties of the SOAP Request node, select the HTTP Transport tab. Ensure the Web Service URL uses the correct port number, as you have obtained in step 1 above (7800 in this example).

   (Note - a production environment would not normally hard-code a port number in the application, but this is done for ease of development in this lab).

   Save and close the subflow.



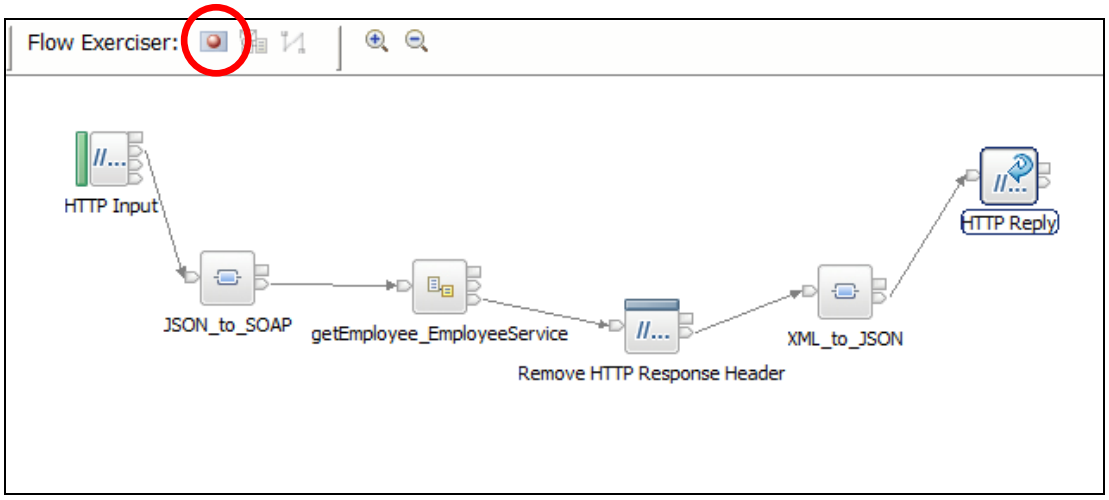
3. Test the EmployeeService_JSONClient Application

This chapter will present two ways of testing this application. The first approach is to use the Integration Bus Flow Exerciser. This will enable you to provide an input message to execute the flow, and to see the executed path of the message flow. It will also allow you to examine the message tree as it was at various points of the executed path.

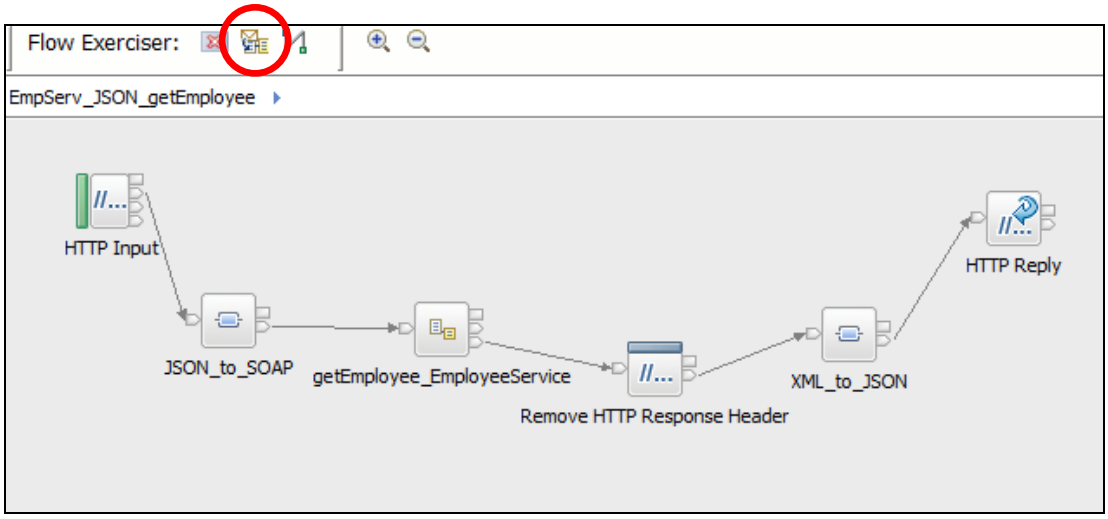
The second approach is to use an external tool to initiate the test. SOAPUI is provided for this purpose. This external tool can then be used in conjunction with the Flow Exerciser to view the flow execution path.

3.1 Test using the Flow Exerciser

1. Make sure the new message flow is open in the flow editor. Click the red record button.



2. This deploys the application, and activates the Flow Exerciser.

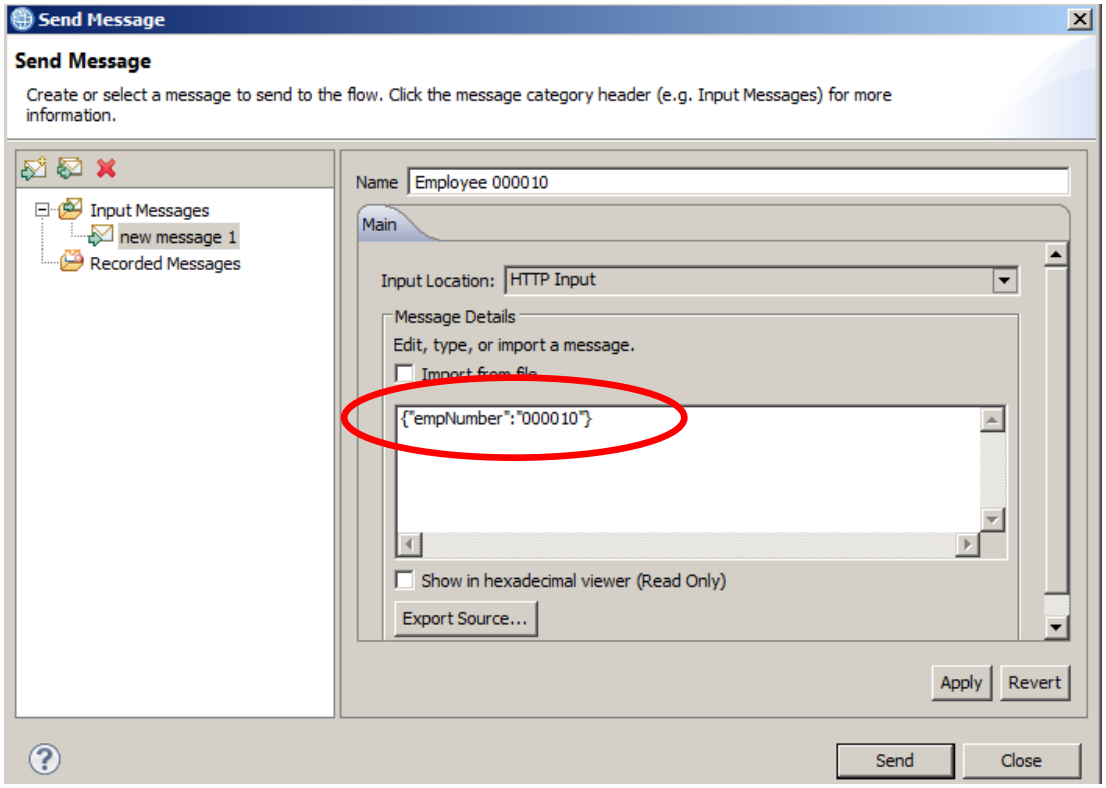


Using the middle icon (indicated above), open the Send Message dialogue.

Create a new message, name it "Employee 000010", and type the following data into the message area:

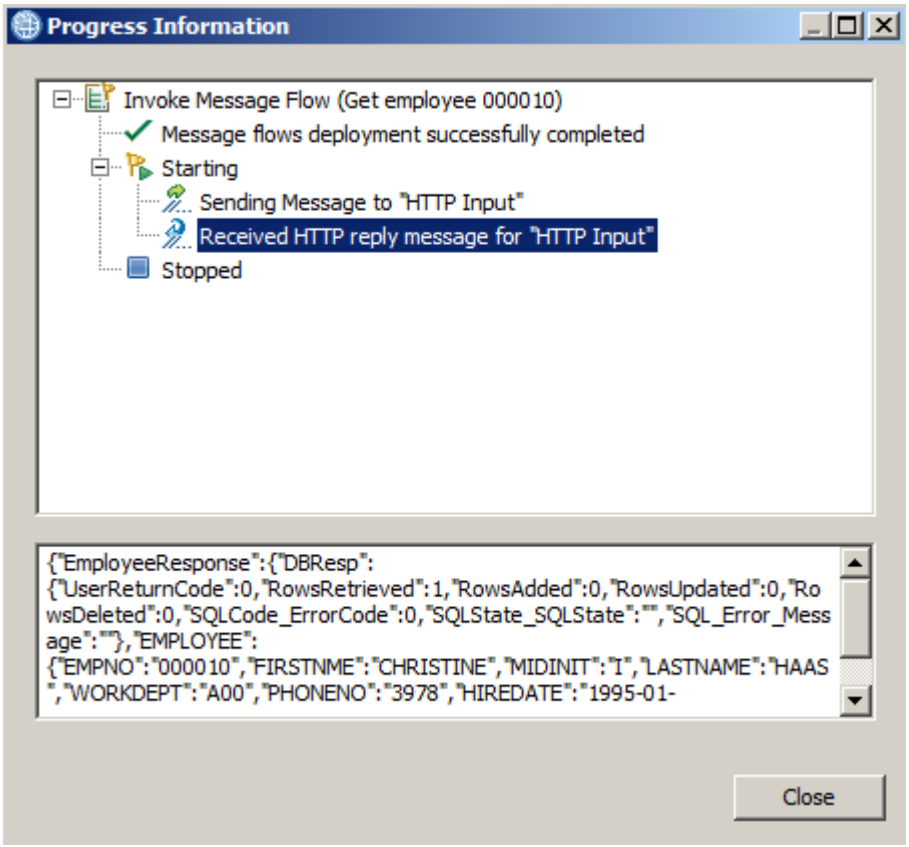
```
{"empNumber":"000010"}
```

Click Send.

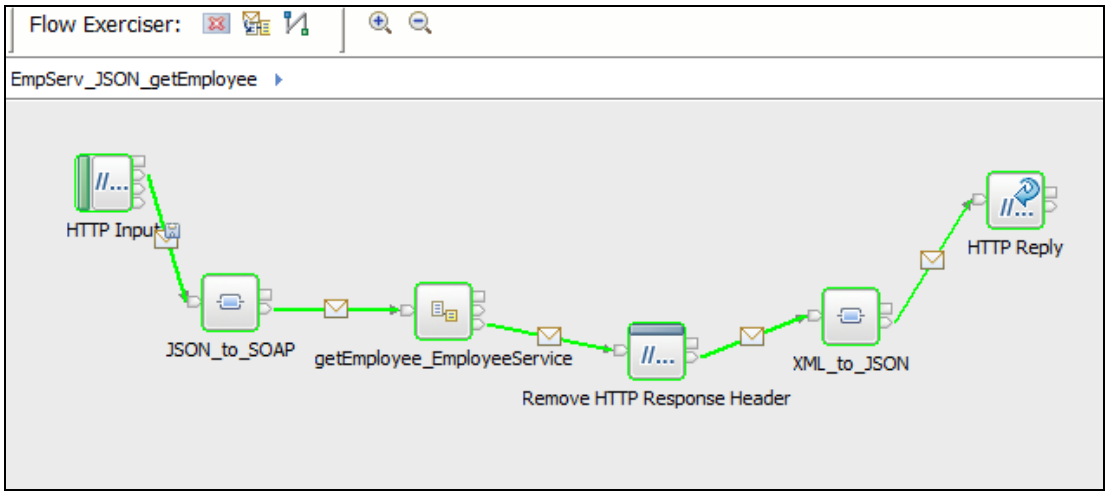


3. When the flow has executed, highlight the "Received HTTP reply....." You will see the output message in the display pane, containing the data retrieved from the EMPLOYEE table.

   Close this window.



4. You will see that the executed path is shown in green. Each connector shows an icon where the recorded message tree can be viewed.

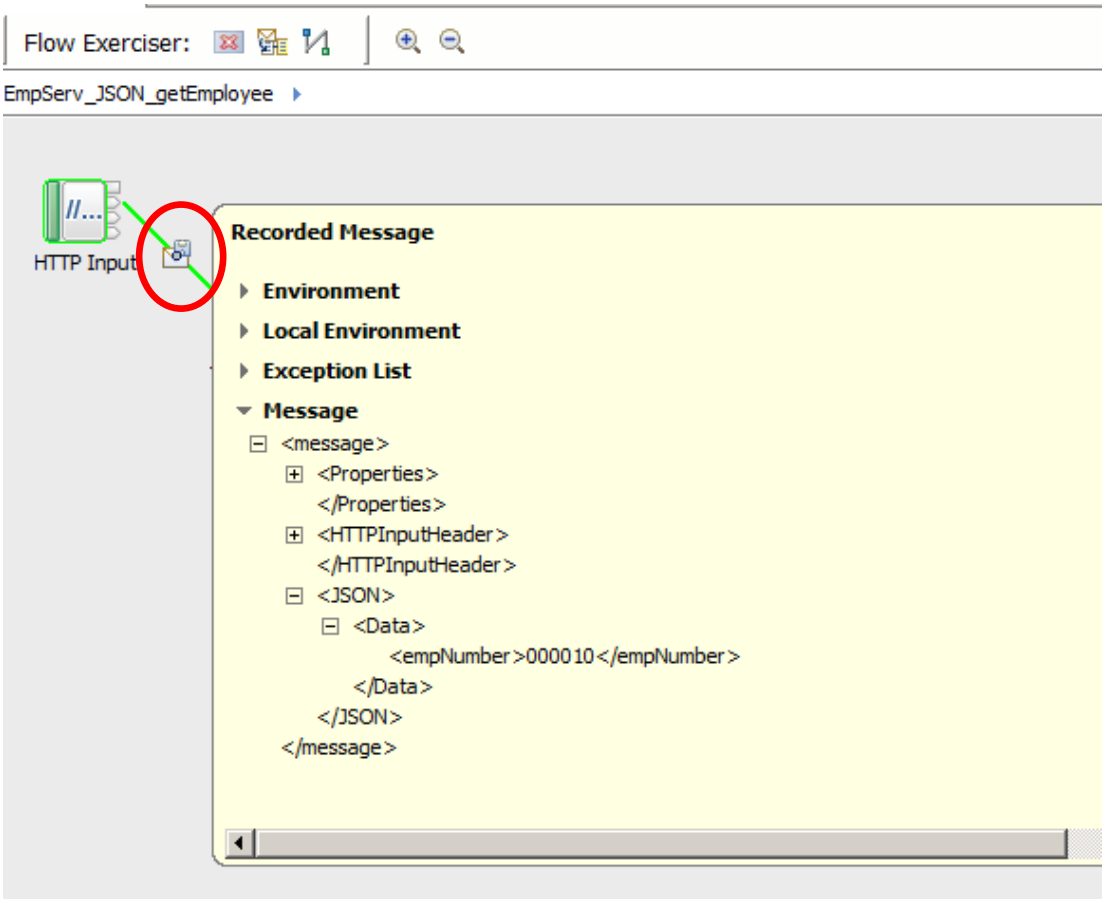


5. Click the icon on the first connector.

   You will see the input message (empNumber = 000010), parsed in the JSON domain. You can also see all the other components of the message tree, such as the Environment, Local Environment, etc.

   Note that the envelope on the related connector shows a small circle, indicating the connector that is currently being displayed.

   Click outside the yellow box to close it.

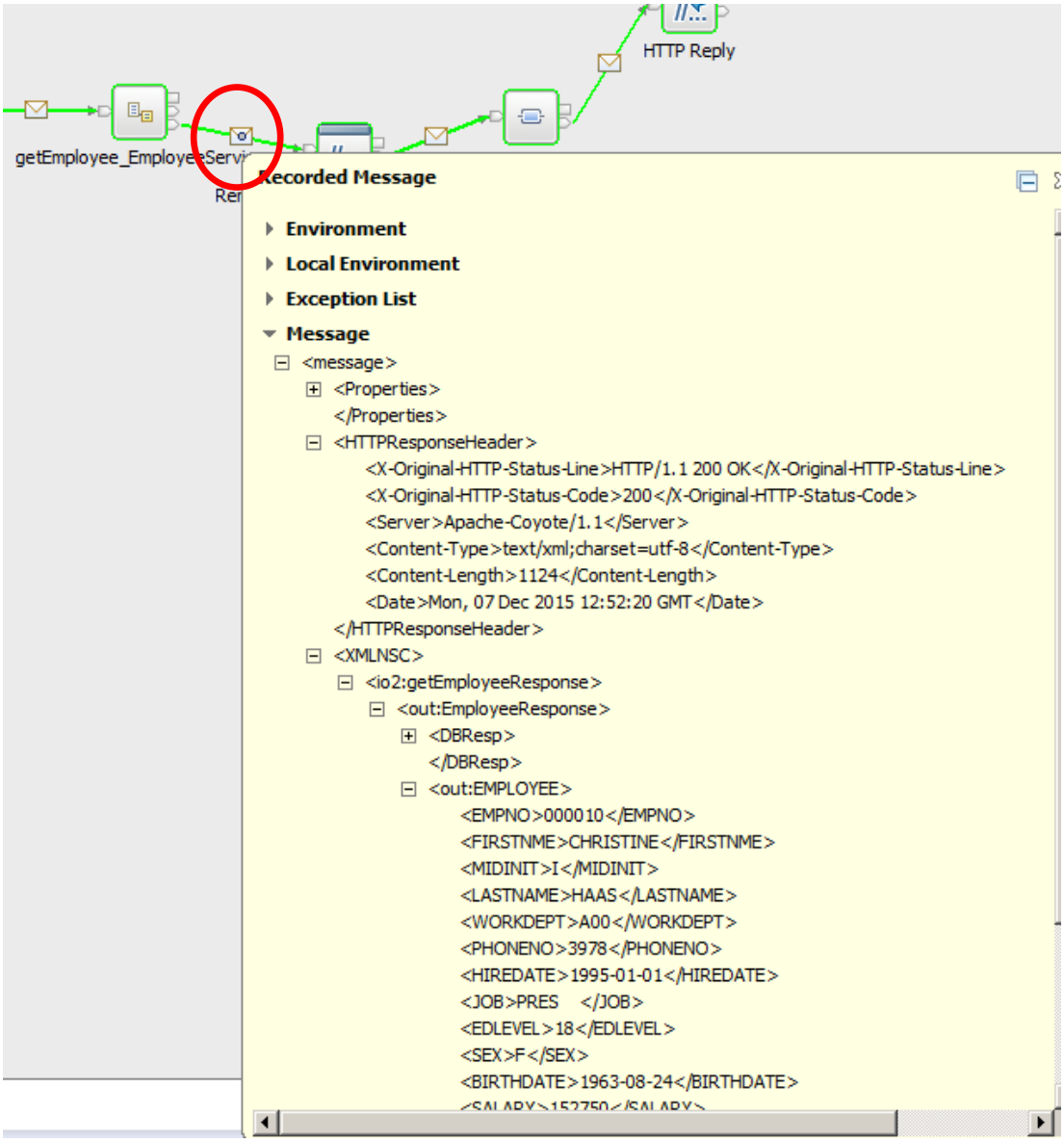


6. Click the icon on the connector after the getEmployee web service subflow.

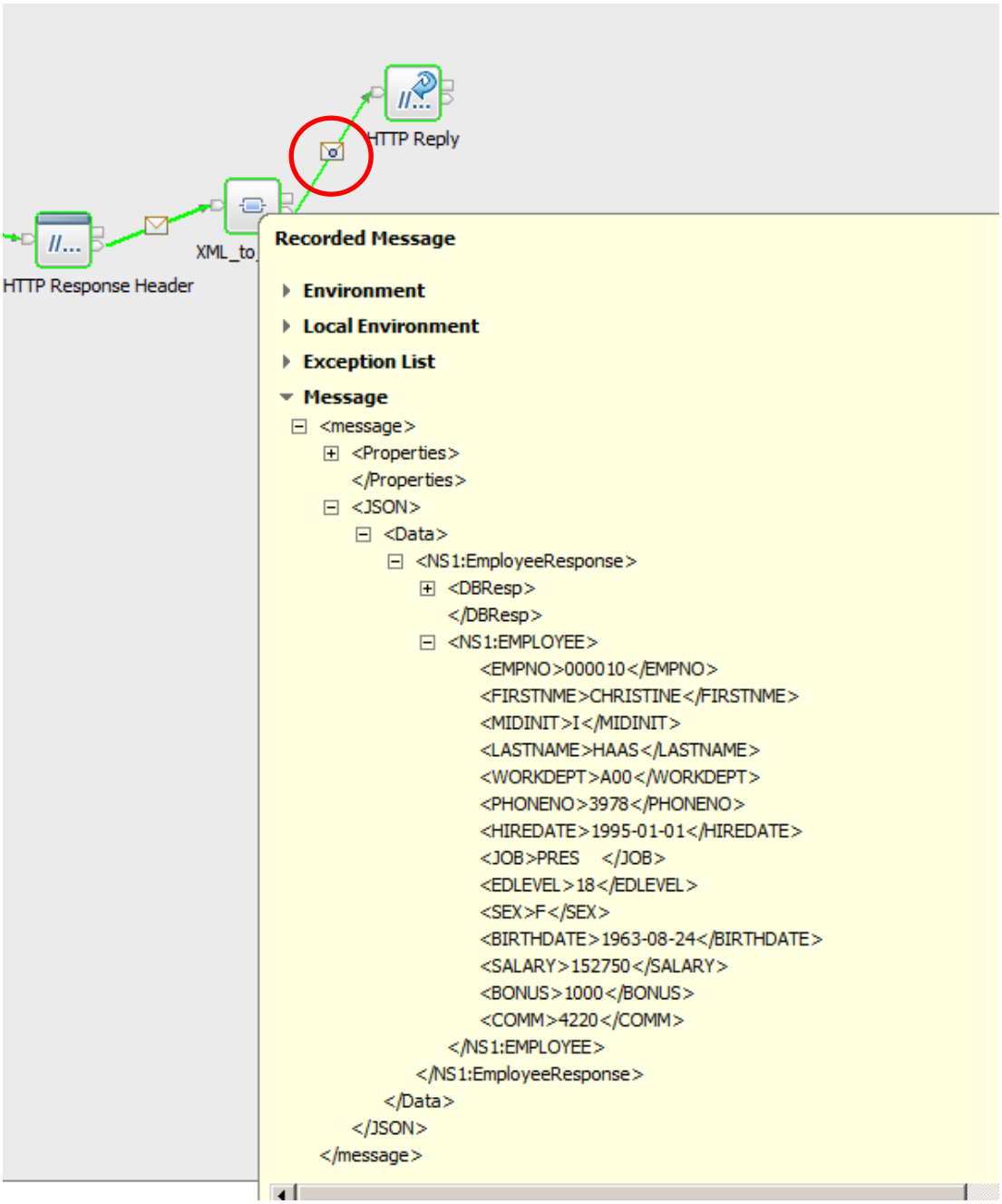
   Expand the Message.

   You will see that the message contains an HTTPResponseHeader at this point in the flow. This is because the flow has invoked a SOAP service, and IIB has added the response header. Note that the Content-Type is text/xml. Because the final response should be "application/json", this header must be removed, and the subsequent map will change the content type.

   You will also see the DBResp elements (eg. number of rows retrieved), and the data that was returned from the EMPLOYEE database. At this point in the flow, the data was in XML format, so the domain is XMLNSC.

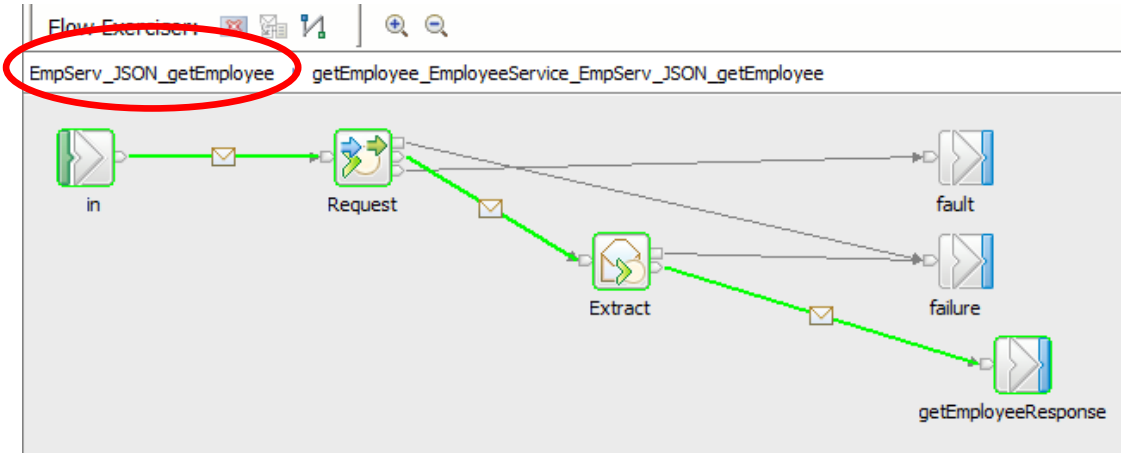


7. Click the icon after the XML_to_JSON map. Looking at the same part of the message tree, you will see that the data is now in JSON format, because it is shown under the JSON domain.



8. Open (double-click) the getEmployee_EmployeeService subflow icon. This will open the subflow, and will also show the green lines of flow execution. You can use the same tools to examine the message tree content at any point.

   When you have finished in the subflow, return to the main flow by clicking EmpServ_JSON_getEmployee.



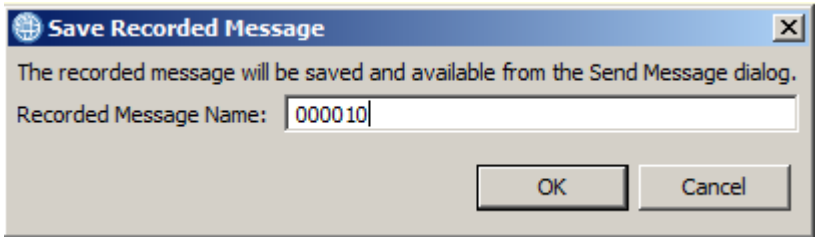
9. Back in the main flow, return to the icon on the first connector.

   On this message display, note that you can save the content of this message for future use.

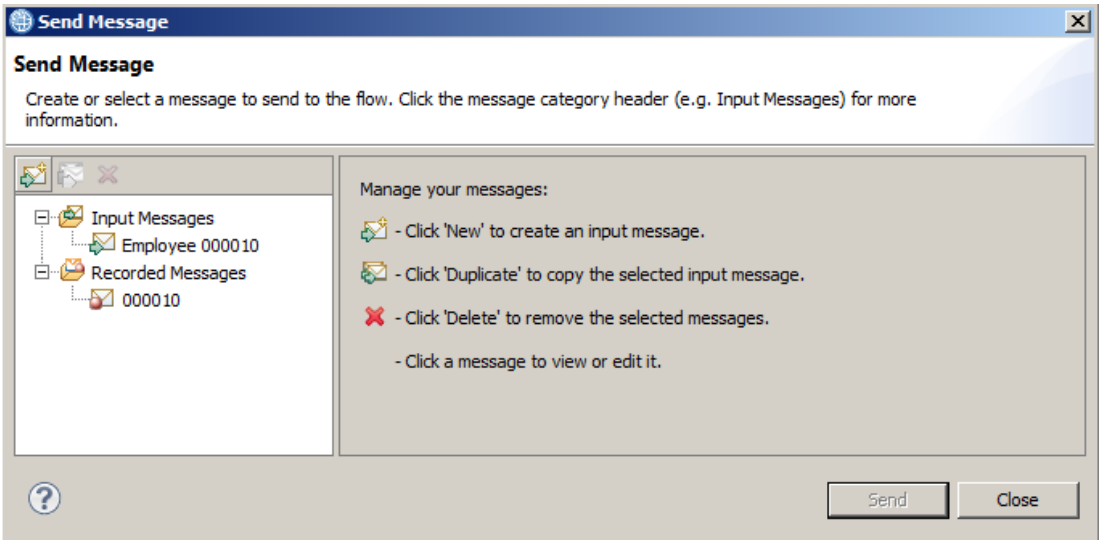
   Click the Save icon to save the message.

   Note that this facility is only available on the connector after the first node in a message flow.

10. Provide a suitable name for the test, for example "000010", and click OK.

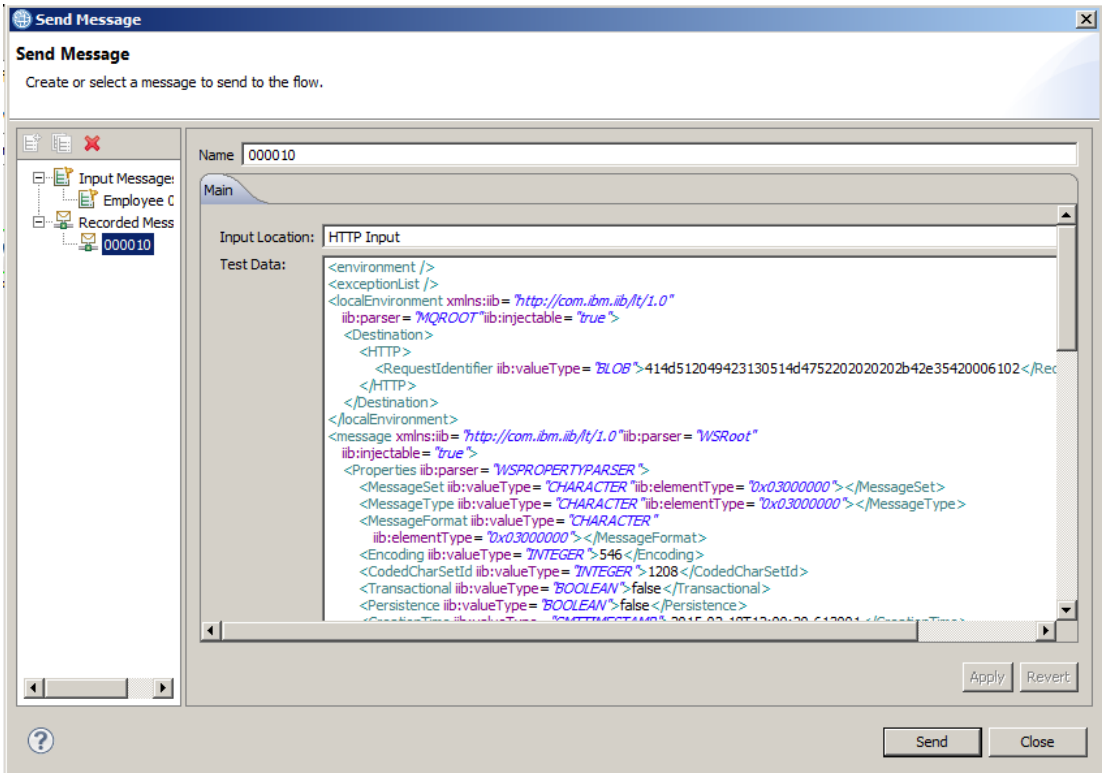


11. Click the Send Message icon again, to return to the Send Message window. You will see that the message has been saved in the Recorded Messages section.



12. Highlight the Recorded Message 000010. The recorded content of the message will be displayed.

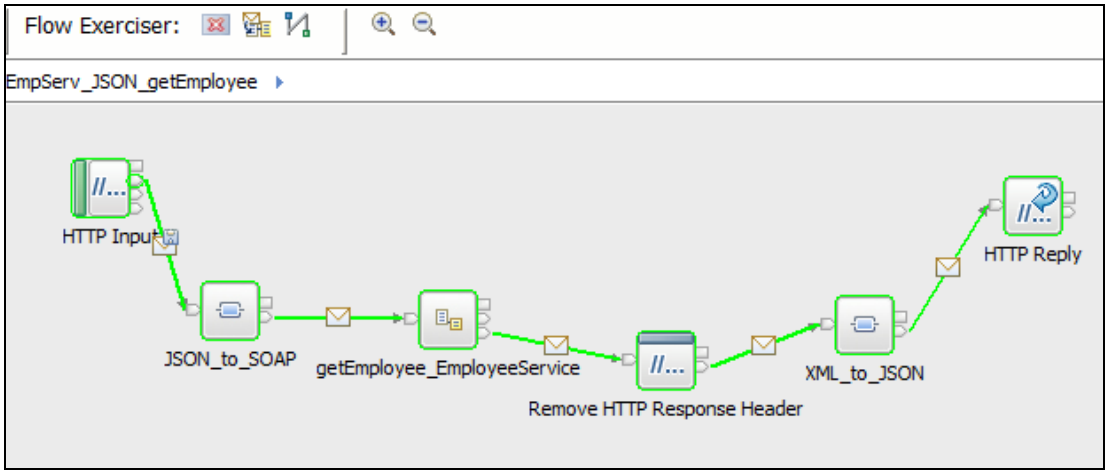
    This can now be resent to the message flow, so click Send.



Provided by IBM BetaWorks

13. The flow will be rerun with the recorded data. When the flow has executed, you will again see the green line highlighting the execution path.

    You can now view the message tree contents as before.



3.2 Test using SOAPUI and the Flow Exerciser

In some testing scenarios, you may need to view multiple flow invocations at the same time, to be able to compare flow executions. The Flow Exerciser will allow this, but you must invoke the flow using an external tool (such as SOAPUI when using http flows). This is described here. We assume that the Flow Exerciser is still running from the previous section.

1. Open SOAPUI from the Start menu (you may already have this open from a previous lab).

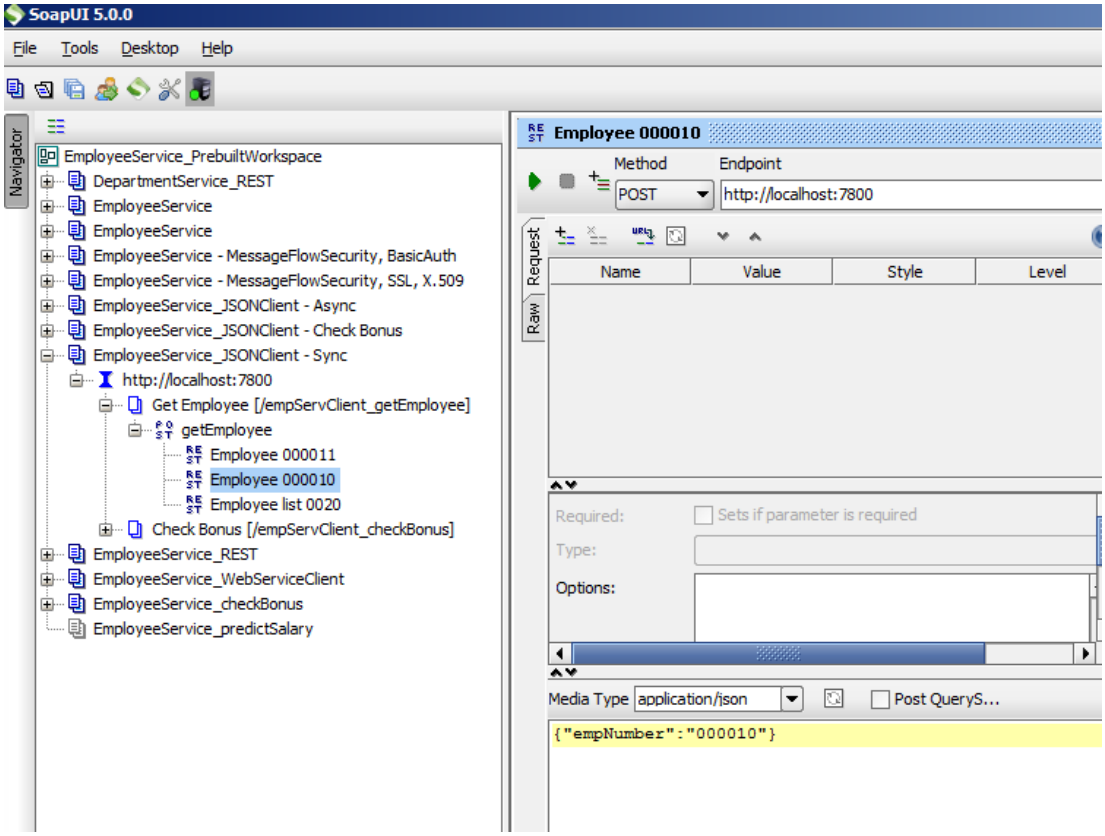
   Open the EmployeeService_PrebuiltWorkspace workspace.

   Expand the project "**EmployeeService_JSONClient - Sync**", and open the request "Employee 000010" that is under getEmployee.

   Note that the message payload that will be sent for this request is a JSON message:

   {"empNumber":"000010"}

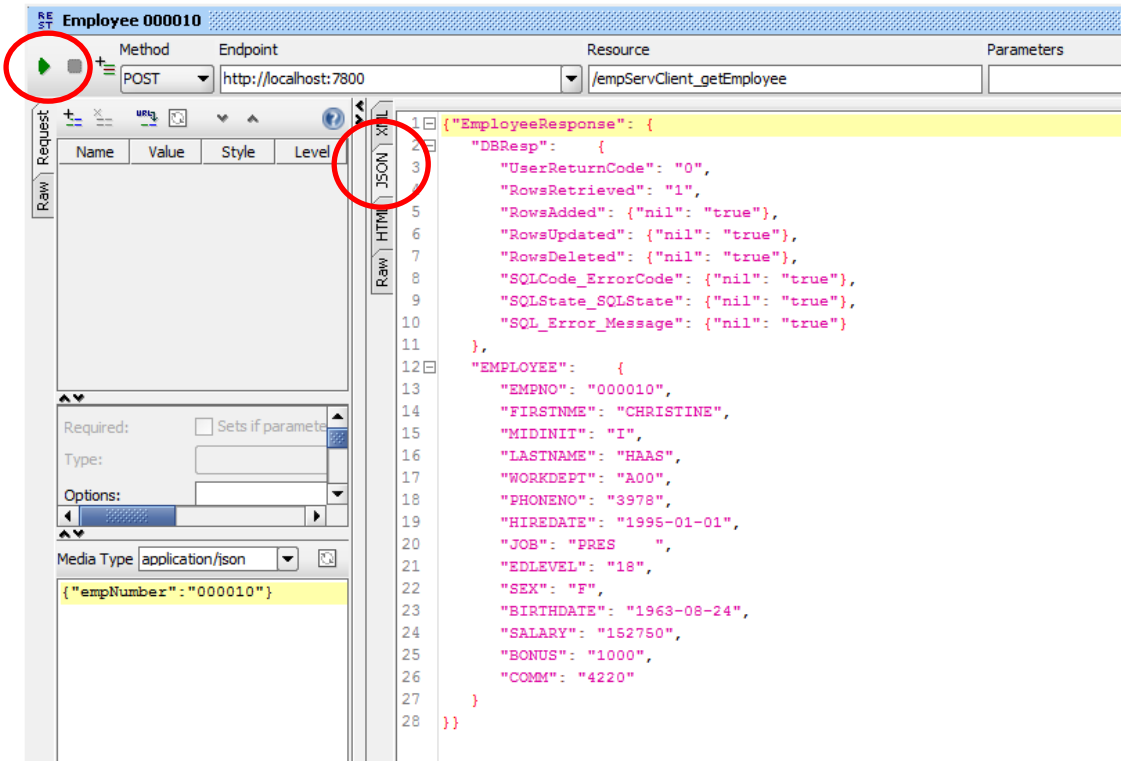
   Note that the endpoint url is localhost:7800. This port should be the one that your new application is deployed to, but if you have started other listeners for some other scenarios, you may need to adjust this endpoint.



   (For info, this SOAPUI project was created by specifying "New REST Project" in the SOAPUI dialogue).

2. Click the green arrow. The SOAPUI test runs, and the returned data will be shown in JSON format in the response pane.

   Make sure you select the JSON tab to see the message in JSON format.

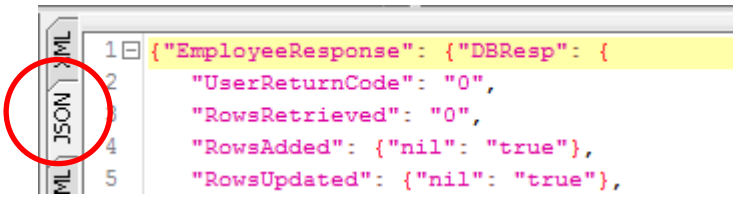


3. Repeat the test using the request "Employee 000011".

   The message payload that will be sent for this request is a JSON message:

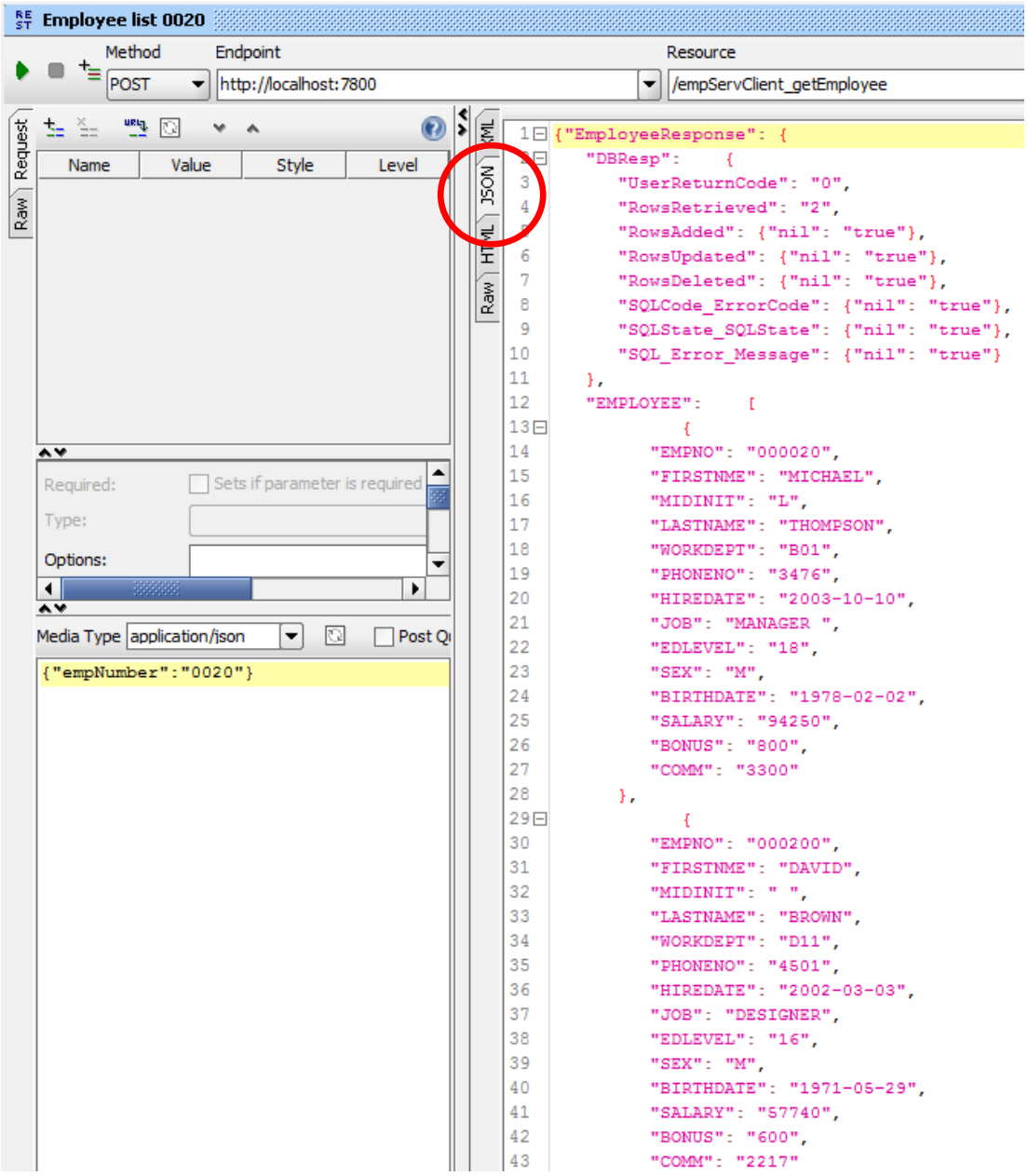
   {"empNumber":"000011"}

   SOAPUI will show the response message RowsRetrieved = 0



4. Open the SOAPUI request Employee list 0020. Click the green arrow.

   Two rows will be returned in JSON format.

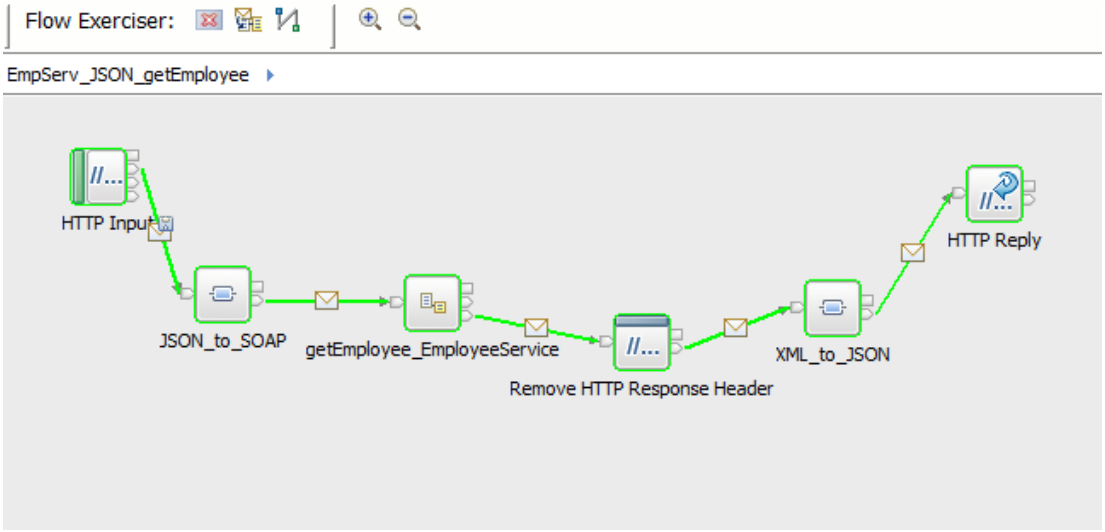


5. Back in the Integration Toolkit, the Flow Exerciser should still be active from the previous test method.

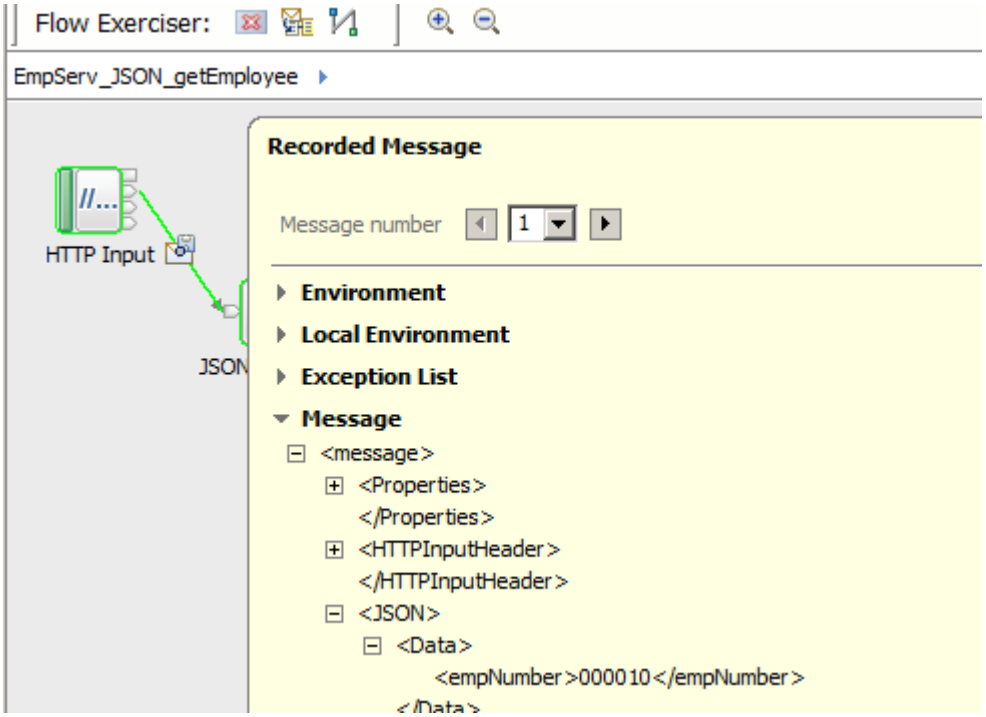
   Click the "View Path" icon, the third button of the Flow Exerciser line.

Provided by IBM BetaWorks

6. This will highlight the flow execution path as before. You can now perform the same tests as before to examine the message tree, as it was at various points in the flow execution.

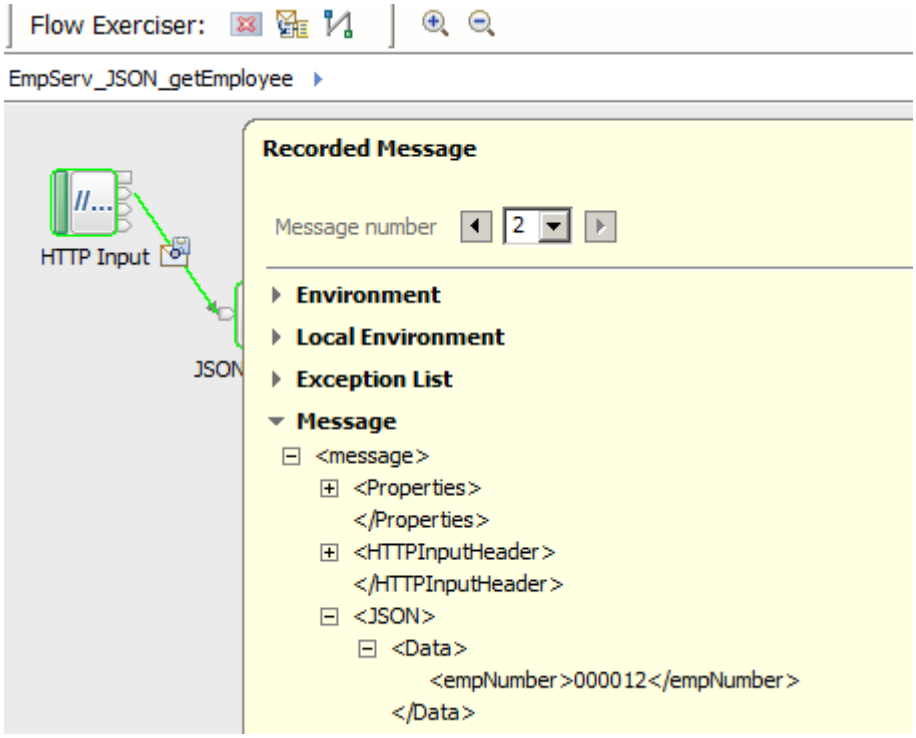


7. Retrieve the message from the first connector. You will see that the pop-up window now shows a "Message number". The first message is shown (empNumber 000010).



8. Move to the next message by clicking the arrow, or by selecting the message number.

   Expanding the Message will show the second message that was sent (empNumber 000011).



(Above screen capture for illustration only, different empNumber shown).

Explore the message tree on other connectors.

When finished, close the Flow Exerciser.

4. Adding data to the Environment Tree (optional)

This chapter shows you how to use the schemaless mapping tools in the Graphical Data Mapper to write data to the Environment tree.

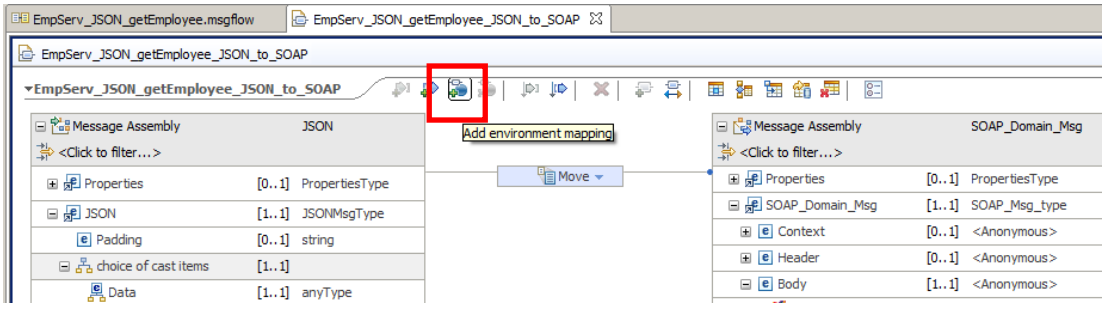
The Environment is a free-format area that allows any application to store data temporarily, for the duration of the message flow. Applications that want to write data to the Environment have to define the required elements themselves. For example, an ESQL Compute node will first define a new Environment element, and then write data to that element.

In IIB Version 10, the GDM node has introduced the ability to write data to the Environment, by providing support for schemaless mapping. This section uses the first Mapping node to write the employee number to the Environment. Later in the flow, a new Mapping node will retrieve this data from the Environment and use it to construct an output message for requests that result in a failure to retrieve data from the database.

4.1 Configure the first mapping node

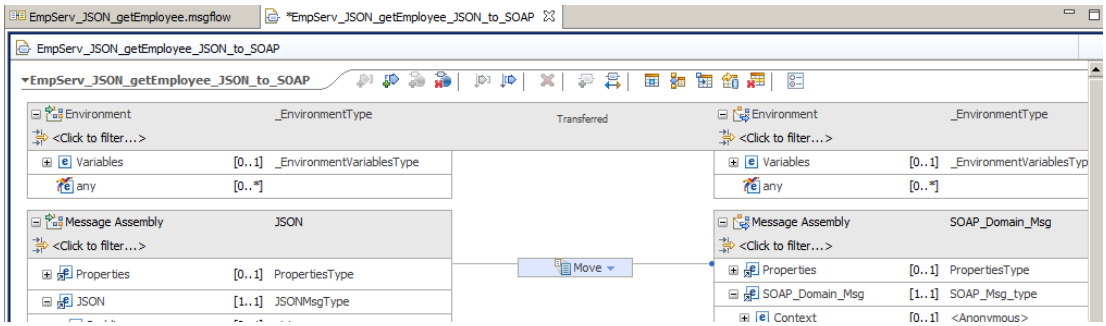
1. Reopen the first Mapping node, JSON_to_SOAP.

   By default, the map does not show the Environment tree. Use the icon on the map header line to add the Environment tree to the map editor.
   The icon is shown in the box below. Hover over the icon to confirm the correct icon. Click to add the Environment to the map.

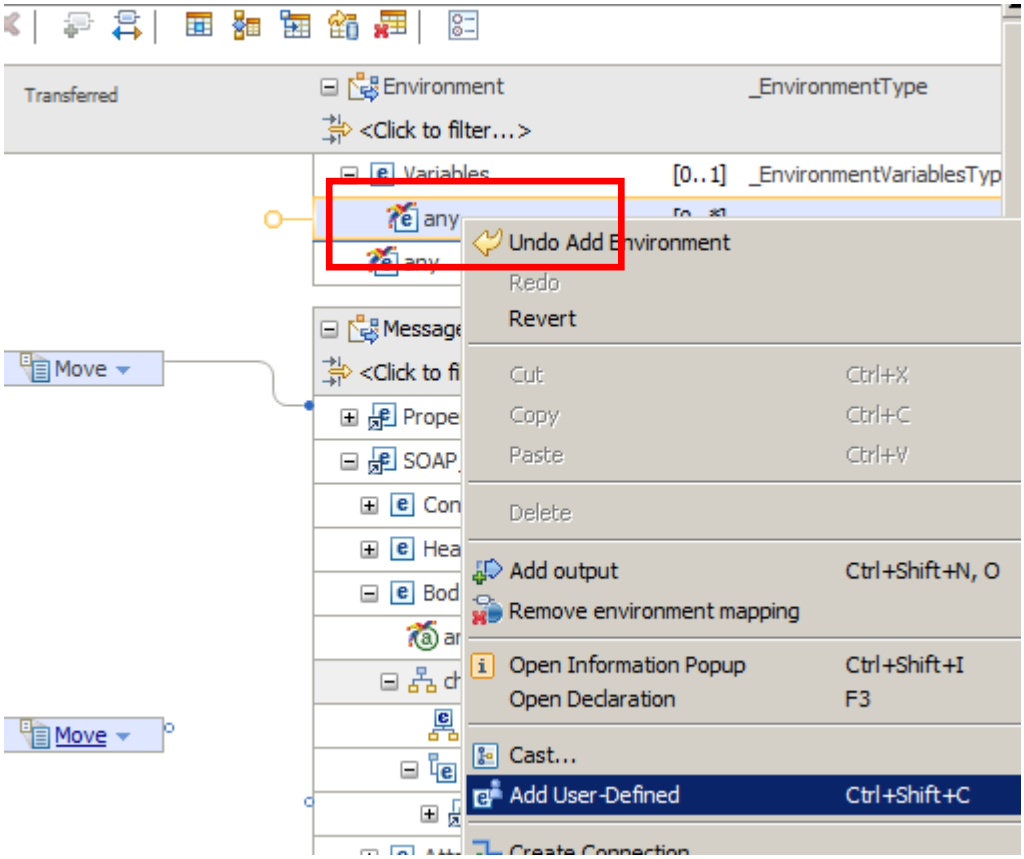


2. The message assemblies will have been updated to show the Environment.

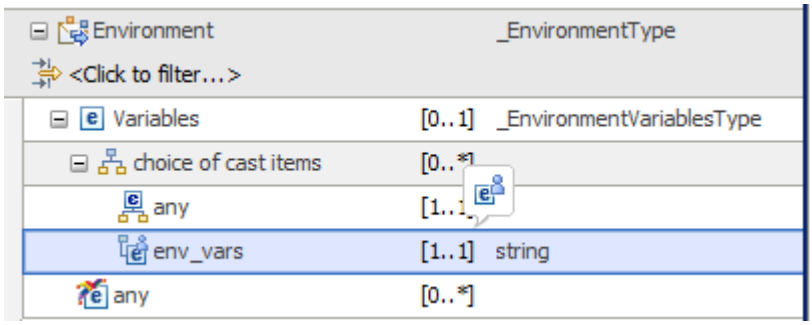
   Note that because the Environment is a common area, this is displayed at the top of the map, bridging the input and output assemblies.



3. In the output Environment/Variables message assembly, right-click the "any" element, and select Add User Defined.
(Do not click on the "any" element just below).



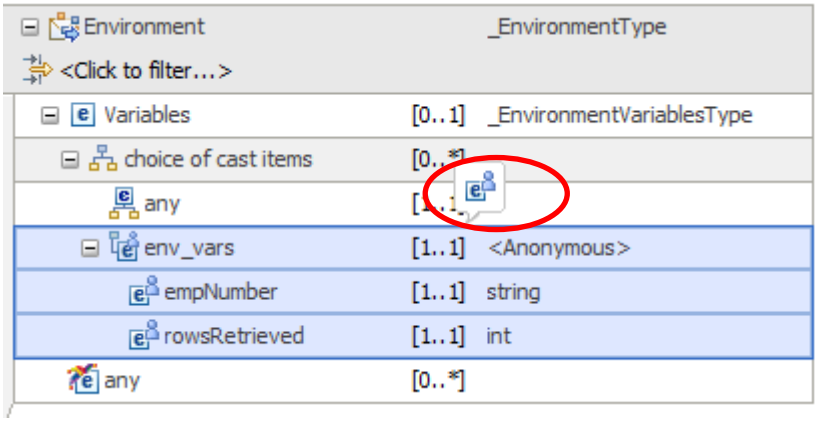
4. Name the new element env_vars. The type will initially be set to "string".



5. Using either the icon highlighted above the env_vars element, (or by using right-click, Add User-Defined), add a new child element named **empNumber**, type = string.

   Add a second child under env_vars, named **rowsRetrieved**, type = int.

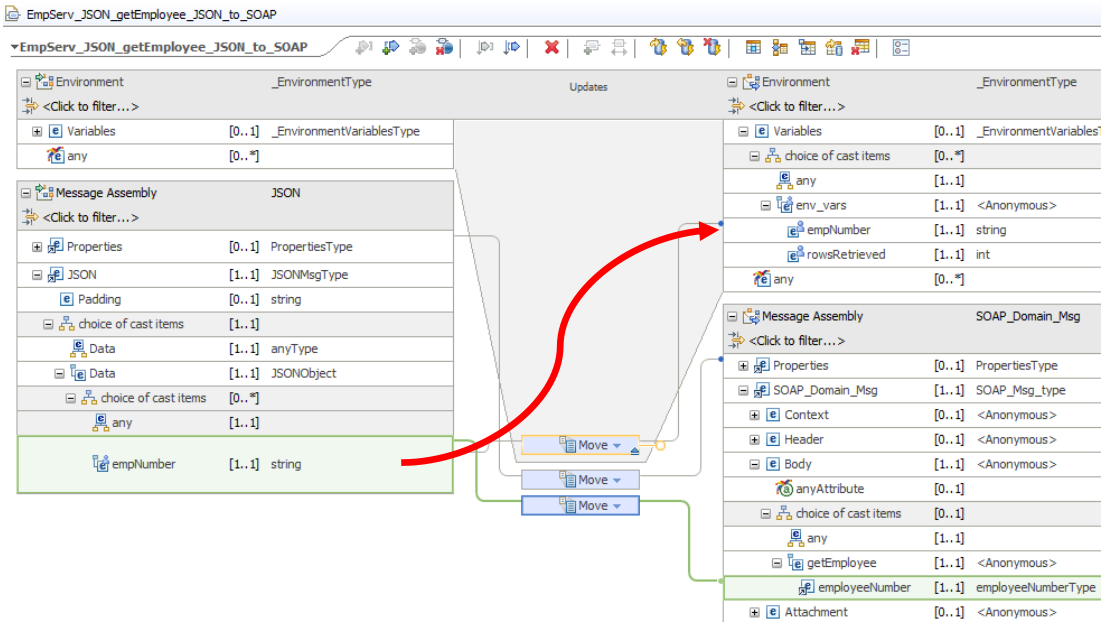
   Be careful to add **rowsRetrieved** under **env_vars**, not under empNumber.



6. Map the input **empNumber** to the Environment element **env_vars/empNumber**.

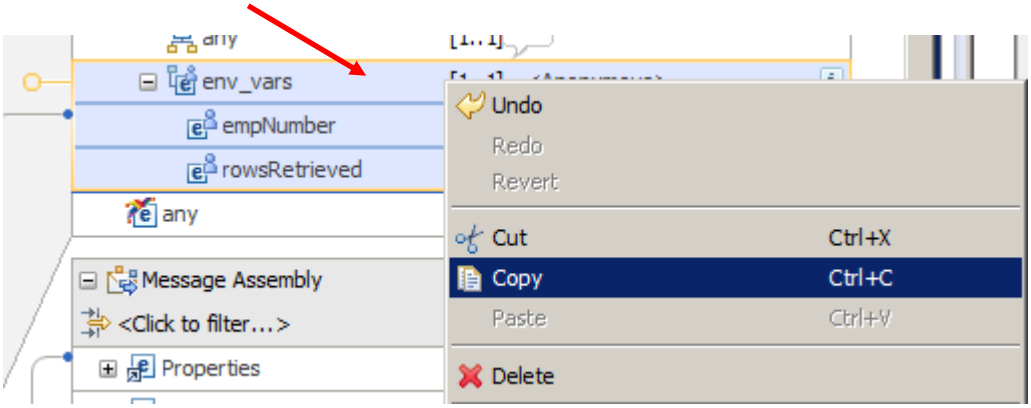
   Note that empNumber has been previously mapped to the SOAP output message. However, you can have two (or more) transforms applying to a single input element.

   Observe the way in which the Environment mappings are displayed in the map editor.



   This map will now save the input employee number to the Environment tree for use later in the flow.

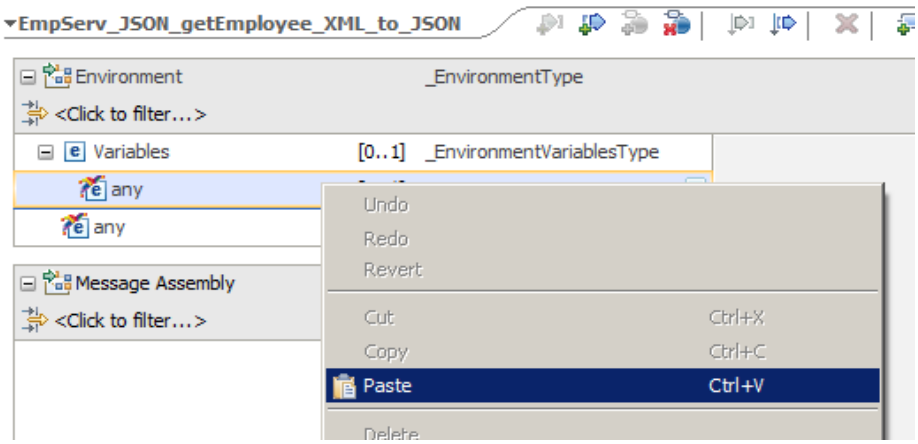
7. For use in later maps, right-click the **env_vars** element, and select Copy (or use Ctrl-C).



Save and Close the JSON_to_SOAP map.

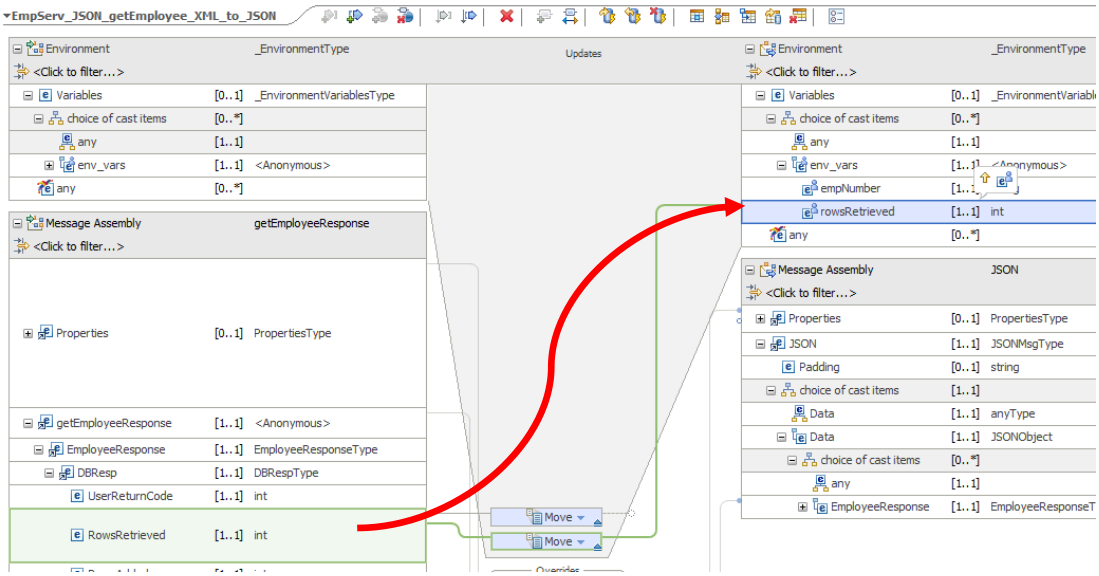
8. Open the map XML_to_JSON.

As before, add the Environment tree to the map, Expand the Variables folder, right-click the "any" element, and select Paste.



9. Map the input DBResp/RowsRetrieved to Environment/Variables/env_vars/rowsRetrieved (a Move transform).

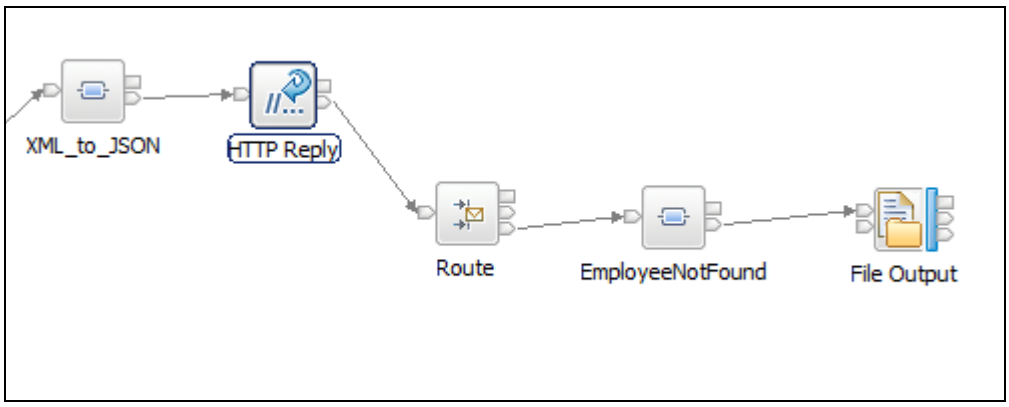
   Save and close the map.



4.2 Add and Configure new nodes

1. Add three new nodes to the flow (after the HTTP Reply node), and connect as shown.

   - Route node (connect the output Match terminal to the new Map node)
   - Mapping node - name EmployeeNotFound
   - File Output node



4.2.1 Configure the Route node

1. Configure the Route node.

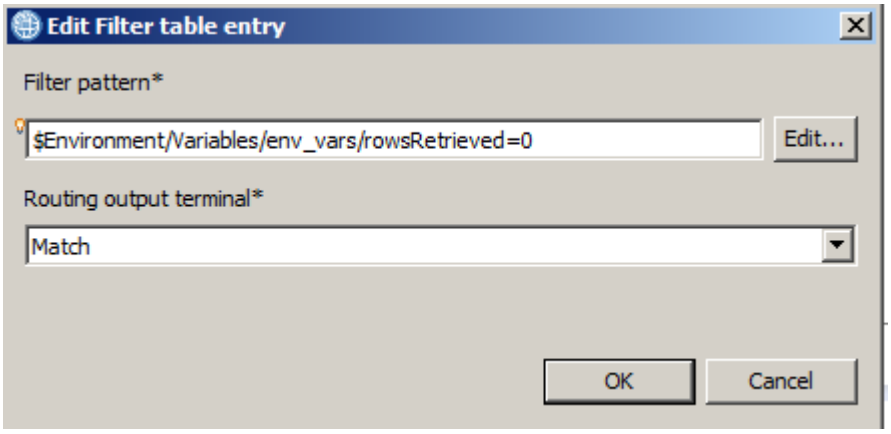
   Highlight the node, and in the node properties, set the Filter pattern to

   $Environment/Variables/env_vars/rowsRetrieved=0.

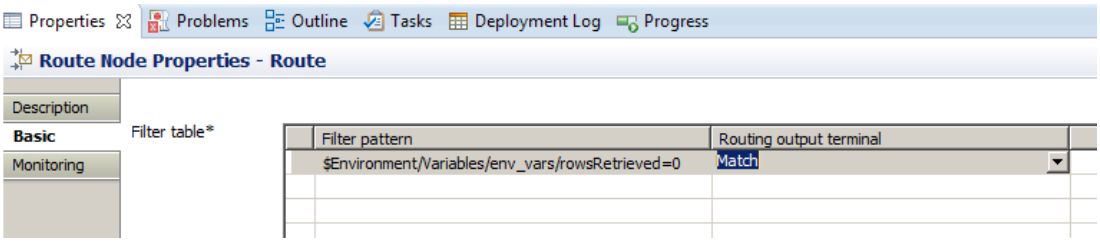
   Use the Add button to open a new filter pattern; you will manually type the above Filter.

   Note - this expression needs to exactly match the Environment element that you created earlier.

   Set the Routing output terminal to Match.



2. This test checks for env_vars/rowsRetrieved=0 being true. If no rows are returned, the message flow will log this event to a log file. Another approach might be to create a Monitoring event to capture this event.

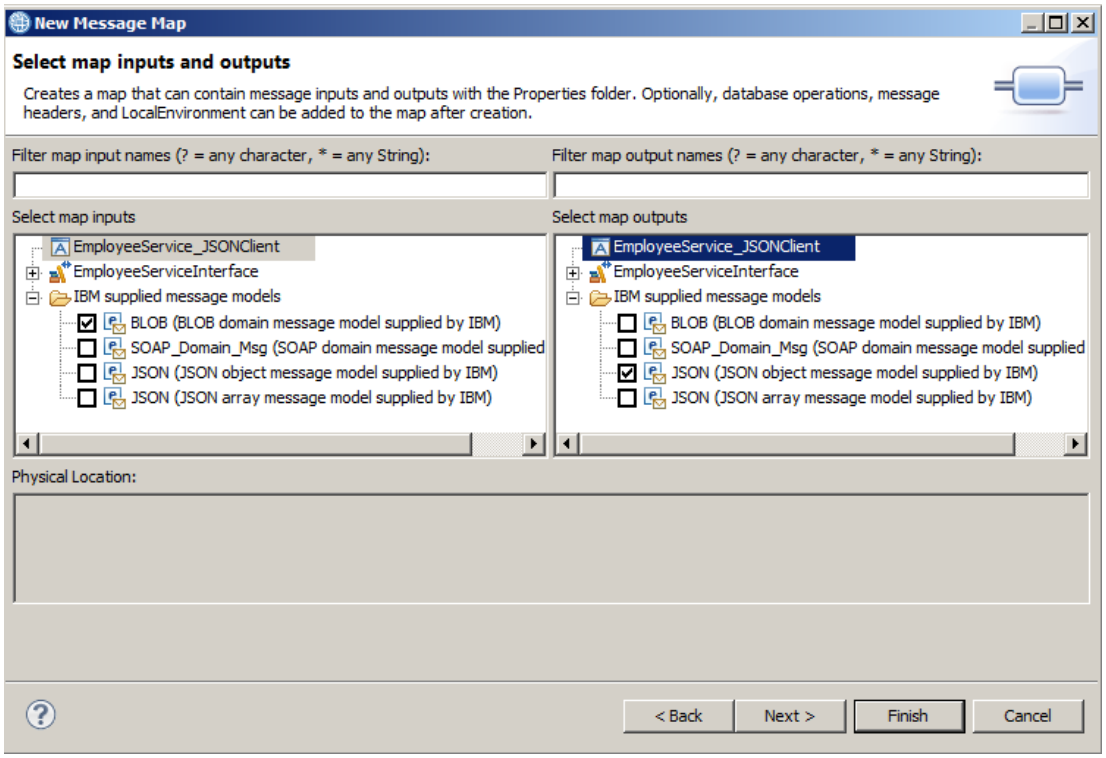


4.2.2 Configure the new Mapping node

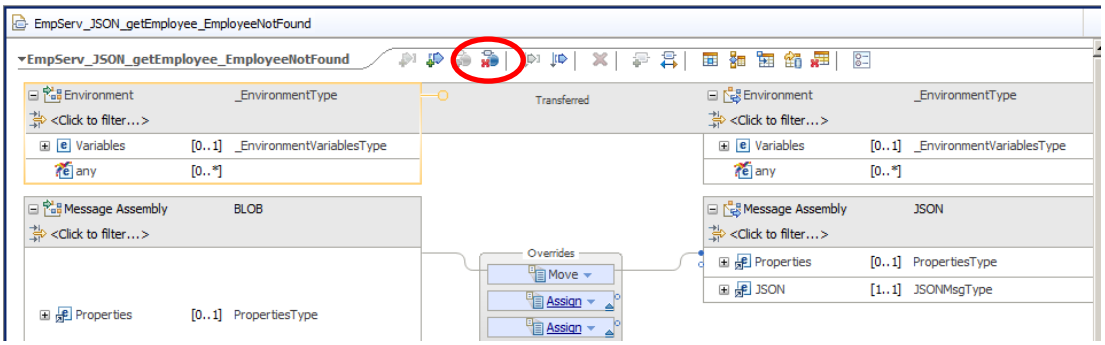
1. Configure the EmployeeNotFound mapping node. Double-click the node, click Next, and:

   - Set Input message assembly = BLOB
   - Set Output message assembly = JSON (object message)

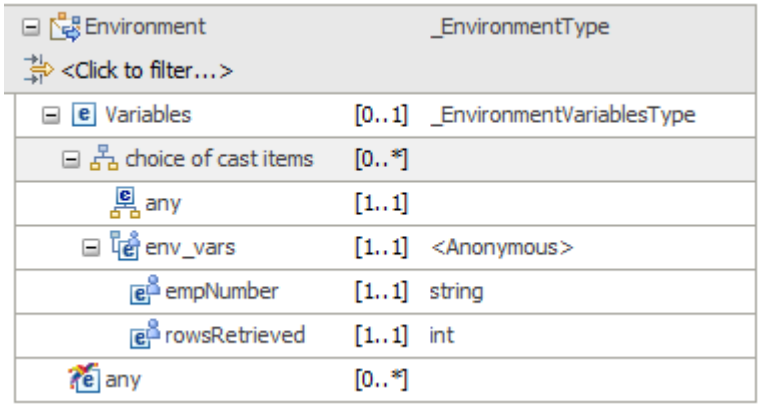
   Click Next, then Finish.



2. As before, add Environment Mapping to the map editor (icon on the map icon line).



3. Using the same technique as earlier, on the input message, right-click "Variables/any" and paste "**env_vars**". (You may need to recopy env_vars into the clipboard from one of the earlier maps).

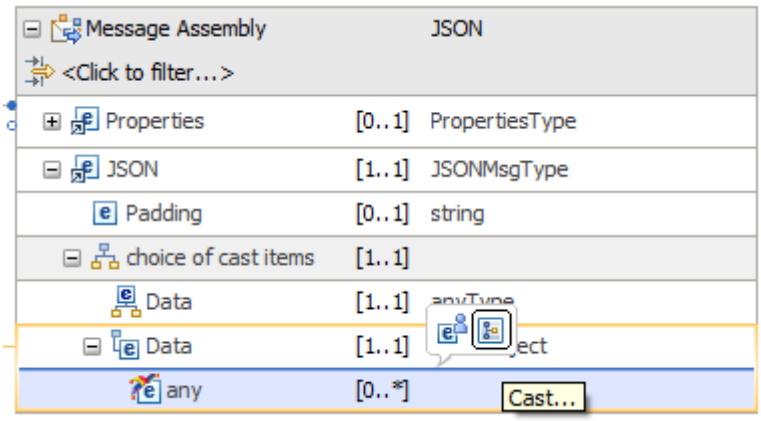


4. Now add some new elements to the JSON part of the output assembly. The elements that are needed are provided in a pre-built schema, so first, copy/paste the file

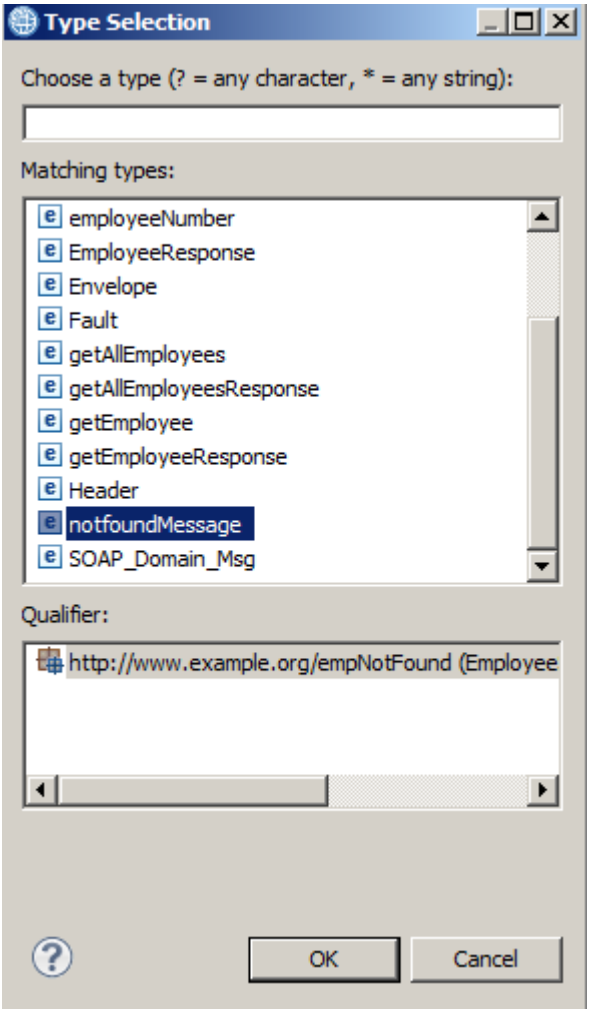
   `c:\student10\Integration_service_JSONClient\resources\empNotFound.xsd`

   from Windows Explorer onto the **EmployeeService_JSONClient** project in the IIB Toolkit.

5. In the map editor, highlight and hover over the JSON/Data/any element. Click the "Cast" icon. (Alternatively, right-click "any", select Cast).



6. From the Type Selection pop-up, select **notfoundMessage**, and click OK.



7. The result should be:

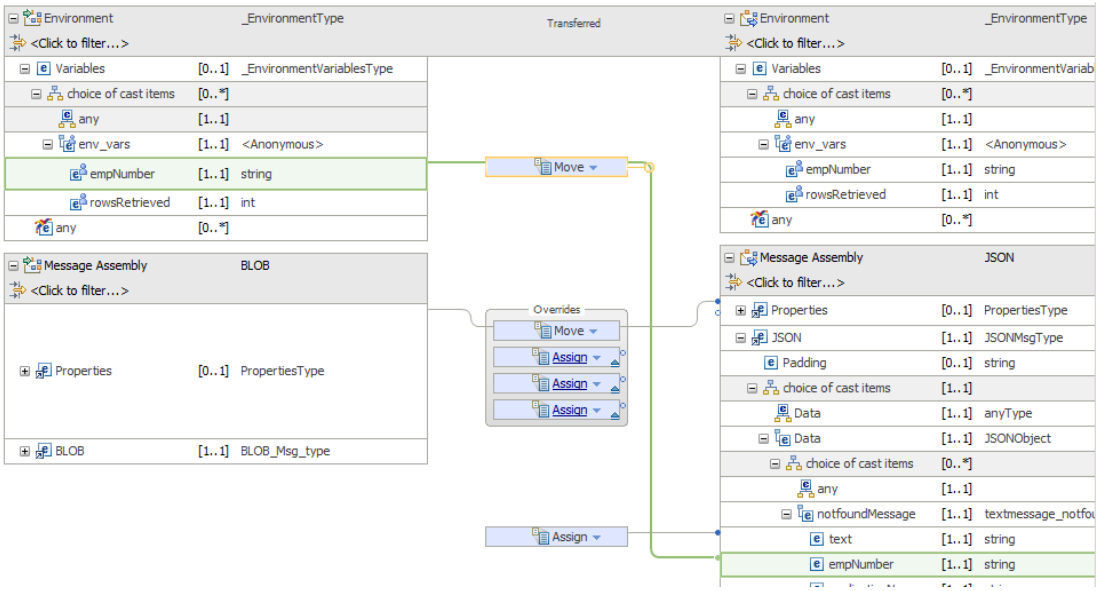
Data	[1..1]	JSONObject
choice of cast items	[0..*]	
any	[1..1]	
notfoundMessage	[1..1]	textmessage_notfound
text	[1..1]	string
empNumber	[1..1]	string
applicationName	[1..1]	string
flowName	[1..1]	string
dateTime	[1..1]	dateTime

4.2.3 Create the mapping transforms

1. Create the transform for the "text" element.

   Use an Assign (right-click text, select Add Assign) to set the value to "Employee record has been requested but not found in database table" (quotation marks not used in the editor).

2. Map the Environment **env_vars/empNumber** to output **notfoundMessage/empNumber** (a Move transform).



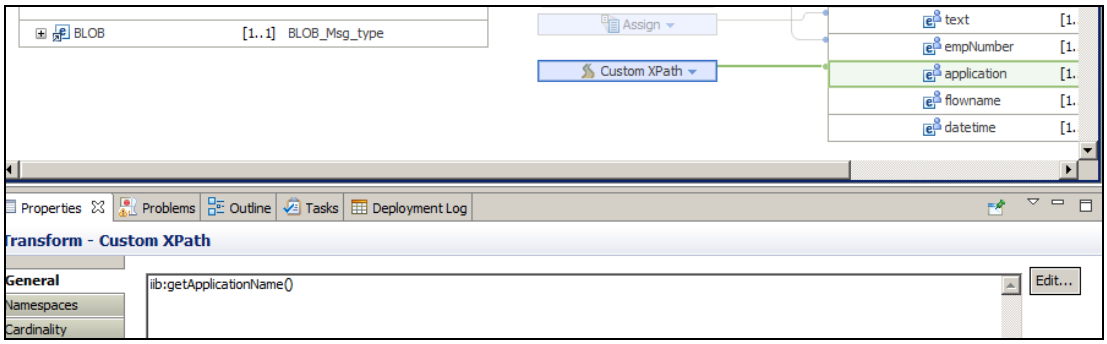
3. For the output **notfoundMessage/applicationName**, right-click "**applicationName**" and select "Add Assign".

   Change the Assign transform to Custom XPath.

   In the XPath properties (General), set the XPath statement to

   iib:getApplicationName()

   (Note - you can type "iib", then invoke Content Assist).



Provided by IBM BetaWorks

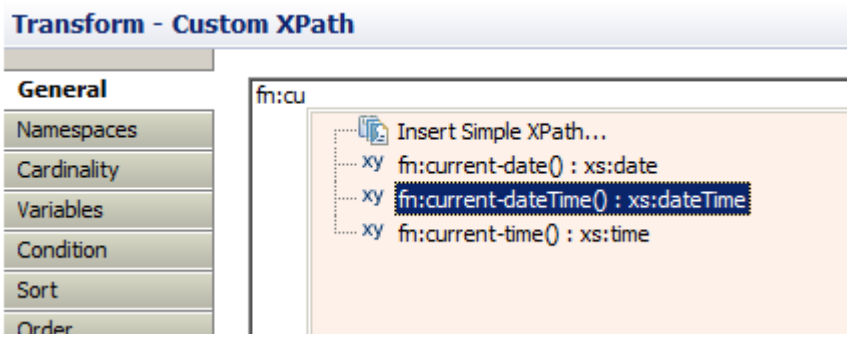
4. For the output "flowName", create a similar transform to application Set the XPath statement to

   iib:getMessageFlowName()

5. For the output dateTime, again create a Custom XPath transform.

   This time, use the content assist function ("fn:cu") to create the statement:

   fn:current-dateTime()

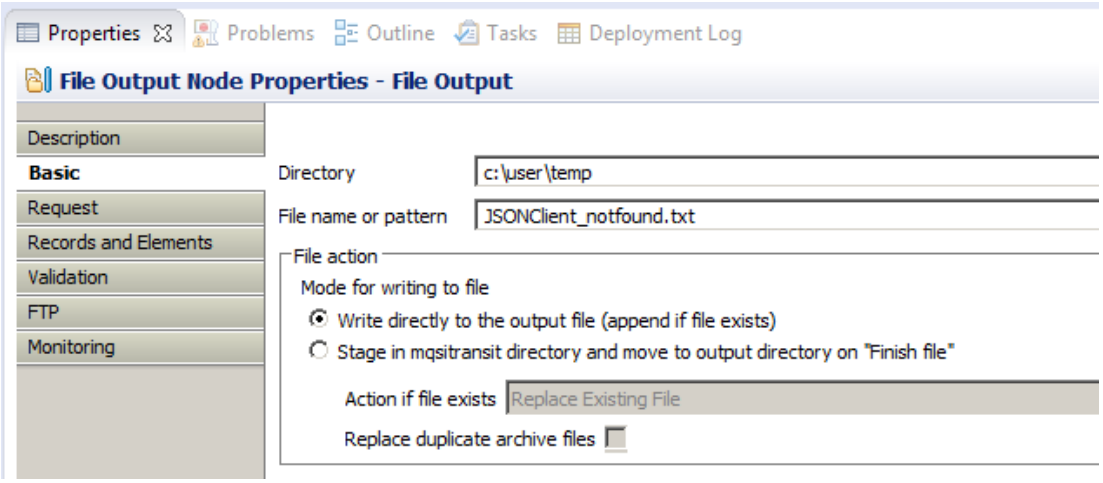


Save and close the map.

4.2.4 Configure the File Output node

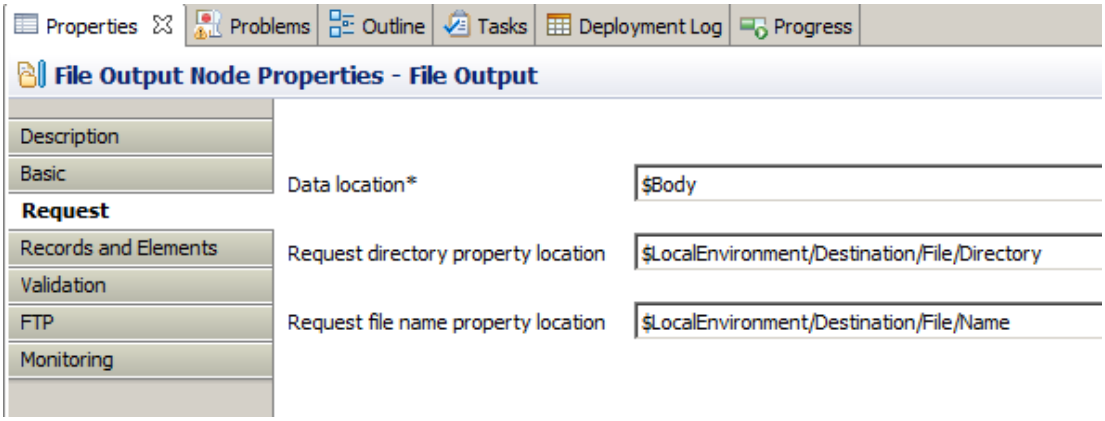
1. On the node properties Basic tab, set:

   - Directory = c:\user\temp
   - File name = JSONClient_notfound.txt
   - File action = Write directly to output file



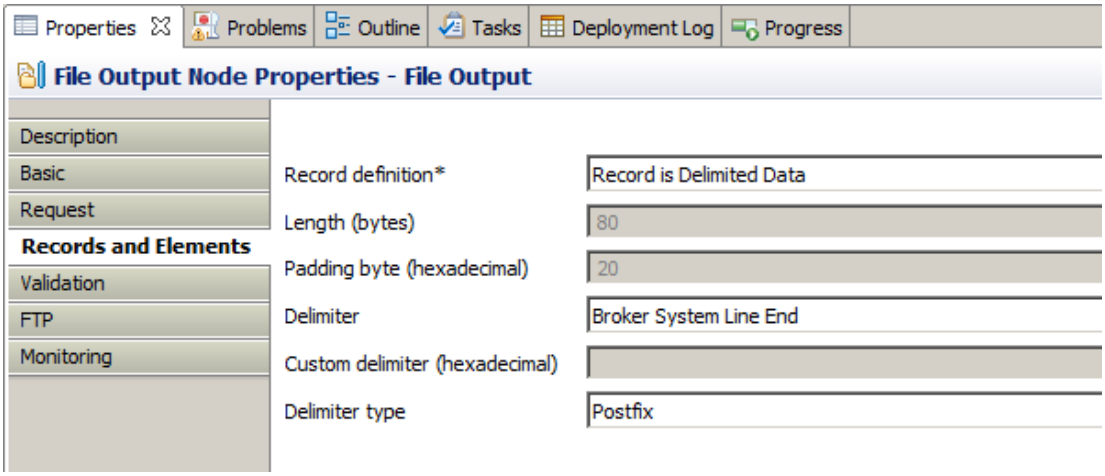
2. On the node properties Request tab, set:

   - Data location = $Body (this may already be set)



3. On the Records and Elements tab, set:

   - Record definition = Record is delimited data
   - Delimiter = Broker System Line End (default value)



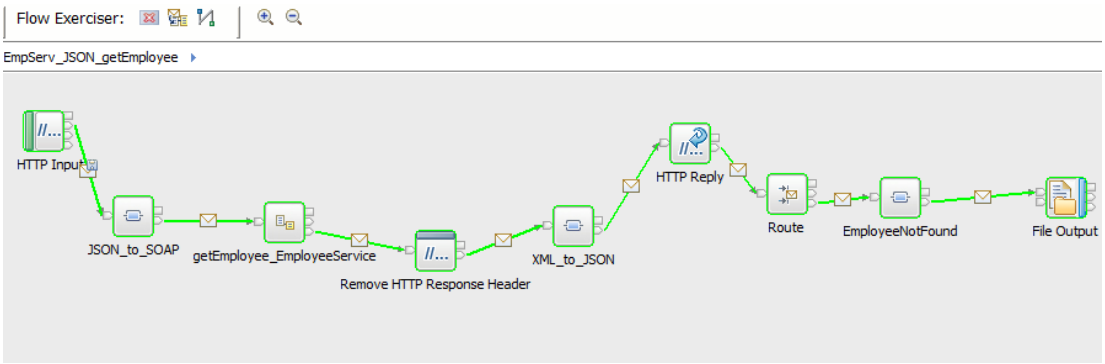
4.3 Deploy and Retest

1. Save the flow.

   Reinvoke the Flow Exerciser (click the red button, and then the Send Message button).

   Send a message containing a non-existent employee (eg. 000012).

   After closing the Send dialogue, the entire message flow should show green (including the final File Output node).



2. Open RFHUtil from the Start menu.

   Open the file c:\user \ temp \ JSONClient_notfound.txt.

   On the Data tab, select the JSON Data Format. You will see the contents of the message tree have been written to the output file, in JSON format.
   (Note - RFHUtil will only show one record; if you expect multiple records, you will need to open the file with Notepad).

   If data is not written to the file, check with the Flow Exerciser and retrieve messages from the connectors. You can view the values of the variables that you defined earlier and compare them with the expected values.

   Close the Flow Exerciser.

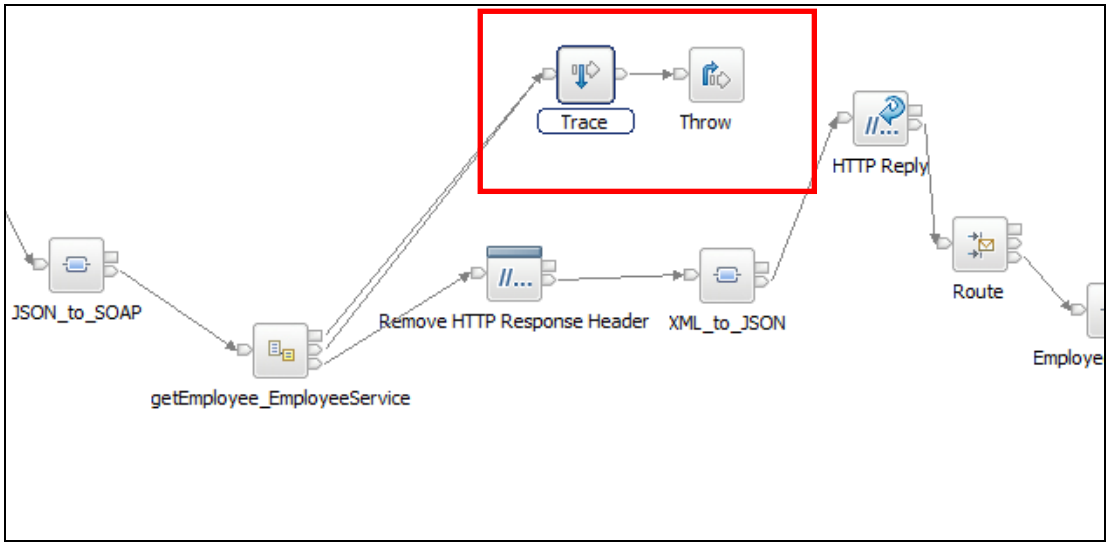
5. Add fault and failure handling (optional)

This application invokes a web service using a SOAP Request node. If the web service request fails (for example if the TCP/IP port has been set incorrectly, or the Integration Bus node or server is unavailable, the SOAP Request node will timeout after 2 minutes. A generic network error will be generated.

To better handle this type of failure, it is good practice to connect both the Fault and Failure terminals of the subflow that invokes the web service. This short optional section shows you how to do this.

1. In the primary message flow, add a Trace node and a Throw node to the flow, and connect as shown.

   Connect both the Failure and Fault terminals of the getEmployee_EmployeeService subflow to the Trace node.

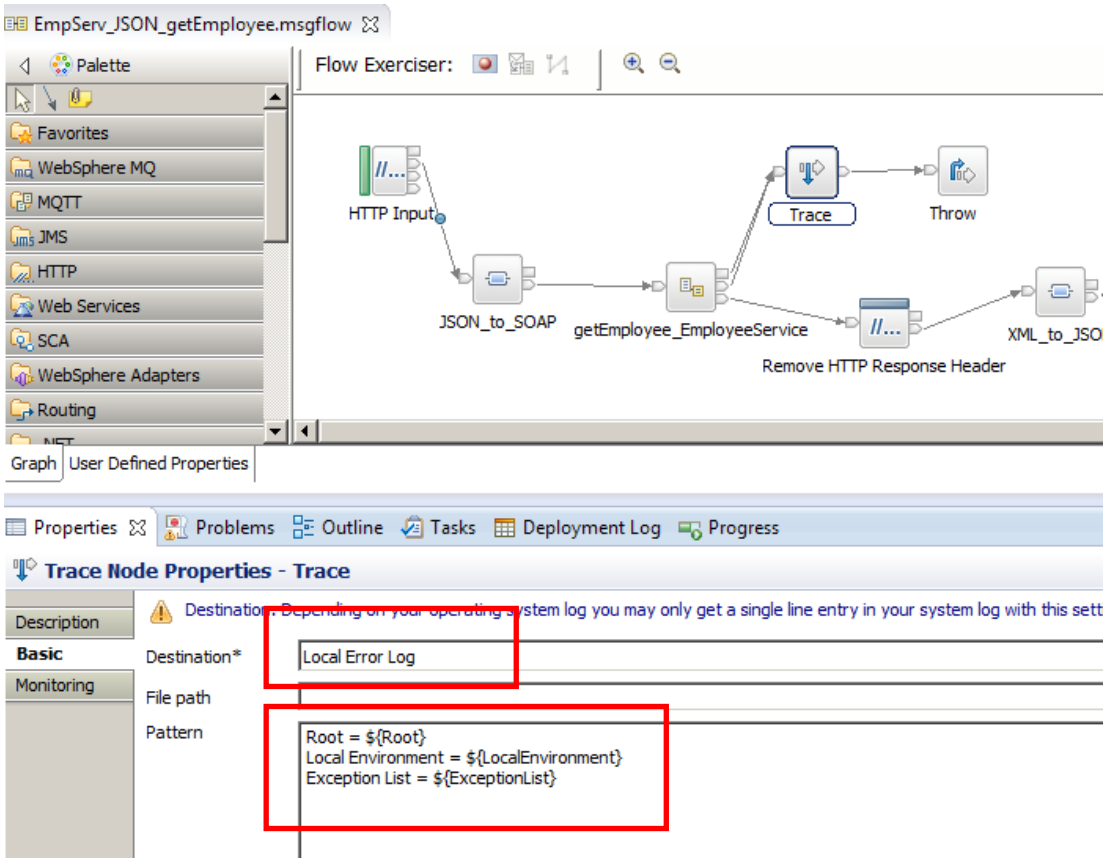


2. Highlight the Trace node, and set the node properties.

   Set the **Destination** to "Local Error Log". (in your own environment, you may want to set this to a file or user log output).

   Set the **Pattern** to:

   Root = ${Root}
   Local Environment = ${LocalEnvironment}
   Exception List = ${ExceptionList}



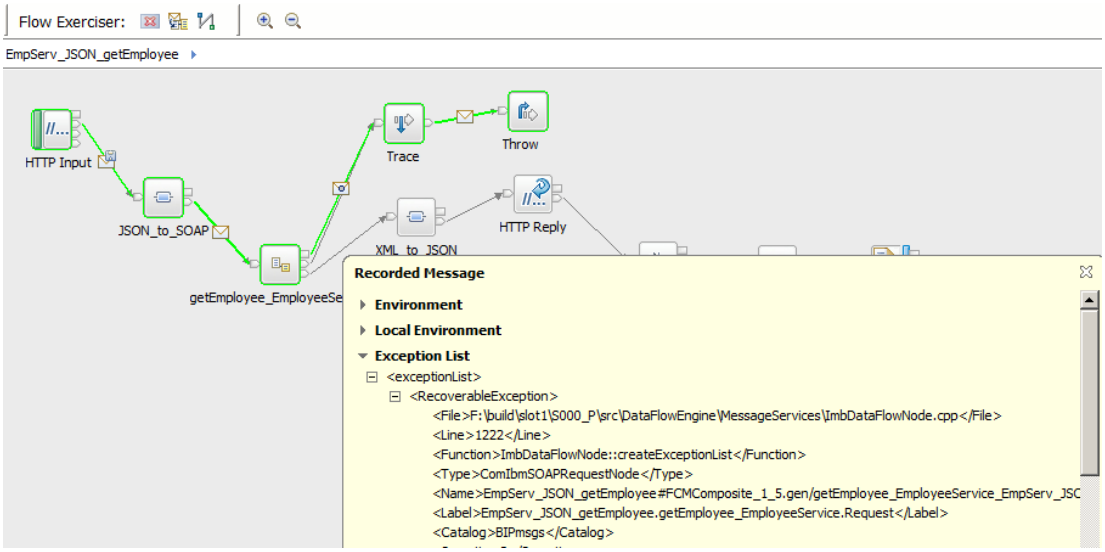
Provided by IBM BetaWorks

3. Save the flow.

   You can now simulate an error scenario by stopping the EmployeeService (use the Integration Nodes pane, right-click EmployeeService, select Stop.

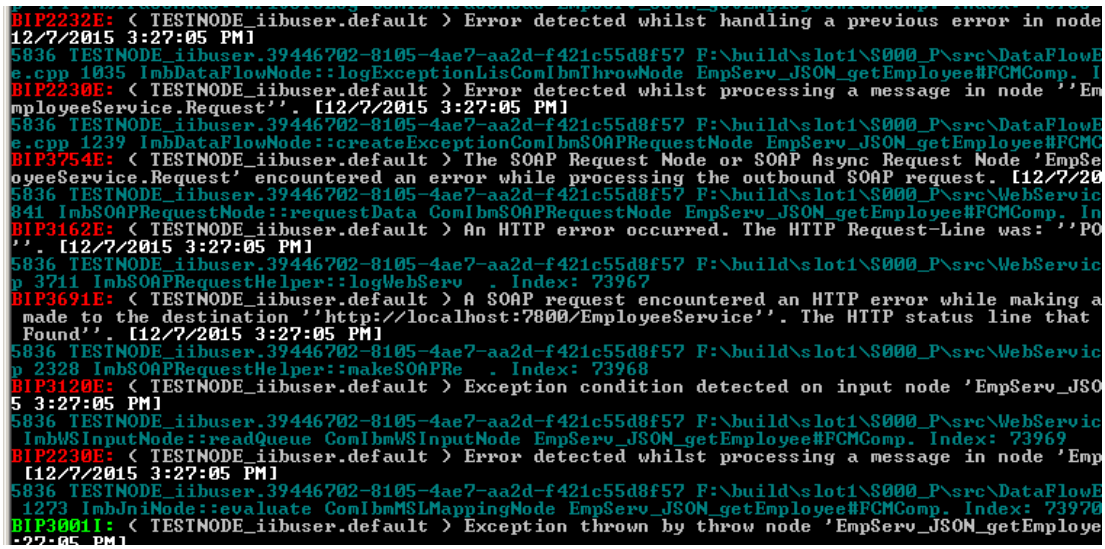
   Rerun the EmployeeService_JSONClient application using the Flow Exerciser. When the application attempts to invoke the EmployeeService, this will fail. The error will be caught, and the details will be displayed both in the Flow Exerciser, and in the Windows Event Log.

   Observe that the Flow Exerciser will show the green error path of the flow. You can click each connector to see the message contents. For example, the connector just before the Trace node will show the ExceptionList, with the reason for the error (near the bottom of the Exception List).



4. The IIB Error Log Console monitor will also show the thrown error message, with the various message components (Root, Environment, etc), and the details of the error message.

   Note that the root cause, EmployeeService is not available, is easily seen from the console log.



6. Invoking the EmployeeService integration service asynchronously (optional)

The first part of this lab invokes the EmployeeService synchronously. This is constructed by dropping the WSDL of the EmployeeService directly onto the flow editor of the JSON message flow. The Integration Toolkit uses this to include a SOAP Request node in the message flow (subflow).

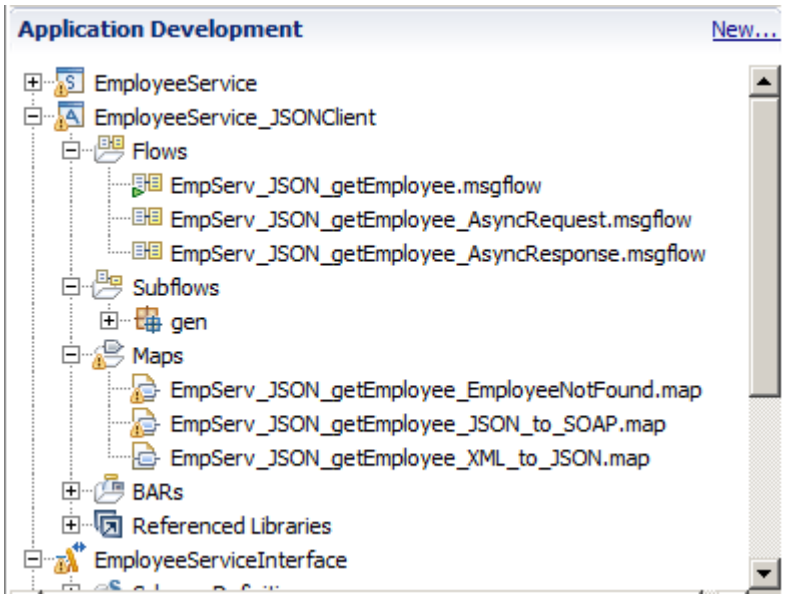
At execution time, the flow invokes the service synchronously. If the service is not available, then the JSON application will throw an error, and the client will not receive the requested data.

IBM Integration Bus provides the capability to invoke a web service asynchronously. In Version 8, this had to be done by using WS-Addressing, and in Versions 9 and 10, web services can be invoked asynchronously by simply providing a local identifier string that the IIB runtime uses to correlate the request and response flows, as shown here.

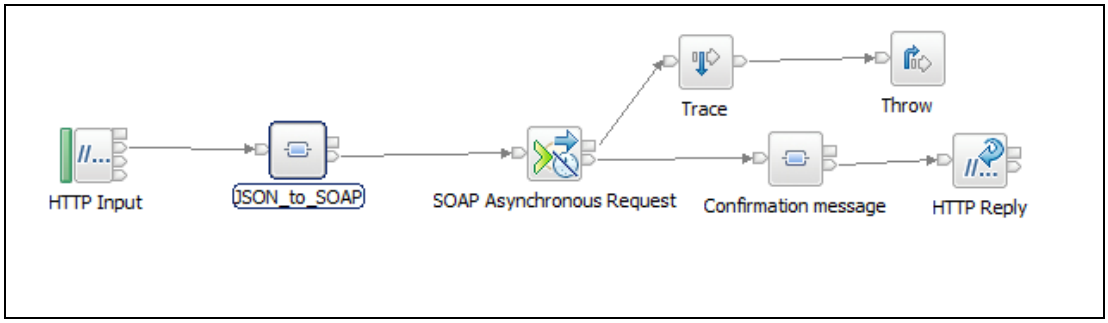
Since you are now an experienced IIB developer, we will omit some of the detailed screen captures.

1. In the EmployeeService_JSONClient application, create two new message flows:

   - EmpServ_JSON_getEmployee_AsyncRequest
   - EmpServ_JSON_getEmployee_AsyncResponse



2. For the getEmployee_AsyncRequest flow, construct the flow as shown here.



Set the node properties as follows. Make sure you carefully set all properties specified here.

HTTP Input

- Basic: Path suffix for URL: /empServClient_getEmployee_async
- Input Message Parsing: Message domain = JSON

Mapping node: JSON_to_SOAP

- Basic: Mapping routine = {default}:EmpServ_JSON_getEmployee_JSON_to_SOAP (use the Browse button to select this)
- Validation: Validate = None

SOAP Async Request

- To populate the node properties here, just drag/drop the EmployeeService.wsdl onto the SOAP Request node.
- Set the Basic Unique identifier = EmpServ123
- HTTP Transport: Use HTTP asynchronous request-response = Ticked

Trace node

- Destination = Local Error Log
- Pattern = Exception List: ${ExceptionList}

Mapping node: Confirmation message - see below

Save the flow.

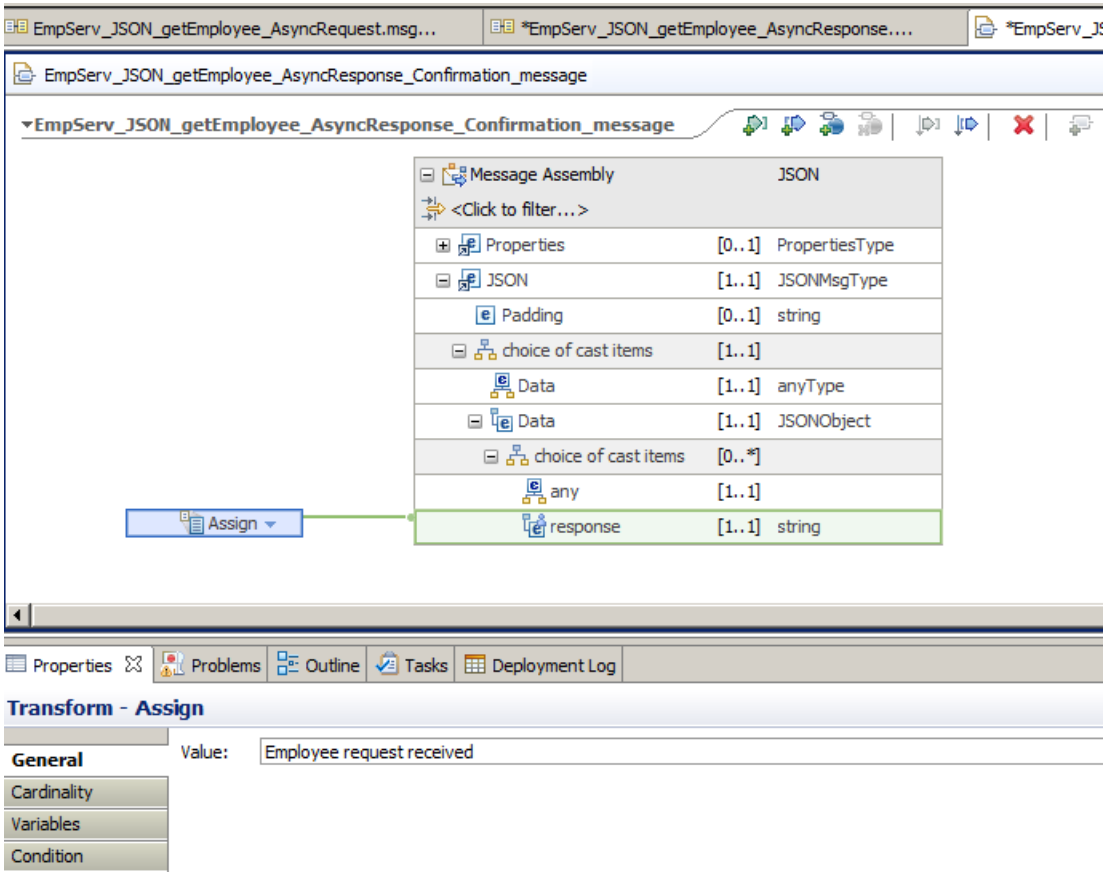
3. Configure the new mapping node: Confirmation Message

   Open this new map and select just an output message, of type JSON (JSON object).

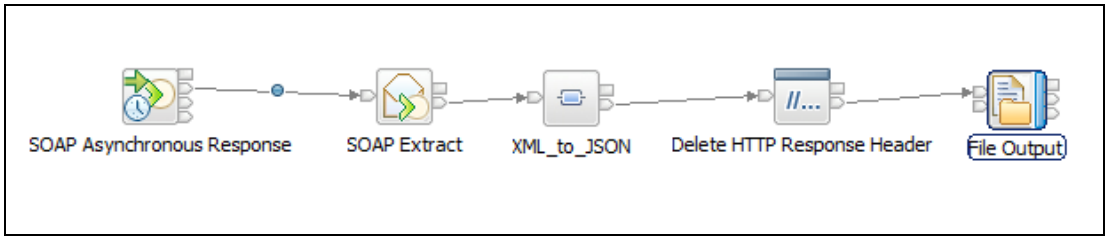
   In the mapping editor, under the JSON/Data/Data/any assembly, create a new element called "response".

   Assign a value to this element using an Assign transform. Set the value to "Employee request received".

   Save and close the mapping editor.



4. For the getEmployee_AsyncResponse flow, construct the flow as shown here:



Set the node properties as follows:

SOAP Async Response
- Basic: Unique identifer = EmpServ123
- Validation: Validate = None

Mapping node: XML_to_JSON
- Basic: Mapping routine = {default}:EmpServ_JSON_getEmployee_XML_to_JSON (use the Browse button to select this)
- Validation: Validate = None

Delete HTTP Response Header
- HTTPResponse tab: Delete Header

File Output
- Basic: Directory = c:\user\temp\
- File name = JSONClient_getEmployeeAsync.txt
- File action: Write directly to output file

Save the flow.

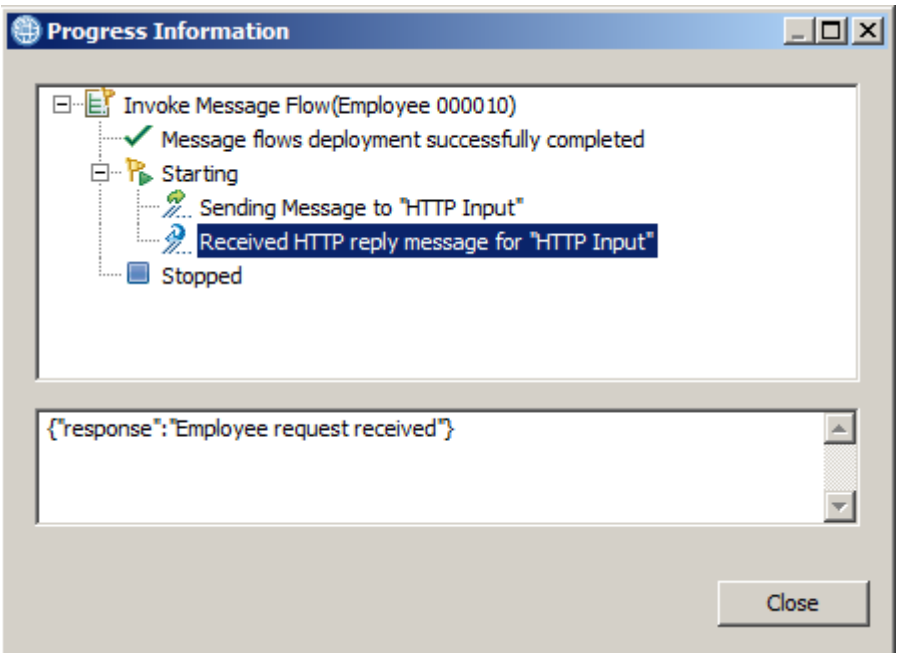
5. EmployeeService is probably Stopped at this point (when you did the Error Handling section of this lab), so to make sure the asynchronous service executes fully, start the EmployeeService now. You can use the Toolkit to do this (right-click the service, Start).

6. Test the new flows with the Flow Exerciser. Note - with IIB V10.0, Asynchronous Web Service Input nodes are not supported with the Flow Exerciser.

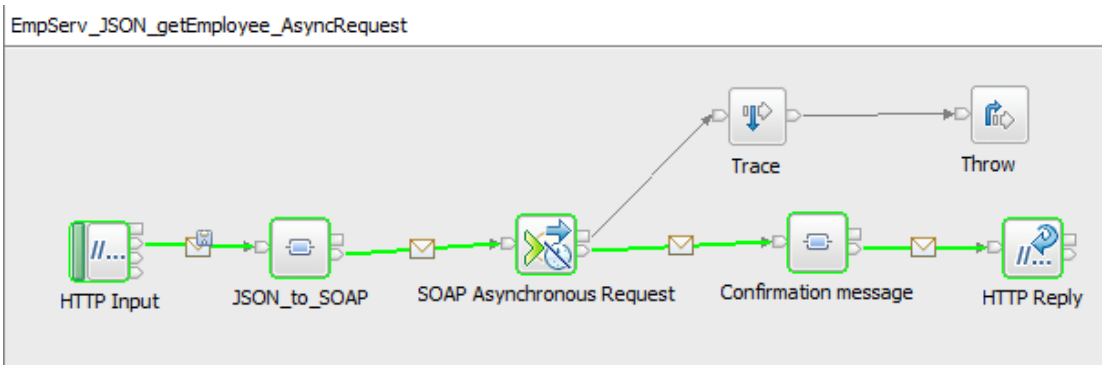
   However, you can invoke the initial request using the Flow Exerciser, so in the AsyncRequest message flow, start the Flow Exerciser.

   In the Flow Exerciser, send the message {"empNumber":"000010"}.

   The Progress Information will show the response from the first message flow, indicating that the request has been received.

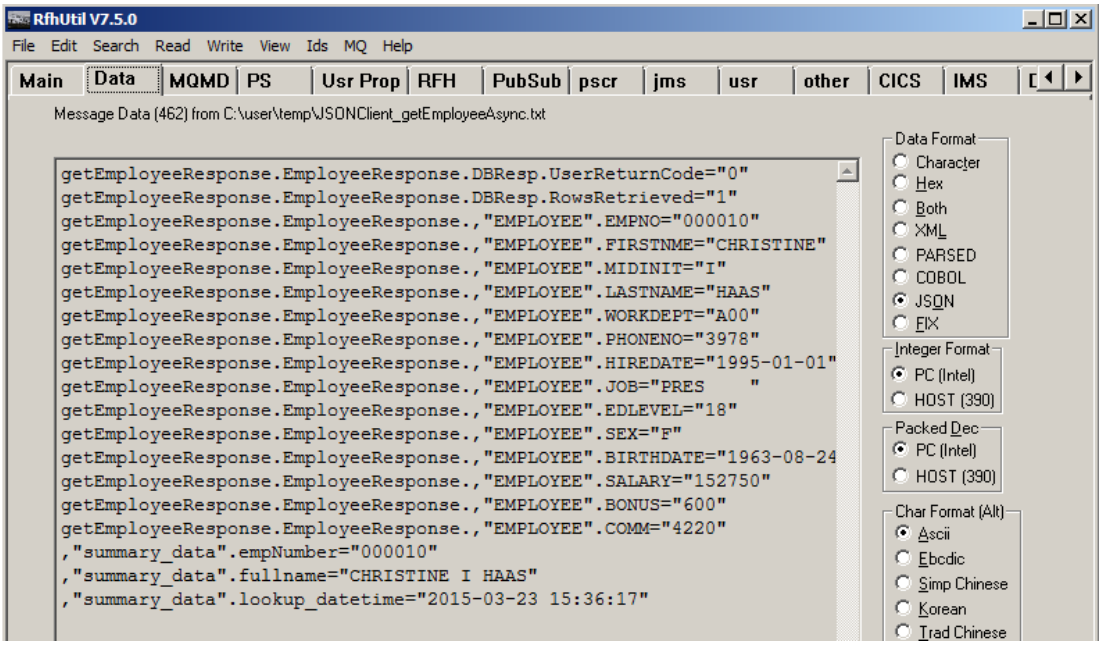


   The Flow Exerciser will again show the execution path of the flow.



7. In RFHUtil, open the file c:\user\temp\ JSONClient_getEmployeeAsync.txt.

   On the Data tab, look at the data retrieved by the async web service.



8. If you get the following error ("202 Ack expected"), you have probably forgotten to select "Use HTTP asynchronous request-response" on the SOAP Async Request node.

```
    )
        (0x01000000:Name     ):RecoverableException = (
          (0x03000000:NameValue):File     =
'F:\build\slot1\S000_P\src\WebServices\WSLibrary\ImbSOAPRequestHelper.cpp'
(CHARACTER)
          (0x03000000:NameValue):Line     = 2350 (INTEGER)
          (0x03000000:NameValue):Function =
'ImbSOAPRequestHelper::makeSOAPRequest' (CHARACTER)
          (0x03000000:NameValue):Type     = '' (CHARACTER)
          (0x03000000:NameValue):Name     = '' (CHARACTER)
          (0x03000000:NameValue):Label    = '' (CHARACTER)
          (0x03000000:NameValue):Catalog  = 'BIPmsgs' (CHARACTER)
          (0x03000000:NameValue):Severity = 3 (INTEGER)
          (0x03000000:NameValue):Number   = 3692 (INTEGER)
          (0x03000000:NameValue):Text     = 'AsyncRequest HTTP Error
returned - 202 Ack expected' (CHARACTER)
          (0x01000000:Name     ):Insert   = (
            (0x03000000:NameValue):Type = 5 (INTEGER)
            (0x03000000:NameValue):Text = 'HTTP/1.1 500 Internal Server
Error' (CHARACTER)
          )
          (0x01000000:Name     ):Insert   = (
            (0x03000000:NameValue):Type = 5 (INTEGER)
            (0x03000000:NameValue):Text =
'http://localhost:7800/EmployeeService' (CHARACTER)
```

END OF LAB GUIDE