



## IBM Integration Bus

# A JSON Application invoking an Integration Service

featuring

- Graphical Data Mapper with schemaless mapping

- JSON input/output

- Mapping element concatenation and Custom XPath

- Accessing runtime information from the mapping node

## October 2014

Hands-on lab built at product

beta code level version 10.0.467.0

For use during IIB V.Next beta program only

|  |           |
|--|-----------|
| <b>1. PREPARE THE PROVIDER SERVICE .....</b>   | <b>3</b>  |
| 1.1 OPEN THE WINDOWS LOG MONITOR FOR IIB .....   | 3         |
| 1.2 IMPORT AND DEPLOY EMPLOYEESERVICE .....  | 4         |
| <b>2. CREATE THE CLIENT APPLICATION .....</b>  | <b>5</b>  |
| 2.1 IMPLEMENT THE MAPPING NODES.....   | 11        |
| 2.1.1 <i>Create the Mappings</i> .....   | 27        |
| 2.1.2 <i>Optional extra mapping (Concatenate, Custom XPath, Retrieve flow details)</i> ..... | 28        |
| 2.1.3 <i>Custom XPath</i> .....  | 31        |
| 2.2 COMPLETE THE MESSAGE FLOW .....  | 33        |
| <b>3. TEST THE EMPSEVCLIENT_JSON1 APPLICATION .....</b>                                      | <b>37</b> |
| 3.1 TEST USING SOAPUI .....  | 37        |
| 3.2 TEST USING NODE.JS WEB BROWSER APPLICATION (OPTIONAL) .....                              | 39        |
| <b>4. ADDING DATA TO THE ENVIRONMENT TREE (OPTIONAL) .....</b>                               | <b>42</b> |
| 4.1 CONFIGURE THE FIRST MAPPING NODE .....   | 42        |
| 4.2 ADD AND CONFIGURE NEW NODES .....  | 46        |
| 4.2.1 <i>Configure the Route node</i> .....  | 46        |
| 4.2.2 <i>Configure the new Mapping node</i> .....  | 47        |
| 4.2.3 <i>Create the mapping transforms</i> .....   | 51        |
| 4.2.4 <i>Configure the File Output node</i> .....  | 53        |
| 4.3 DEPLOY AND RETEST .....  | 54        |
| <b>5. ADD FAULT AND FAILURE HANDING (OPTIONAL) .....</b>                                     | <b>55</b> |
| <b>6. INVOKING EMPLOYEESERVICE INTEGRATION SERVICE ASYNCHRONOUSLY (OPTIONAL)</b>             | <b>58</b> |

# 1. Prepare the Provider Service

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

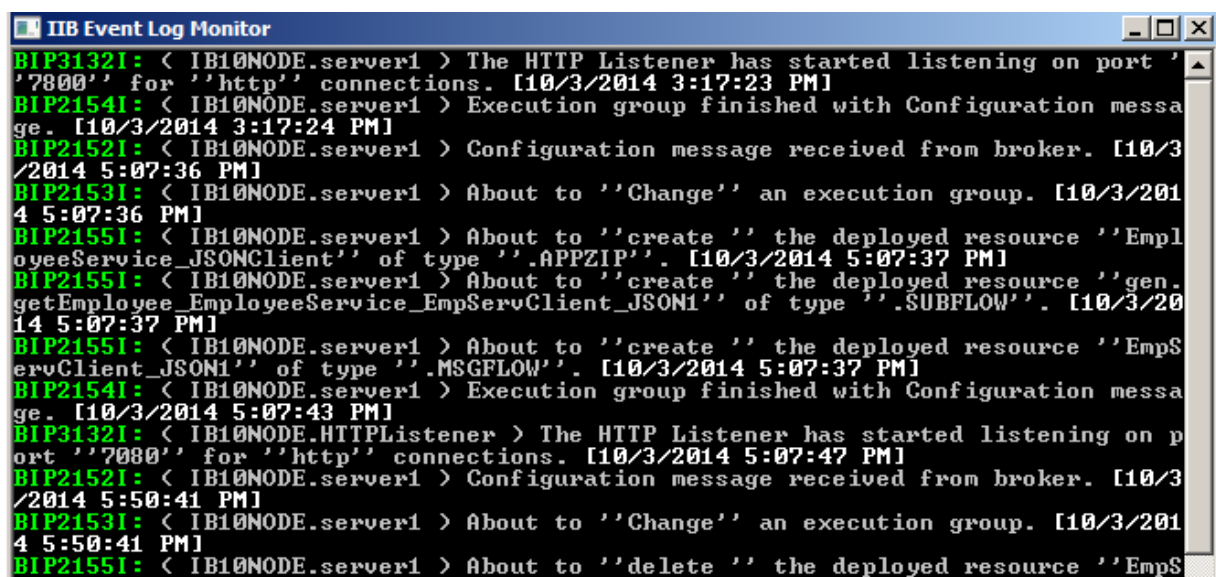
1. Use a web service connection to invoke an existing Integration Service (EmployeeService).
2. Use the Graphical Data Mapper to demonstrate schemaless mapping, mapping JSON input and output data.
3. Use the Graphical Data Mapper to write and read elements to the Environment tree.
4. Use the Graphical 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 the prebuilt solution of EmployeeService, and deploy it to the broker node.

## 1.1 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.



```
IIB Event Log Monitor
BIP3132I: < IB10NODE.server1 > The HTTP Listener has started listening on port '
'7800' for 'http' connections. [10/3/2014 3:17:23 PM]
BIP2154I: < IB10NODE.server1 > Execution group finished with Configuration messa
ge. [10/3/2014 3:17:24 PM]
BIP2152I: < IB10NODE.server1 > Configuration message received from broker. [10/3
/2014 5:07:36 PM]
BIP2153I: < IB10NODE.server1 > About to 'Change' an execution group. [10/3/201
4 5:07:36 PM]
BIP2155I: < IB10NODE.server1 > About to 'create ' the deployed resource 'Empl
oyeeService_JSONClient' of type '.APPZIP'. [10/3/2014 5:07:37 PM]
BIP2155I: < IB10NODE.server1 > About to 'create ' the deployed resource 'gen.
getEmployee_EmployeeService_EmpServClient_JSON1' of type '.SUBFLOW'. [10/3/20
14 5:07:37 PM]
BIP2155I: < IB10NODE.server1 > About to 'create ' the deployed resource 'EmpS
ervClient_JSON1' of type '.MSGFLOW'. [10/3/2014 5:07:37 PM]
BIP2154I: < IB10NODE.server1 > Execution group finished with Configuration messa
ge. [10/3/2014 5:07:43 PM]
BIP3132I: < IB10NODE.HTTPListener > The HTTP Listener has started listening on p
ort '7800' for 'http' connections. [10/3/2014 5:07:47 PM]
BIP2152I: < IB10NODE.server1 > Configuration message received from broker. [10/3
/2014 5:50:41 PM]
BIP2153I: < IB10NODE.server1 > About to 'Change' an execution group. [10/3/201
4 5:50:41 PM]
BIP2155I: < IB10NODE.server1 > About to 'delete ' the deployed resource 'EmpS
```

## 1.2 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 "ES\_JSONClient", or similar.

2. Import the PI file c:\student10\integration\_service\solution\EmployeeService.467.zip.
3. In the EmployeeService service, expand Resources\BARs, and deploy EmployeeService.bar to IB10NODE/server1.

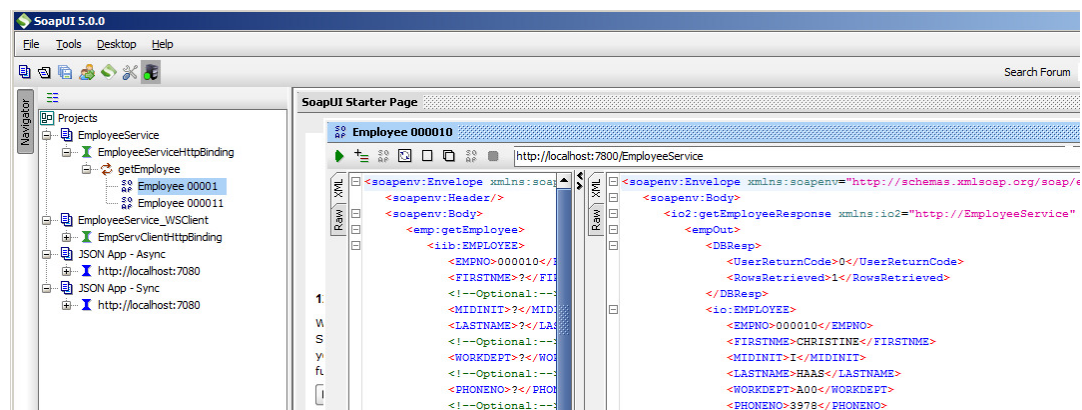
This will overwrite any existing services of the same name in server1.

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

Open SOAPUI, and import the SOAPUI project  
c:\student10\integration\_service\SOAPUI Projects\EmployeeService-soapui-project.xml.

(This may already have been imported into the SOAPUI tool. It will use the port 7800).

Open the test for 000010. 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.

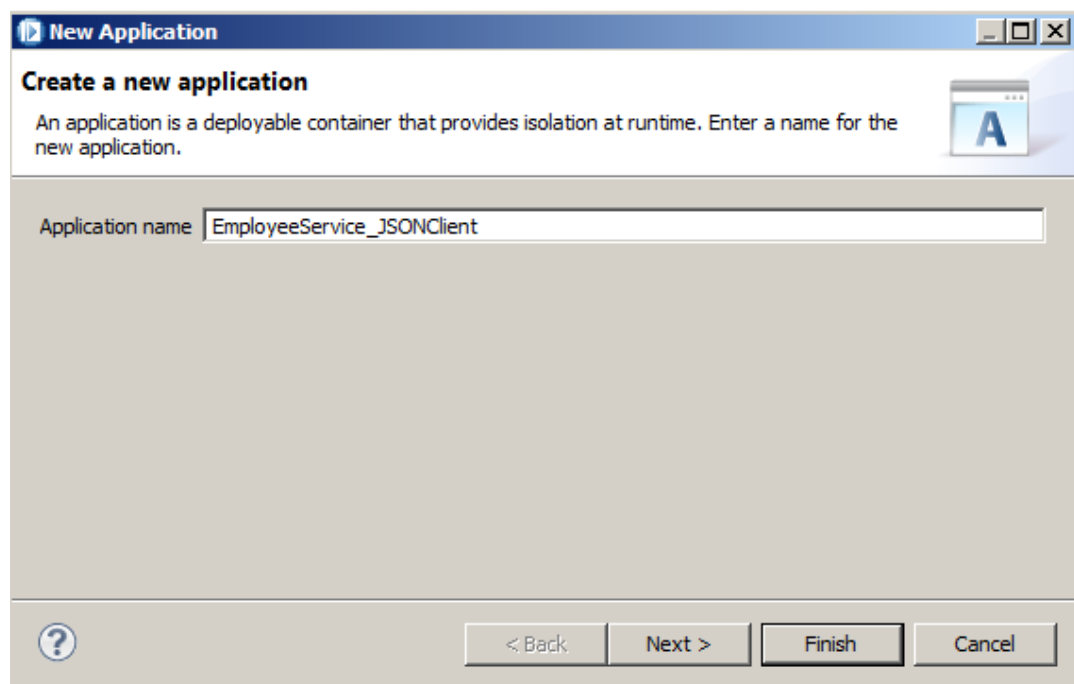
The EmployeeService uses the EMPLOYEE tables of the SAMPLE database, and has implemented the getEmployee operation.

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

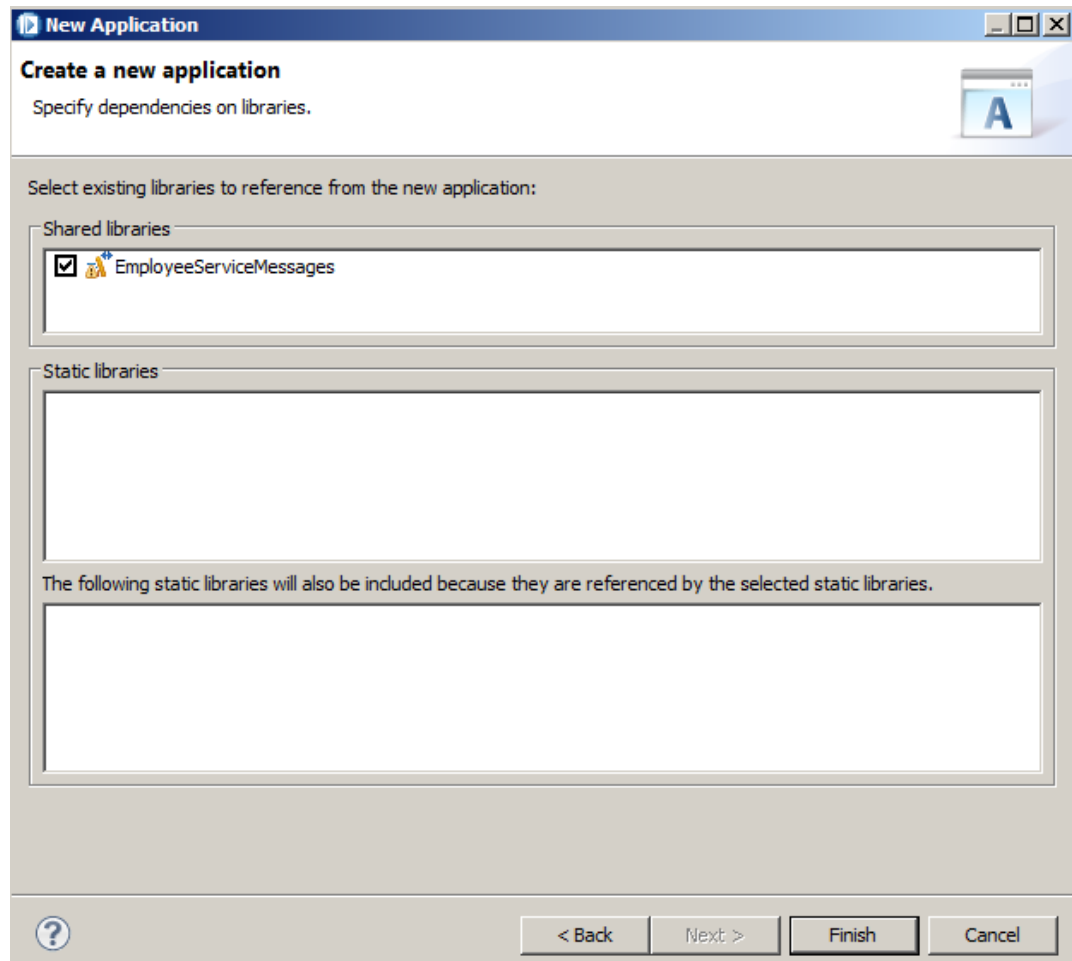
1. In the workspace, create a new Application named EmployeeService\_JSONClient.

Click Next.



2. Select EmployeeServiceMessages as a referenced Shared Library.

Click Finish.



The image shows a 'New Application' dialog box with a blue title bar. The main heading is 'Create a new application' with the instruction 'Specify dependencies on libraries.' Below this, it says 'Select existing libraries to reference from the new application:'. There are two sections: 'Shared libraries' and 'Static libraries'. In the 'Shared libraries' section, 'EmployeeServiceMessages' is selected with a checkmark. The 'Static libraries' section is empty. Below the static libraries section, it says 'The following static libraries will also be included because they are referenced by the selected static libraries.' and shows an empty list. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. A help icon (?) is also present in the bottom left corner.

**New Application**

**Create a new application**  
Specify dependencies on libraries.

Select existing libraries to reference from the new application:

Shared libraries

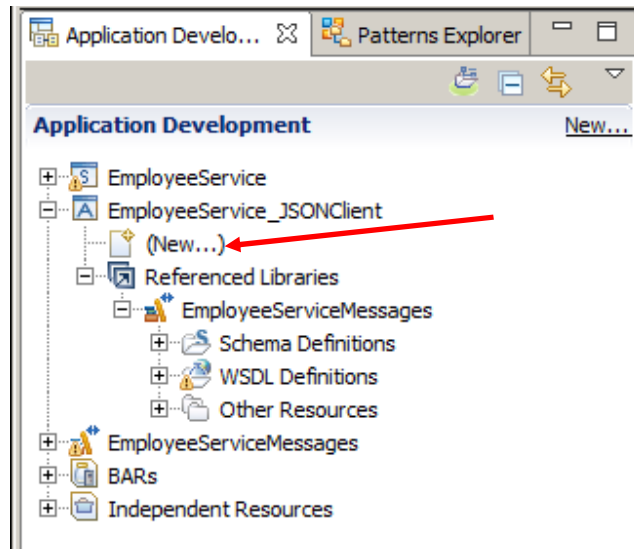
- ☒ EmployeeServiceMessages

Static libraries

The following static libraries will also be included because they are referenced by the selected static libraries.

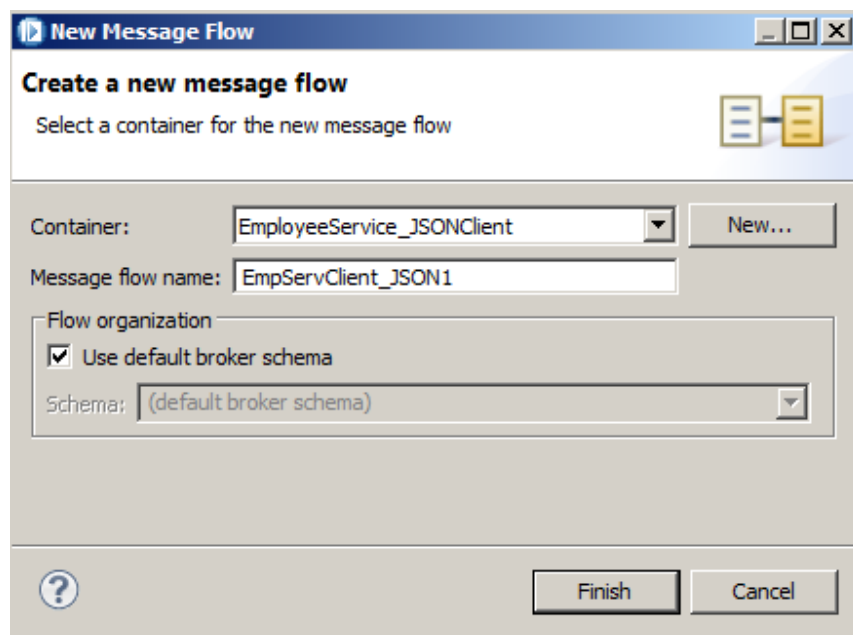
< Back Next > Finish Cancel

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



Name the new flow EmpServClient\_JSON1.

Click Finish.



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

- HTTP Input
- Mapping Node - name it "JSON\_to\_SOAP"
- Another Mapping Node - name it "XML\_to\_JSON"
- HTTP Reply

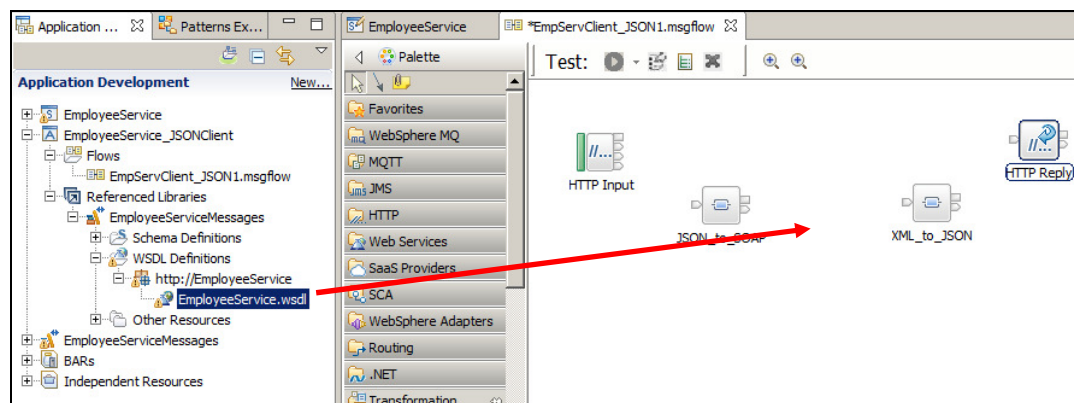
For the HTTP Input node:

- Set the Basic URL Path Suffix property to **/empServClient\_JSON1** (the forward slash is important).
- Set the Input Message Parsing Message Domain to **JSON**.



5. 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, between the two mapping nodes.





6. 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.

**Configure New Web Service Usage**

**Configure web service usage**

Specify the details of how the selected web service will be used in the message flow. Only SOAP HTTP and SOAP JMS bindings are supported.

Web service usage

☐ Expose message flow as web service

☒ Invoke web service from message flow

Web service parameters

Port type: EmployeeService

Binding: EmployeeServiceHttpBinding

Service port: EmployeeServiceHttpPort

Transport: HTTP

Binding operations:

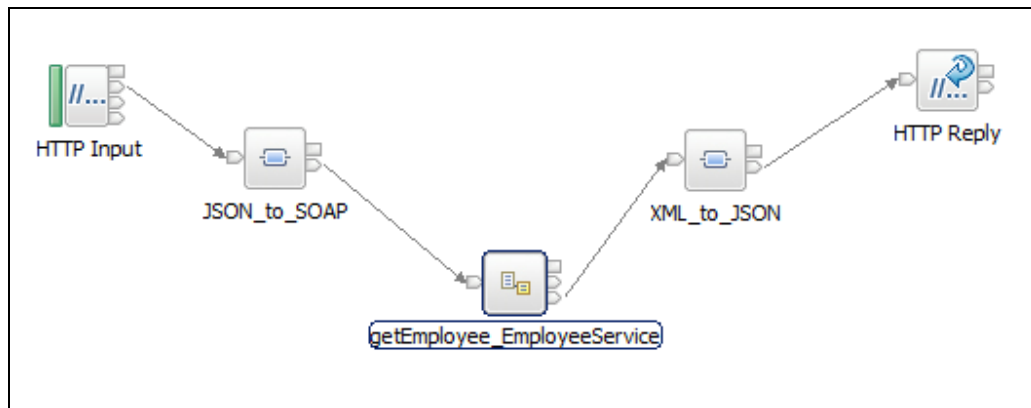
☒ getEmployee(getEmployee)

Select All Deselect All

? < Back Next > Finish Cancel

7. Connect the flow nodes as shown, and save the flow.

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

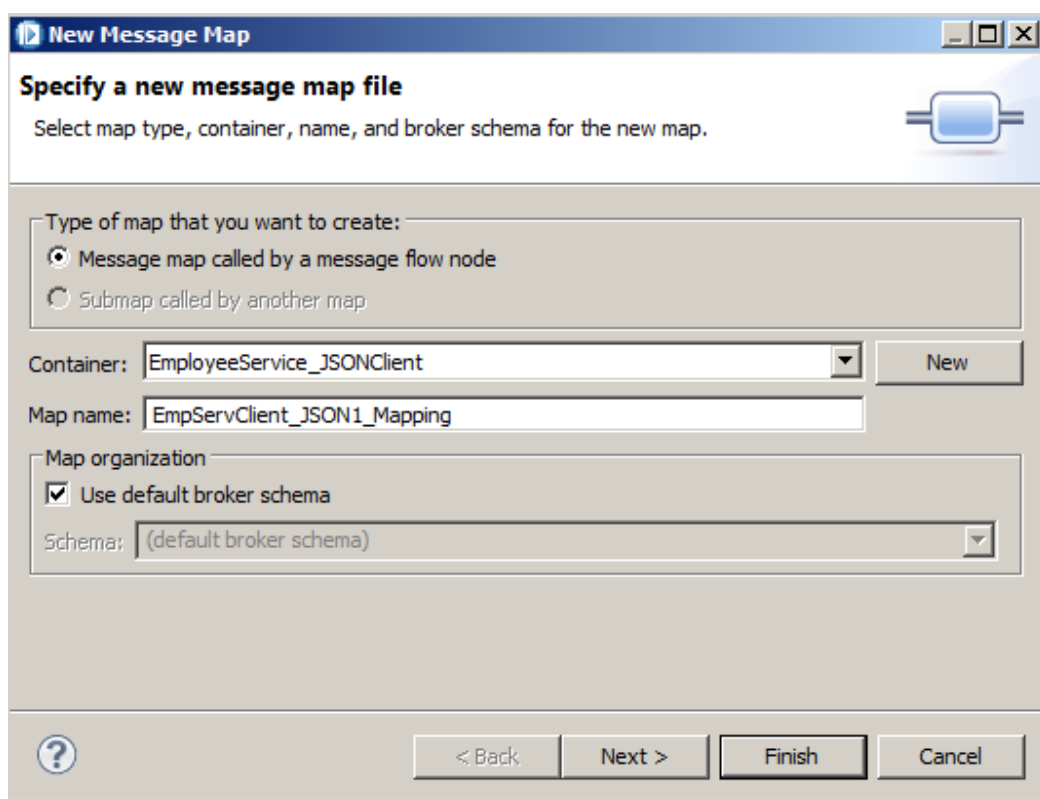


## 2.1 Implement the Mapping Nodes

This section is going to use the Mapping Nodes with the schemaless mapping support provided in IIB V10. The first map 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 SOAP Extract node).

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

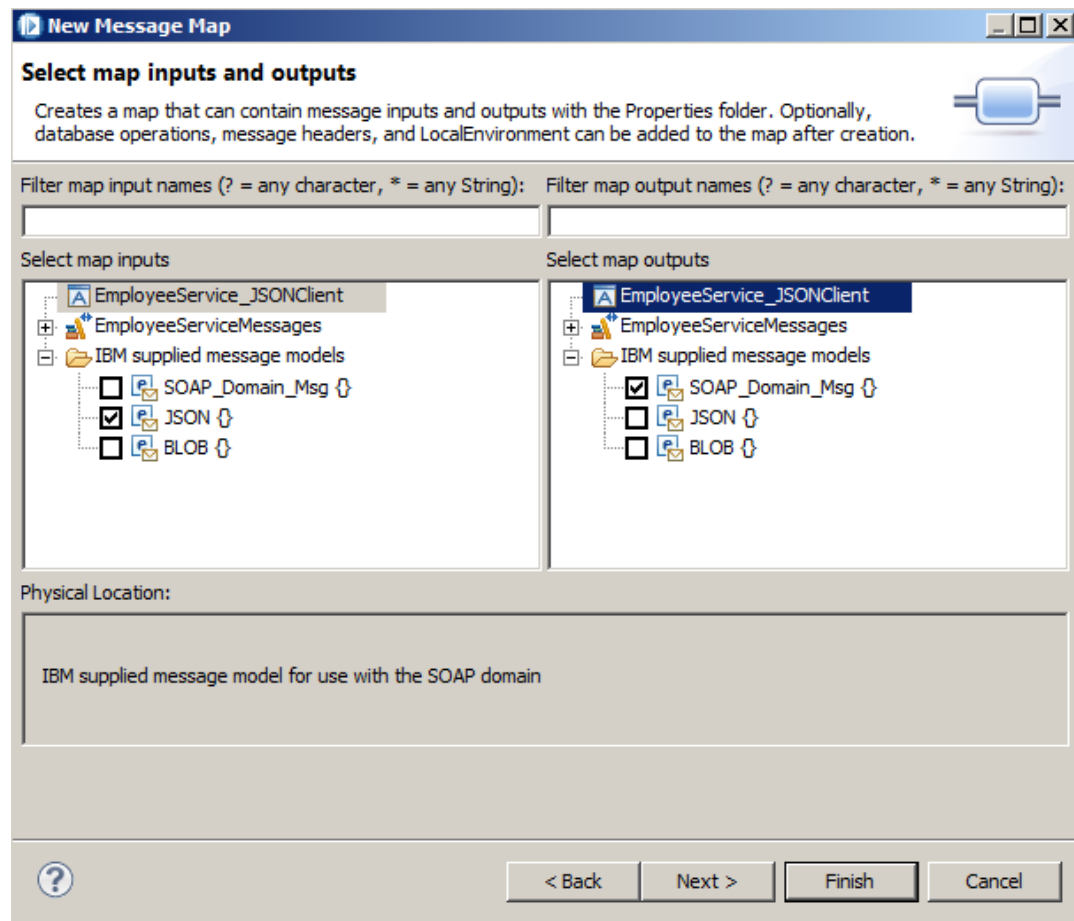


The image shows a 'New Message Map' dialog box with the following fields and options:

- Title:** New Message Map
- Instruction:** Specify a new message map file. Select map type, container, name, and broker schema for the new map.
- Type of map that you want to create:**
  - ☒ Message map called by a message flow node
  - ☐ Submap called by another map
- Container:** EmployeeService\_JSONClient (dropdown menu)
- Map name:** EmpServClient\_JSON1\_Mapping
- Map organization:**
  - ☒ Use default broker schema
  - Schema:** (default broker schema) (dropdown menu)
- Buttons:** < Back, Next >, Finish, Cancel

2. At the next window, select the map inputs and outputs.
  - For the input, expand the IBM supplied message models, and choose JSON.
  - For the output, under the same folder, choose SOAP\_Domain\_Msg.

Click **Next**.



The image shows the 'New Message Map' dialog box in IBM Integration Bus. The title bar says 'New Message Map'. Below the title bar, there's a section titled 'Select map inputs and outputs' with a description: 'Creates a map that can contain message inputs and outputs with the Properties folder. Optionally, database operations, message headers, and LocalEnvironment can be added to the map after creation.' To the right of this text is a small icon of a message map. Below the description are two filter boxes: 'Filter map input names (? = any character, \* = any String):' and 'Filter map output names (? = any character, \* = any String):'. The main area is divided into two panes: 'Select map inputs' and 'Select map outputs'. Both panes show a tree view of the project structure. In the 'Select map inputs' pane, the tree is expanded to 'IBM supplied message models', and 'JSON' is selected with a checkmark. In the 'Select map outputs' pane, the tree is also expanded to 'IBM supplied message models', and 'SOAP\_Domain\_Msg' is selected with a checkmark. Below the panes is a 'Physical Location:' field with the text 'IBM supplied message model for use with the SOAP domain'. At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. A help icon (?) is located to the left of the buttons.

**New Message Map**

**Select map inputs and outputs**

Creates a map that can contain message inputs and outputs with the Properties folder. Optionally, database operations, message headers, and LocalEnvironment can be added to the map after creation.

Filter map input names (? = any character, \* = any String): Filter map output names (? = any character, \* = any String):

Select map inputs Select map outputs

EmployeeService\_JSONClient EmployeeService\_JSONClient

EmployeeServiceMessages EmployeeServiceMessages

IBM supplied message models IBM supplied message models

SOAP\_Domain\_Msg SOAP\_Domain\_Msg

JSON JSON

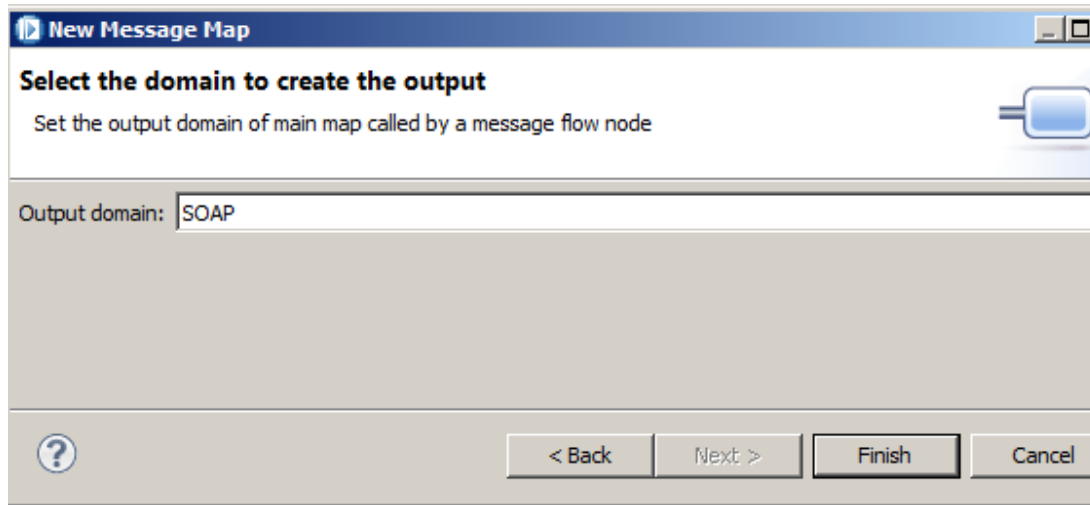
BLOB BLOB

Physical Location:

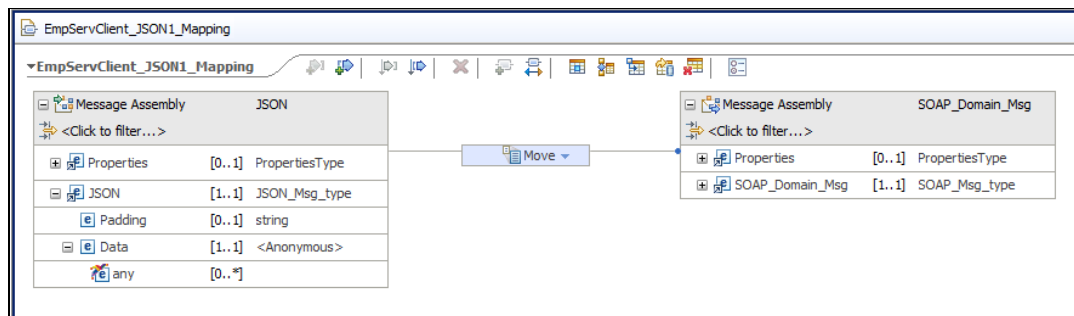
IBM supplied message model for use with the SOAP domain

? < Back Next > Finish Cancel

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



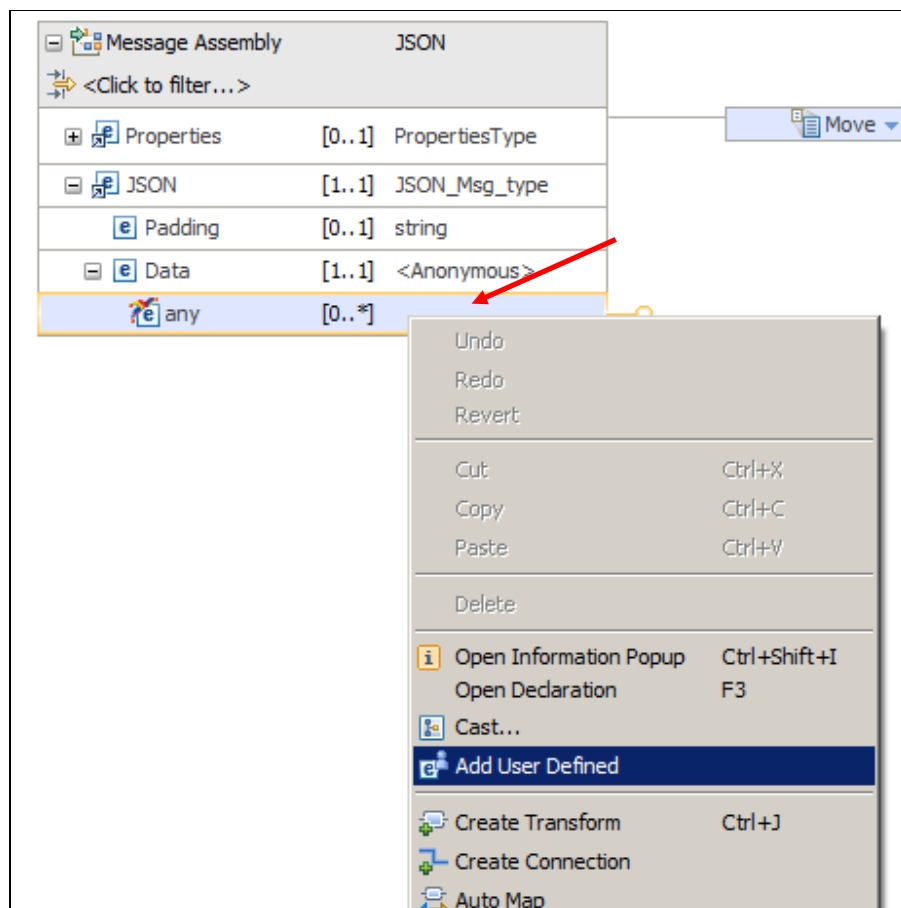
4. The mapping editor will load. Expand the Message Assembly for the input message.



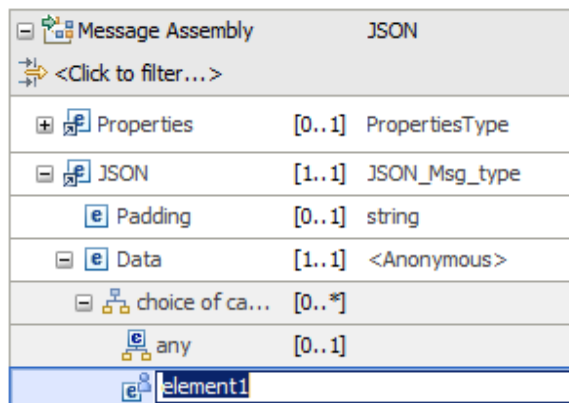
5. You are going to add a user-defined element to the JSON part of this input message assembly.

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.

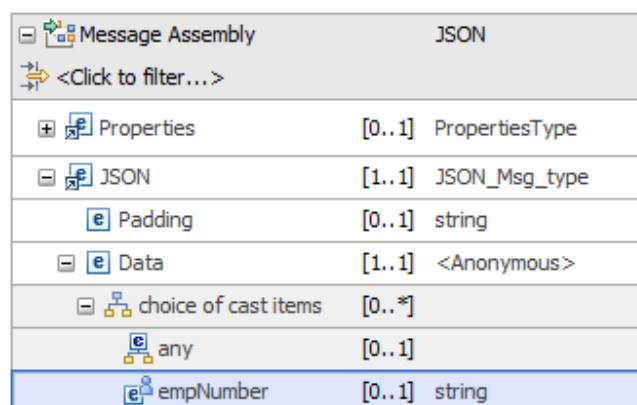


6. The word "element1" is added to the JSON message.



| Message Assembly     |                 | JSON                  |
|----------------------|-----------------|-----------------------|
| <Click to filter...> |                 |                       |
| +                    | Properties      | [0..1] PropertiesType |
| -                    | JSON            | [1..1] JSON_Msg_type  |
|                      | Padding         | [0..1] string         |
| -                    | Data            | [1..1] <Anonymous>    |
|                      | choice of ca... | [0..*]                |
|                      | any             | [0..1]                |
|                      | element1        |                       |

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

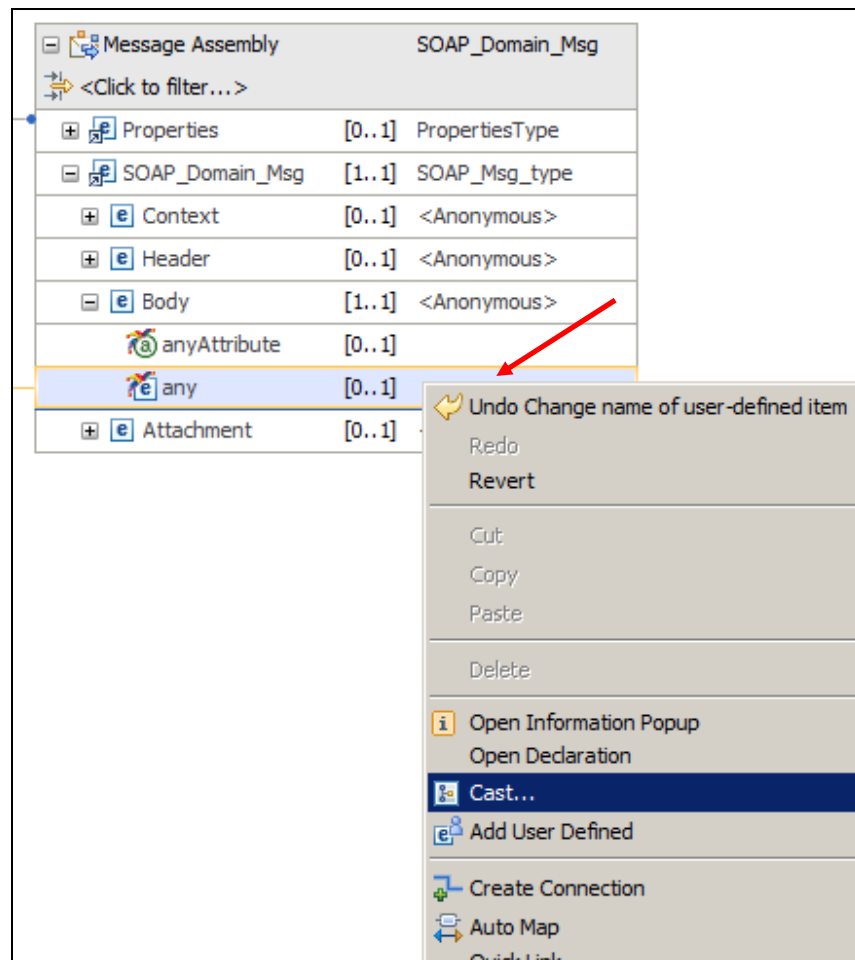


| Message Assembly     |                      | JSON                  |
|----------------------|----------------------|-----------------------|
| <Click to filter...> |                      |                       |
| +                    | Properties           | [0..1] PropertiesType |
| -                    | JSON                 | [1..1] JSON_Msg_type  |
|                      | Padding              | [0..1] string         |
| -                    | Data                 | [1..1] <Anonymous>    |
|                      | choice of cast items | [0..*]                |
|                      | any                  | [0..1]                |
|                      | empNumber            | [0..1] 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 EmployeeService interface.

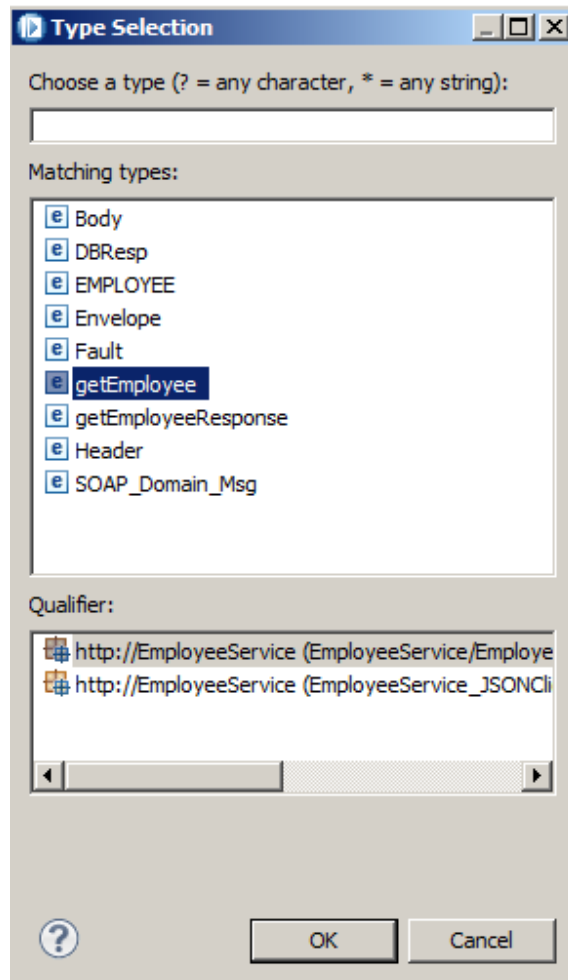
In the output message assembly, expand the SOAP\_Domain\_Msg, then Body.

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



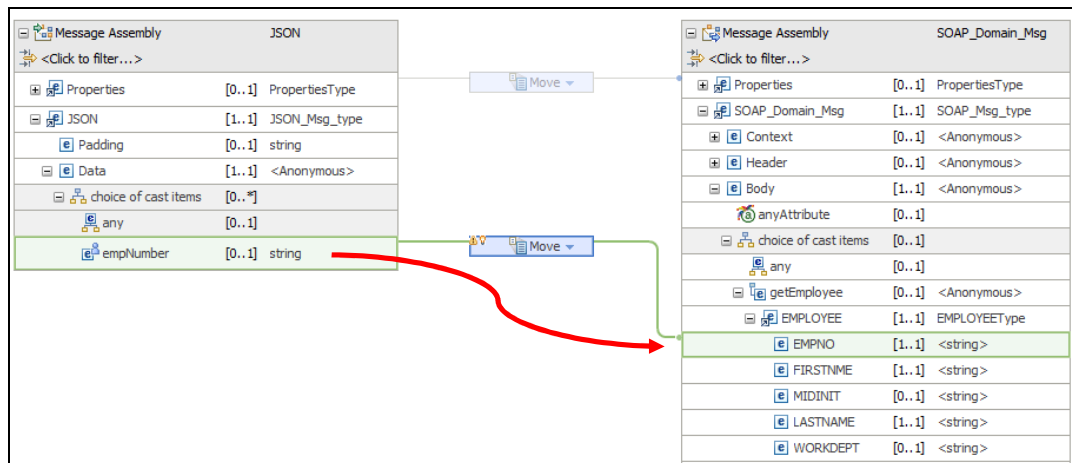


9. From the Type Selection window, highlight `getEmployee`, and click OK.  
(`getEmployee` is the name of the input message for `EmployeeService`).



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

This will create a Move transform.



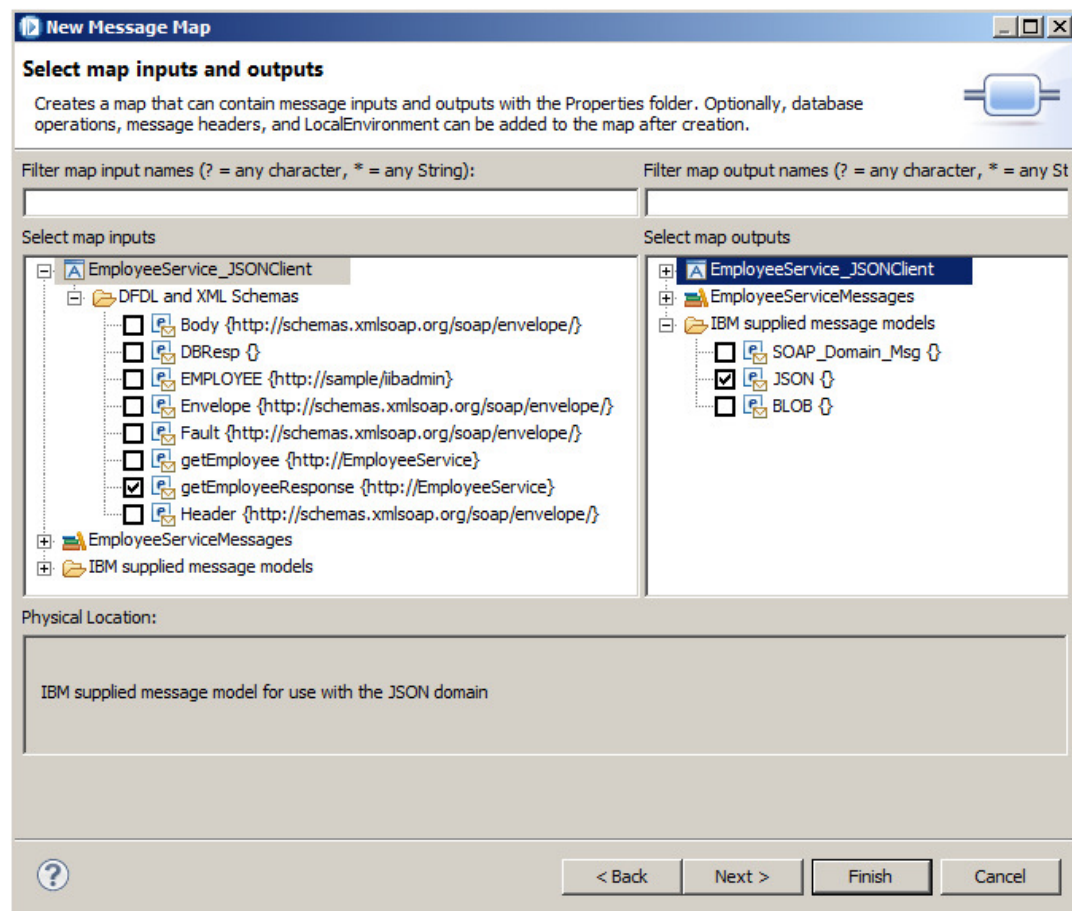
Save and close this mapping node.

## 11. Configure the second mapping node (XML\_to\_JSON).

Double-click the node, and at the input and output selection, make the following choices:

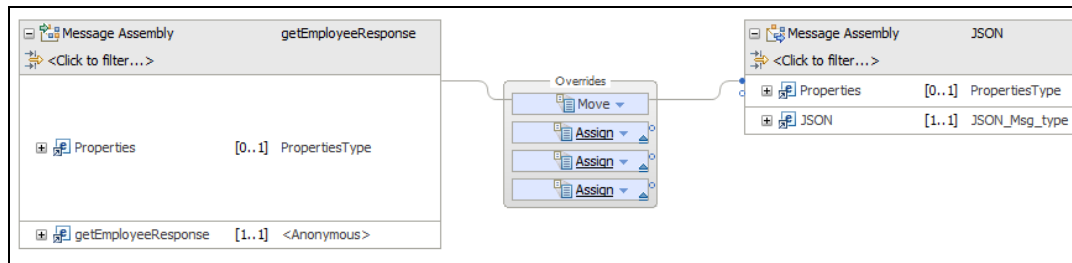
- For the input message, because the generated subflow which invokes the web service has included a SOAP Extract node, you do not need to specify a SOAP\_Domain message. Instead, expand EmployeeService\_JSONClient/DFDL and XML Schemas, and select the getEmployeeResponse message.
- For the output message, expand IBM supplied message models, and select JSON.

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



12. 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.



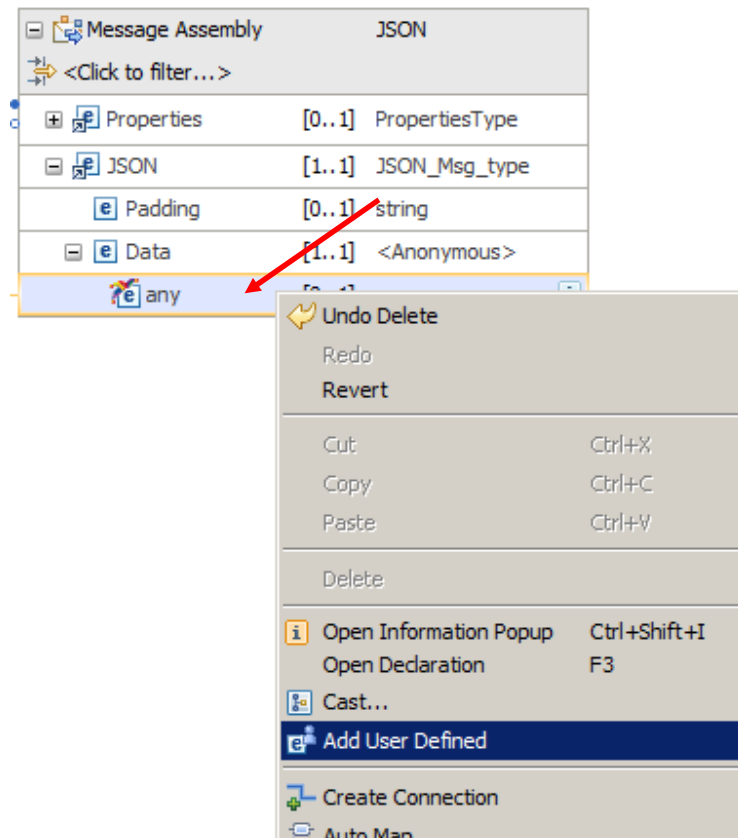
13. Before creating the map transformations, you are going to add a number of elements to the output message, using the JSON format, using the schemaless mapping tools in IIB v10.

You will add a number of elements, so that the resulting output message has the following structure. Note that we have included the elements about the result of the SQL function, and then various elements for the user data.

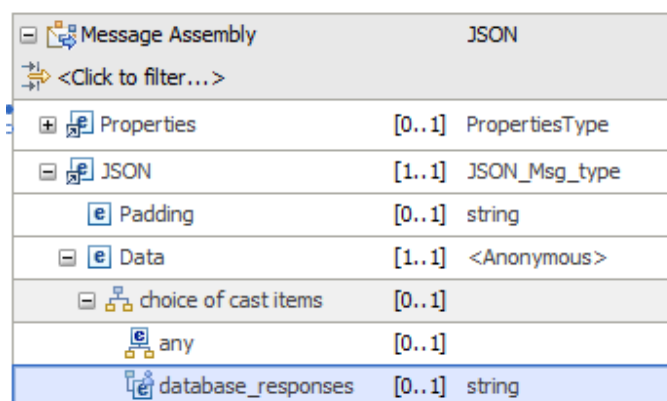
When complete, the output message assembly will look like this:

|                      |        |             |
|----------------------|--------|-------------|
| Data                 | [1..1] | <Anonymous> |
| choice of cast items | [0..*] |             |
| any                  | [0..1] |             |
| database_responses   | [0..1] | <Anonymous> |
| user_return_code     | [1..1] | int         |
| rows_retrieved       | [1..1] | int         |
| rows_added           | [1..1] | int         |
| rows_updated         | [1..1] | int         |
| rows_deleted         | [1..1] | int         |
| employee_details     | [0..1] | <Anonymous> |
| employee_number      | [1..1] | string      |
| name                 | [1..1] | string      |
| department           | [1..1] | string      |
| start_date           | [1..1] | date        |
| payments             | [1..1] | <Anonymous> |
| current_date         | [1..1] | date        |
| salary               | [1..1] | decimal     |
| bonus                | [1..1] | decimal     |
| commission           | [1..1] | decimal     |

14. To create this, first, add a JSON element that will contain the database response information (eg. the user return code from EmployeeService, and the number of rows retrieved). Right-click on the "any" element, and select Add User Defined from the context menu.



15. Overtyp the name of the new element, naming it "database\_responses". Leave the element type unchanged.



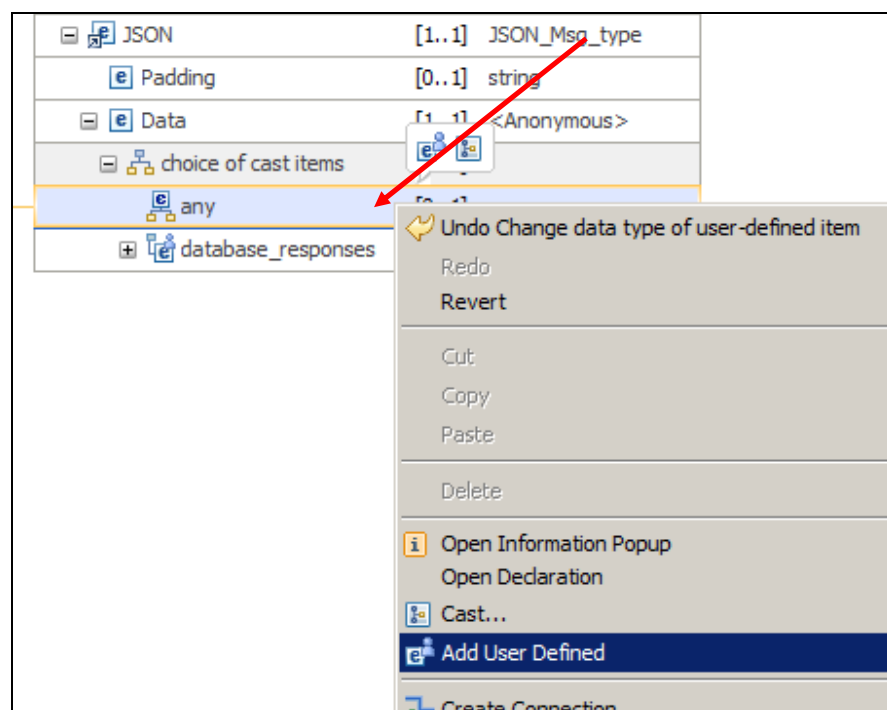
16. Now add some elements as child elements under `database_responses`. Do this by right-clicking `database_responses` for each new element, and select **Add Child Element**.

Add the following elements, as shown. Set the type of all these elements to "int".

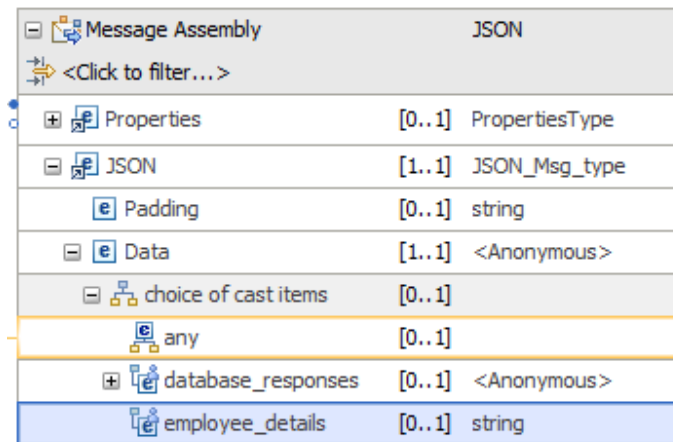
- `user_return_code`
- `rows_retrieved`
- `rows_added`
- `rows_updated`
- `rows_deleted`

|                              |        |             |
|------------------------------|--------|-------------|
| [-] [e] Data                 | [1..1] | <Anonymous> |
| [-] [e] choice of cast items | [0..1] |             |
| [e] any                      | [0..1] |             |
| [-] [e] database_responses   | [0..1] | <Anonymous> |
| [e] user_return_code         | [1..1] | int         |
| [e] rows_retrieved           | [1..1] | int         |
| [e] rows_added               | [1..1] | int         |
| [e] rows_updated             | [1..1] | int         |
| [e] rows_deleted             | [1..1] | int         |

17. Now, add the elements required for the user data. Right-click the "any" element. Select **"Add User Defined"** from the context menu.

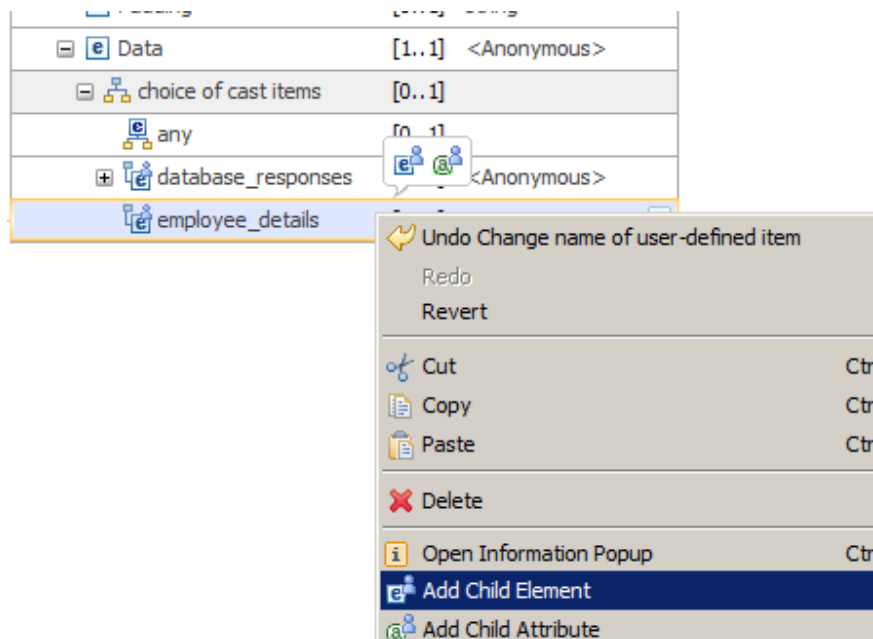


18. Rename the new element to `employee_details`. Leave the element type as string.









| Message Assembly     |                      | JSON                  |
|----------------------|----------------------|-----------------------|
| <Click to filter...> |                      |                       |
| +                    | Properties           | [0..1] PropertiesType |
| -                    | JSON                 | [1..1] JSON_Msg_type  |
|                      | Padding              | [0..1] string         |
| -                    | Data                 | [1..1] <Anonymous>    |
| -                    | choice of cast items | [0..1]                |
|                      | any                  | [0..1]                |
| +                    | database_responses   | [0..1] <Anonymous>    |
|                      | employee_details     | [0..1] string         |

19. Now add elements under the `employee_details` element. Right-click `employee_details`, and select "Add Child Element".



20. Rename the new element "employee\_number". Note that the type of "employee\_details" has changed to <Anonymous>.















Leave the type of "employee\_number" as string.

|  |        |             |
|--|--------|-------------|
|  any  | [0..1] |             |
|   database_responses | [0..1] | <Anonymous> |
|   employee_details   | [0..1] | <Anonymous> |
|  employee_number  | [1..1] | string      |

21. Add four more elements in the same way, by right-clicking the "employee\_details" element (four times). Make sure these elements are all Child elements of employee\_details.

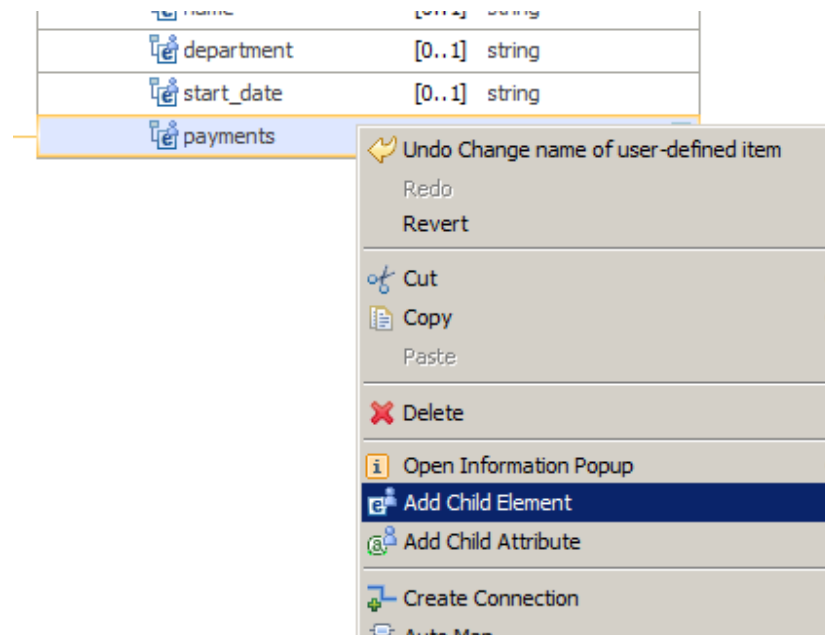
- name (type string)
- department (type=string)
- start\_date (type=date)
- payments (type=string)

At this point, the output message will look like this:

|  |        |             |
|--|--------|-------------|
|   Data                 | [1..1] | <Anonymous> |
|   choice of cast items | [0..1] |             |
|  any  | [0..1] |             |
|   database_responses   | [0..1] | <Anonymous> |
|   employee_details     | [0..1] | <Anonymous> |
|  employee_number  | [1..1] | string      |
|  name   | [1..1] | string      |
|  department   | [1..1] | string      |
|  start_date   | [1..1] | date        |
|  payments   | [1..1] | string      |



22. Now add some child elements under "payments". Right-click "payments", and select Add Child Element.

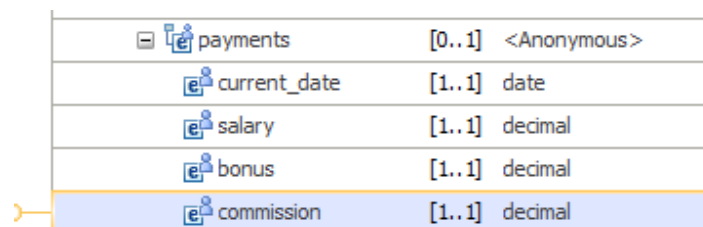


23. Name the new element "current\_date", and set its type to "date".

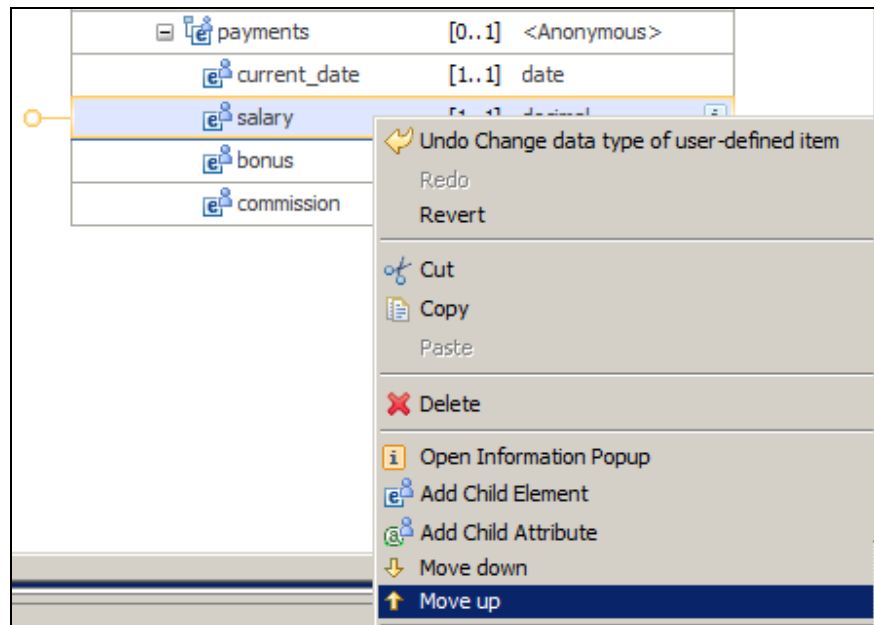


24. Add three further child elements under payments, all of type "decimal".

- salary
- bonus
- commission

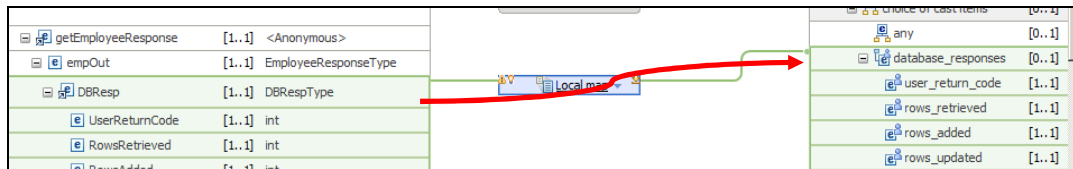


25. If you want to edit the name and position of the new elements, you can change the definitions you have just made. Clicking on the element name allows you to change its name. Right-clicking the elements gives you tools to move the elements up or down in the output message.



## 2.1.1 Create the Mappings

1. First, map DBResp to database\_responses.



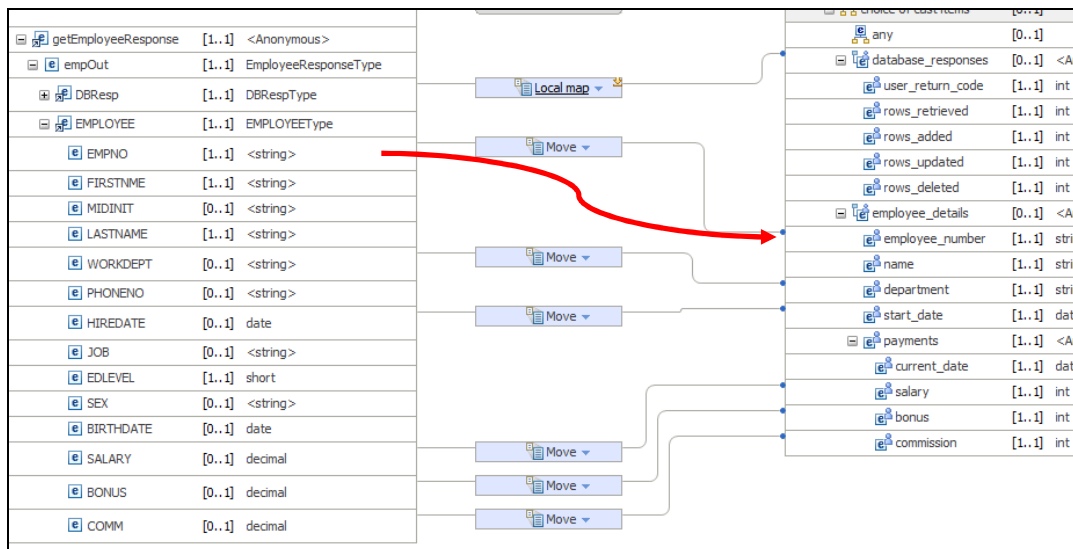
This will initially create a Local map.

Click "Local map", and use the Auto Map function to map all the elements in DBResp (use the auto-map icon on the icon row in the editor).

When complete, return to the highest level of the map.

2. Map the following elements, using a simple Move (drag input element to output element).
  - EMPNO --> employee\_number
  - WORKDEPT --> department
  - HIREDATE --> start\_date
  - SALARY --> salary
  - BONUS --> bonus
  - COMMISSION --> commission

The map will now look like this:



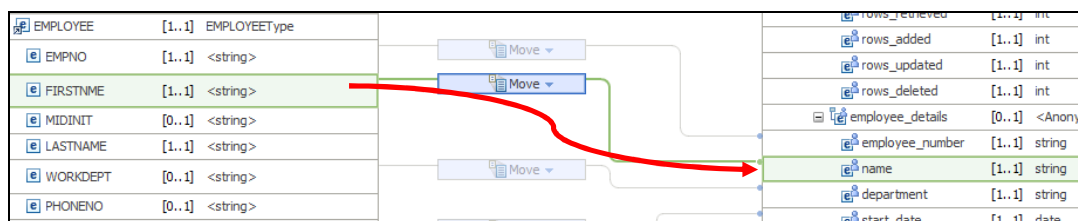
## 2.1.2 Optional extra mapping (Concatenate, Custom XPath, Retrieve flow details)

Although not required for the scenario to work, the following sections show some further examples of the mapping node, using the Concatenate and Custom XPath function.

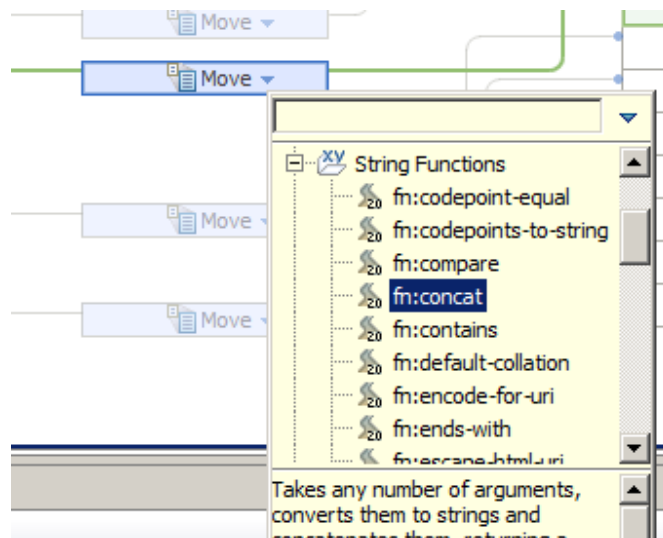
You may proceed direct to Complete the Message Flow on page 33.

1. **Concatenate function** - you will take the three name elements, and concatenate them, with embedded spaces, to create the output element "name".

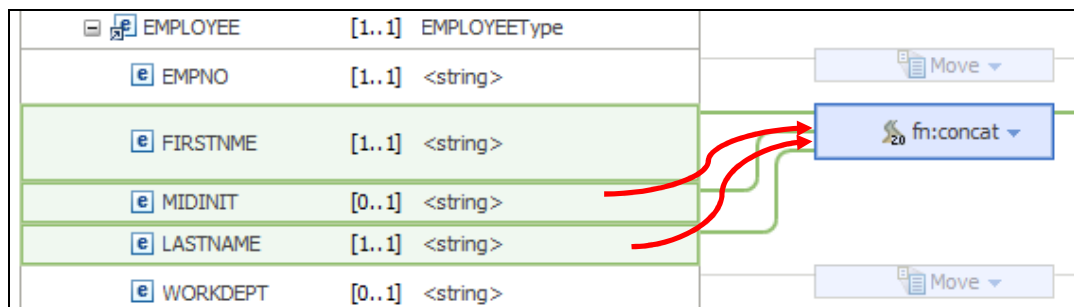
Drag the input element FIRSTNME to the output element name.



2. This will generate a Move transform. Using the drop-down arrow, change this to "fn:concat" (in the string Functions folder).



3. Connect the elements MIDINIT and LASTNAME to the concat function box.



Highlight the Concat function box, and look at the Properties pane. You will see that three elements have been added (\$FIRSTNAME, space, and \$LASTNAME). \$MIDINIT is not shown at this point, because it is not a mandatory element in the input message.

The screenshot shows the 'Properties' pane for the 'Transform - concat' function. The 'General' tab is selected. The 'Parameters' section contains a table with three rows:

| Name    | Type      | Value        |
|---------|-----------|--------------|
| string1 | xs:string | \$FIRSTNAME2 |
| string2 | xs:string | "            |
| string3 | xs:string | \$LASTNAME1  |

To the right of the table are buttons: 'Add', 'Edit...', and 'Remove'.

4. Use the Add button to add two further parameters, string4 and string5.

The screenshot shows the 'Properties' pane for the 'Transform - concat' function. The 'Parameters' section now contains a table with five rows:

| Name    | Type      | Value        |
|---------|-----------|--------------|
| string1 | xs:string | \$FIRSTNAME2 |
| string2 | xs:string | "            |
| string3 | xs:string | \$LASTNAME1  |
| string4 | xs:string | "            |
| string5 | xs:string | "            |

The 'Add' button is highlighted with a red rectangle.

5. Click the value of string3. Its initial value will be LASTNAME. Change this to MIDINIT.

Parameters:

| Name    | Type      | Value       |  |
|---------|-----------|-------------|--|
| string1 | xs:string | \$FIRSTNAME |  |
| string2 | xs:string | "           |  |
| string3 | xs:string | \$MIDINIT   |  |
| string4 | xs:string | "           |  |
| string5 | xs:string | "           |  |
|         |           |             |  |
|         |           |             |  |

Add  
Edit...  
Remove

Change the value of string5 to LASTNAME.

Parameters:

| Name    | Type      | Value       |  |
|---------|-----------|-------------|--|
| string1 | xs:string | \$FIRSTNAME |  |
| string2 | xs:string | "           |  |
| string3 | xs:string | \$MIDINIT   |  |
| string4 | xs:string | "           |  |
| string5 | xs:string | \$LASTNAME  |  |
|         |           |             |  |
|         |           |             |  |

Add  
Edit...  
Remove

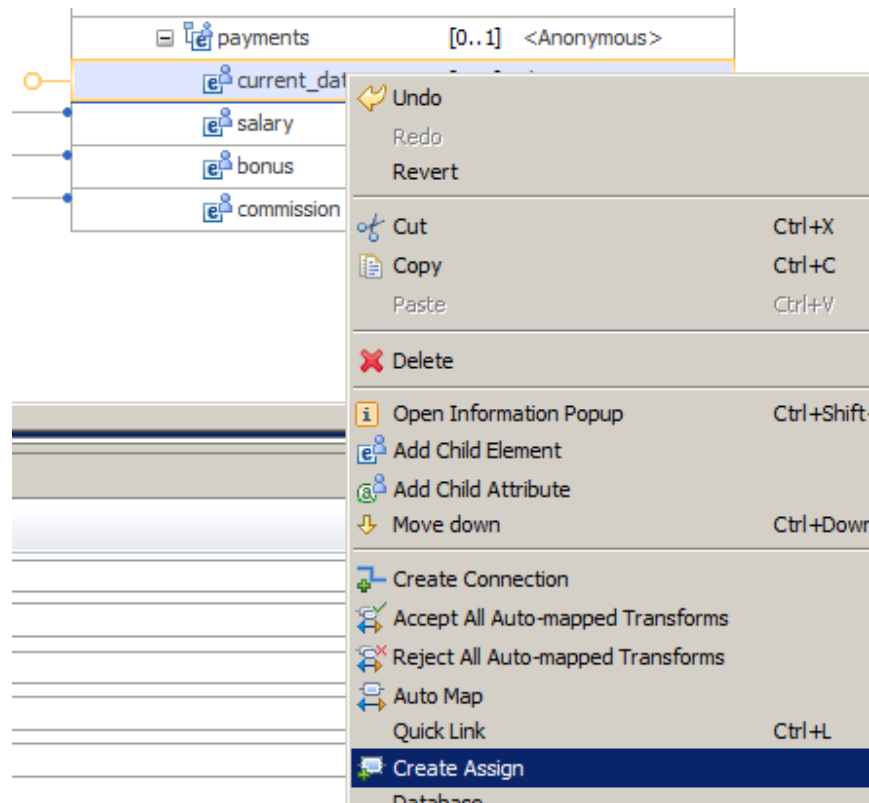
6. Change the values of string2 and string4 to add a space between the two apostrophes.

(Highlight each line and click Edit).

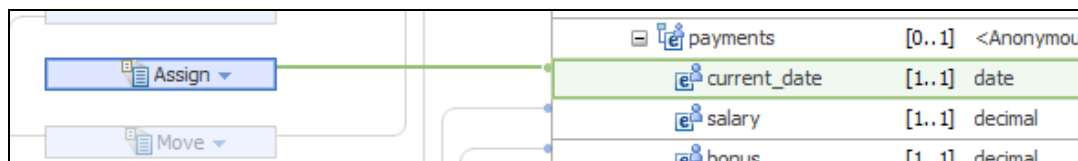
### 2.1.3 Custom XPath

Now use XPATH to set a value for the current\_date element.

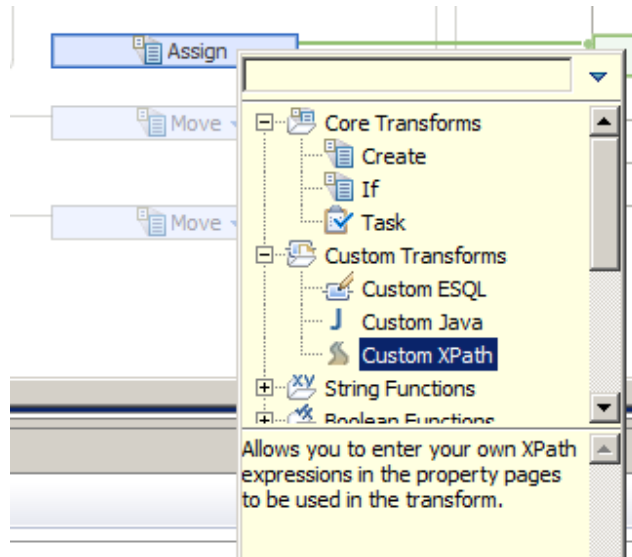
1. Under payments, right-click current\_date, and select Create Assign.



An Assign function will be created.



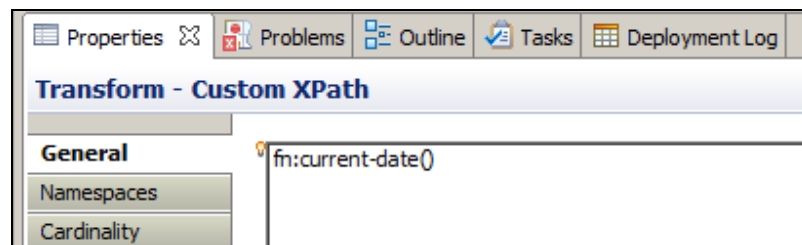
- Using the drop-down arrow on the Assign transform, change this to a Custom XPath function (in the Custom Transforms folder).



- In the Properties of the Custom XPath transform, set the General property to

**fn:current-date()**

(Hint: type fn:cu, then use Ctrl-space to provide the list of possible completions).



- The second map is now complete, so save it and close the map editor.

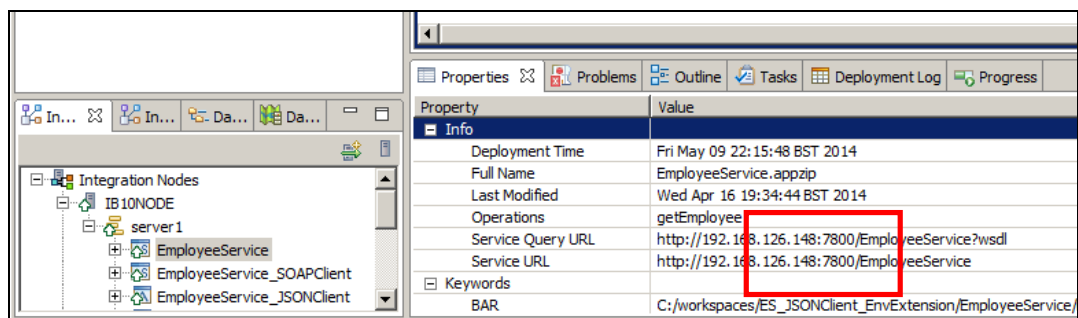


## 2.2 Complete the Message Flow

1. Two final changes are necessary. First, you must make sure that the new application is configured to use the correct port number of the EmployeeService.

In the Integration Nodes pane, expand IB10NODE and server1.

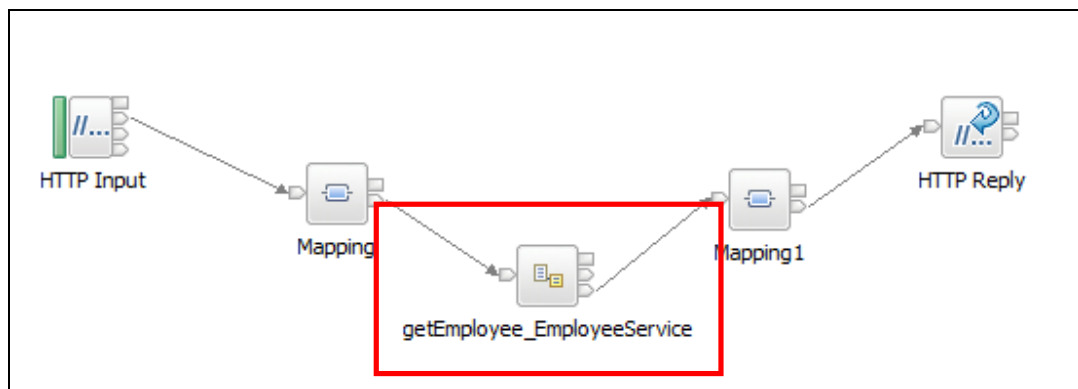
Highlight EmployeeService. In the properties of the deployed service, you will see the Service URL that is active for this service. Note and record the port number that is in use. In the example below, this is 7800, but may be different on your system.



The screenshot shows the IBM Integration Bus Properties window for the EmployeeService. The left pane shows the Integration Nodes tree with IB10NODE expanded, and server1 expanded, showing EmployeeService selected. The right pane shows the Properties tab with the Info section expanded. The Service URL is highlighted with a red box, showing the port number 7800.

| Property          | Value   |
|-------------------|---|
| Deployment Time   | Fri May 09 22:15:48 BST 2014                              |
| Full Name         | EmployeeService.appzip                                    |
| Last Modified     | Wed Apr 16 19:34:44 BST 2014                              |
| Operations        | getEmployee   |
| Service Query URL | http://192.168.126.148:7800/EmployeeService?wsdl          |
| Service URL       | http://192.168.126.148:7800/EmployeeService               |
| Keywords          |   |
| BAR               | C:/workspaces/ES_JSONClient_EnvExtension/EmployeeService/ |

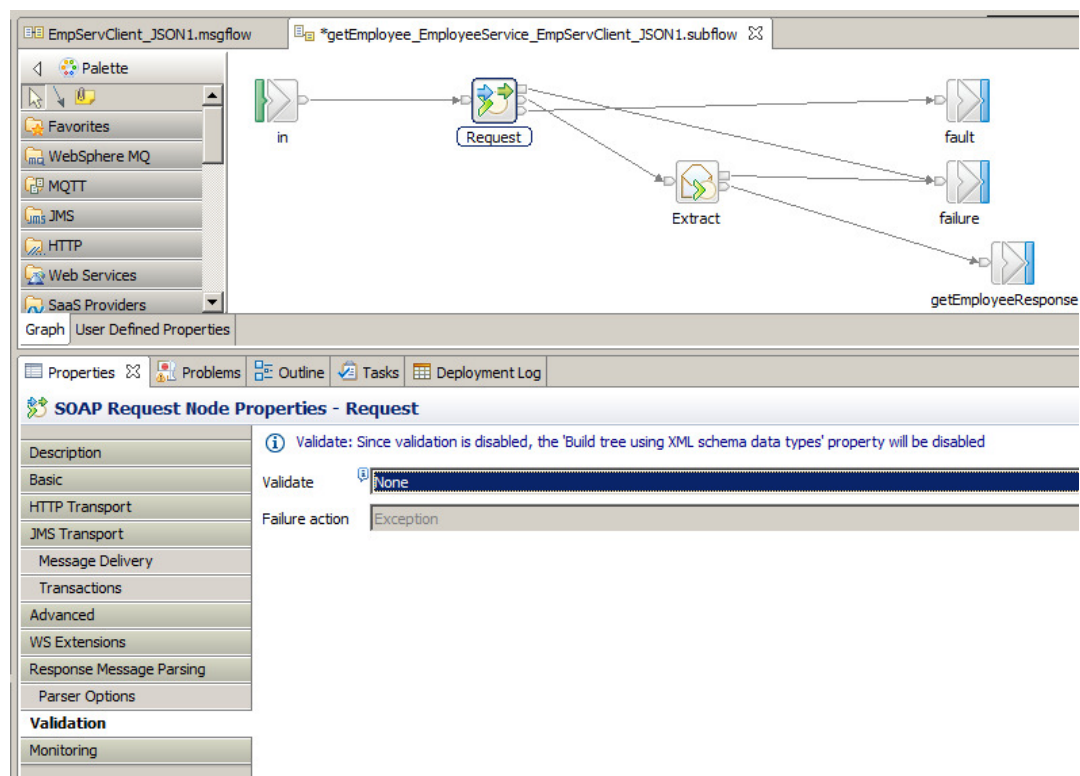
2. In the primary message flow, double-click the generated subflow getEmployee\_EmployeeService.



3. Because the original schema derived from the EMPLOYEE table contains certain column constraints, these have been reflected in the Validation requirements for the EmployeeService web service.

For ease of simple testing, the generated SOAP Request node should be configured to remove Validation.

Highlight the SOAP Request node, and in the node properties select the Validation tab. Set the Validate value to None.



4. Finally, again 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).

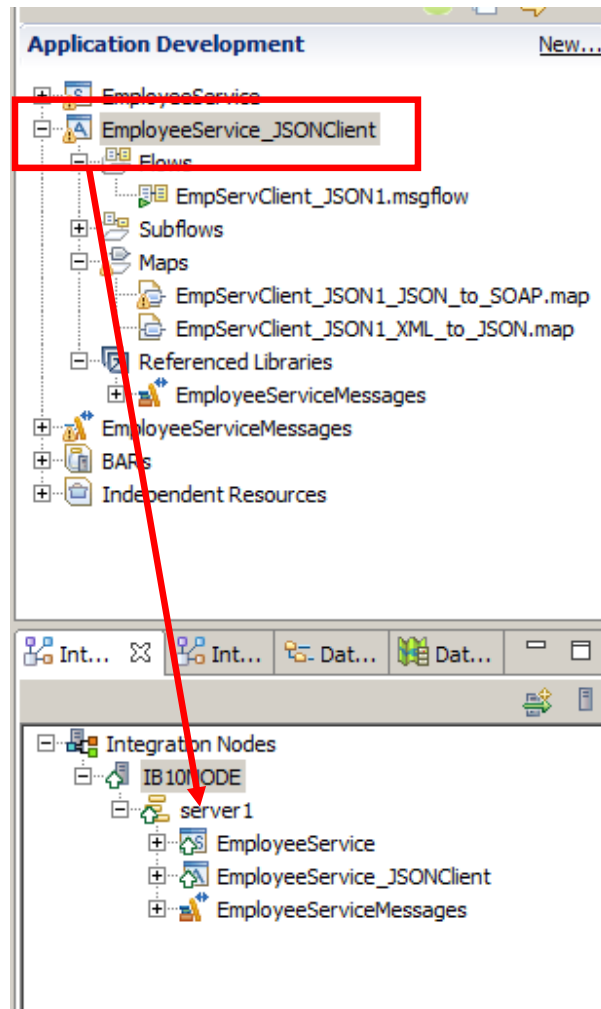
Save and close the subflow.

The screenshot displays the IBM Integration Bus V10 Workshop interface. The top pane shows a subflow diagram titled 'getEmployee\_EmployeeService\_Flow1.subflow'. The diagram starts with an 'in' connector, followed by a 'Request' node. From the 'Request' node, the flow branches into two paths: one leading to an 'Extract' node and then to a 'getEmployeeResponse' connector, and another leading directly to a 'fault' connector. There are also 'failure' and 'getEmployeeResponse' connectors shown as part of the flow logic.

The bottom pane shows the 'SOAP Request Node Properties - Request' dialog. The 'Web service URL\*' field is highlighted with a red rectangle and contains the text 'http://localhost:7800/EmployeeService'. Below this field, a hint text reads 'e.g. http://server/path/to/service'. Other fields in the dialog include 'Request timeout (in seconds)' set to 120, 'HTTP(S) proxy location' set to '<enter your proxy server (if any)>', 'Protocol (if using SSL)' set to TLS, 'Allowed SSL ciphers (if using SSL)' set to '<enter any specific SSL Ciphers you wish to use>', and 'Use compression' set to none.

5. Deploy the application to the server.

In the navigator, drag the EmployeeService\_JSONClient to IB10NODE/server1.



## 3. Test the EmpServClient\_JSON1 Application

### 3.1 Test using SOAPUI

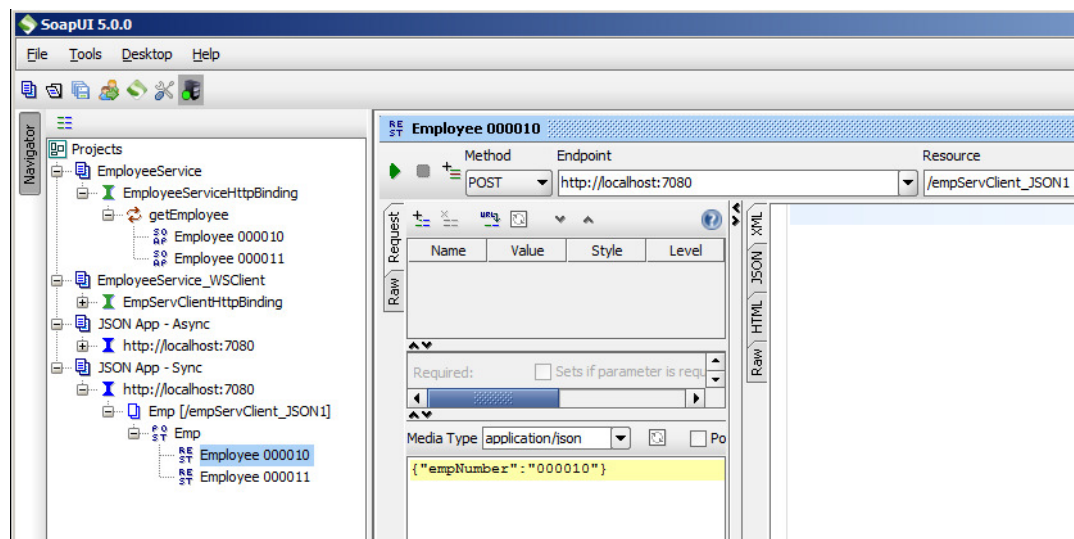
1. Open SOAPUI (you may already have this open from a previous lab).

Expand the project "JSON App - Sync", and open the request "Employee 000010".

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

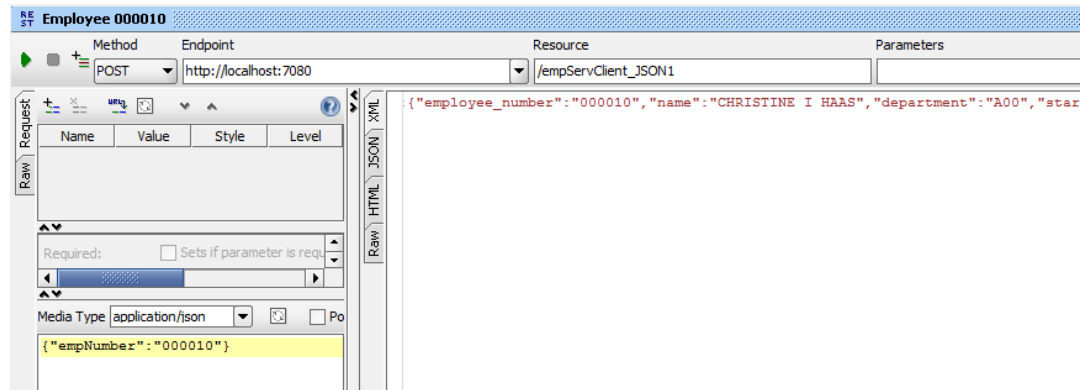
```
{"empNumber":"000010"}
```

Note that the endpoint url is localhost:7080. 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.

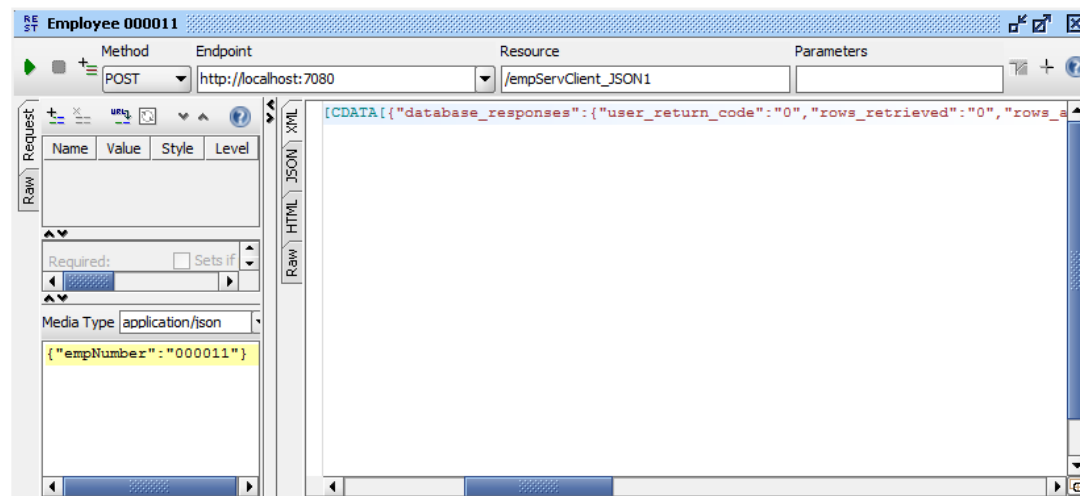


(For info, this SOAPUI project was created by specifying "New REST Project" in the SOAPUI dialogue).

2. The SOAPUI test runs, and the returned data will be shown in JSON format in the response pane. You will need to use the slide bar to view the full response data.



3. Similarly, use the "Employee 000011" request to show the response when the row is not present in the Employee table.

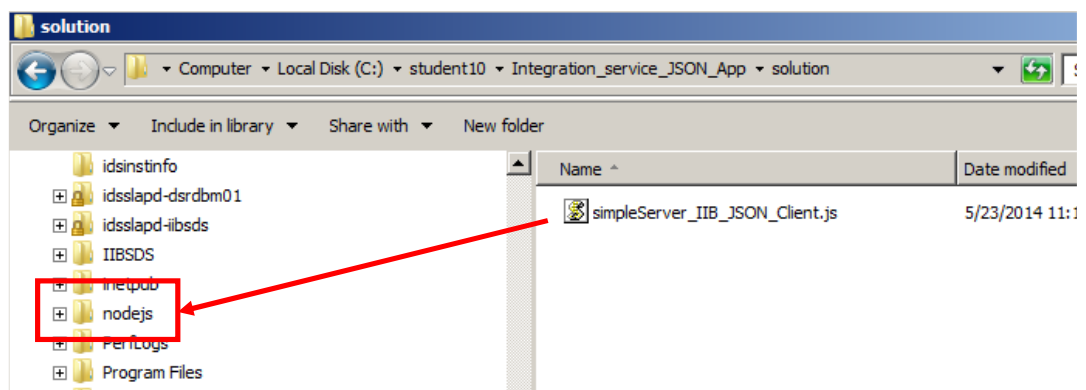


### 3.2 Test using Node.js Web Browser application (Optional)

In this part of the lab, you can test the developed application, using an example Web Browser application. This will present a more 'real' example of the IBM Integration Bus capabilities. You will complete the following steps:

- Use **Node.js** to start an HTTP server and run a Web application;
  - In a Web Browser window you view the Web application and retrieve 'Employee data' invoking the '**EmpServClient\_JSON1**' IBM Integration Bus application
1. In Windows Explorer, navigate to the folder  
C:\student10 \ Integration\_service\_JSON\_App \ solution.

**Copy** the file '**simpleServer\_IIB\_JSON\_Client.js**' and paste it in the **c:\nodejs** folder.

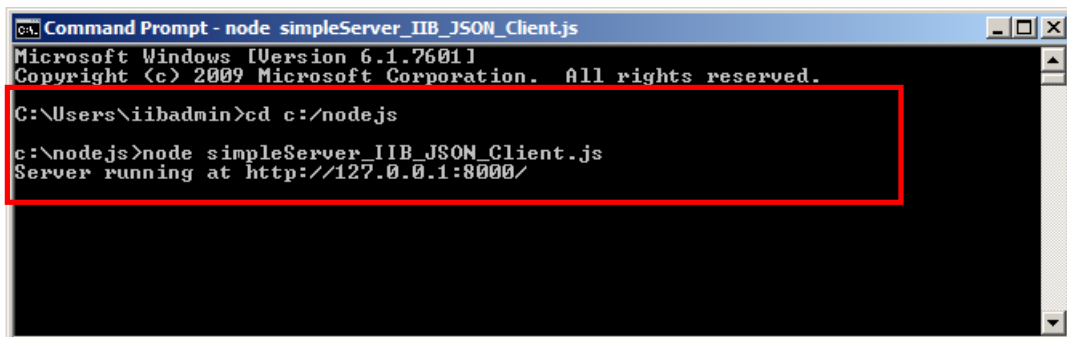


2. In a Windows Command Prompt, do the following:

- Change directory to c:\nodejs  
:
- Enter the command:

```
node simpleServer_IIB_JSON_Client.js
```

- View the message on the bottom of the Command Console:  
Server running at <http://127.0.0.1:8000/>



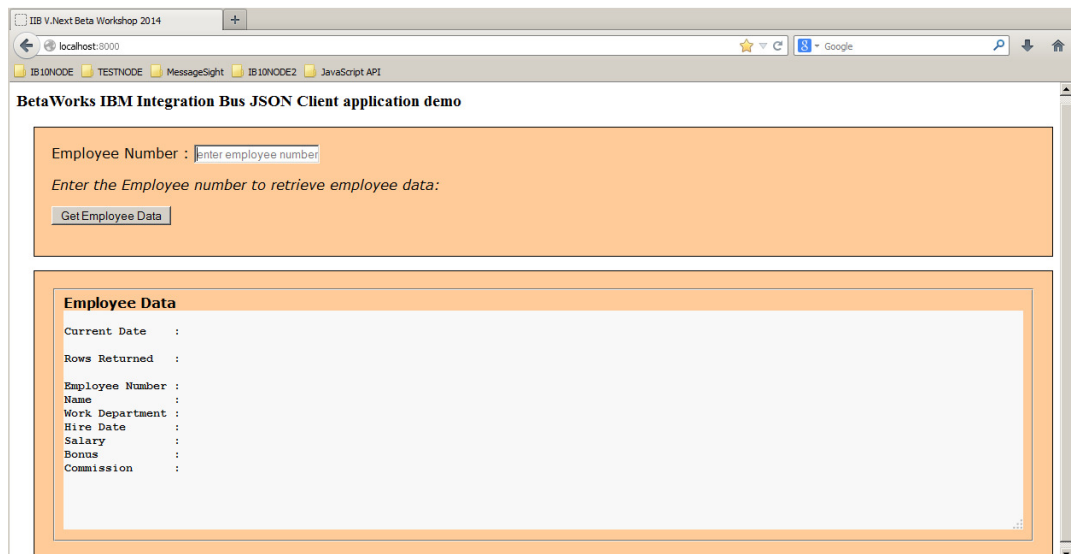
```
Command Prompt - node simpleServer_IIB_JSON_Client.js
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\iibadmin>cd c:/nodejs
c:\nodejs>node simpleServer_IIB_JSON_Client.js
Server running at http://127.0.0.1:8000/
```

- **Important: Please do not close the Command Prompt, as this will stop the server. You can however minimize the window.**

3. In a Web Browser, navigate to <http://127.0.0.1:8000> (or <http://localhost:8000> )

This will open the Web page of the ‘EmpServClient\_JSON1’ Web application:





4. Enter an employee number 000010 in the field as shown and press 'Get Employee Data'

BetaWorks IBM Integration Bus JSON Client application demo

Employee Number : 000010

Enter the Employee number to retrieve employee data:

Get Employee Data

**Employee Data**

Current Date :  
Rows Returned :  
Employee Number :  
Name :  
Work Department :  
Hire Date :  
Salary :  
Bonus :  
Commission :

5. You will receive the response in the 'Employee Data' field:

BetaWorks IBM Integration Bus JSON Client application demo

Employee Number : enter employee number

Enter the Employee number to retrieve employee data:

Get Employee Data

**Employee Data**

Current Date : 2014-05-23  
Rows Returned : 1  
Employee Number : 000010  
Name : CHRISTINE I HAAS  
Work Department : A00  
Hire Date : 1995-01-01  
Salary : 152750  
Bonus : 1000  
Commission : 4220

Test the application using other Employee numbers. Some examples of existing 'employees' in the Database are **000020**, **000030**, **000050**.

**Note:** If you enter a non-existent Employee Number, the Integration Bus will return values as 'null'.

## 4. Adding data to the Environment Tree (optional)

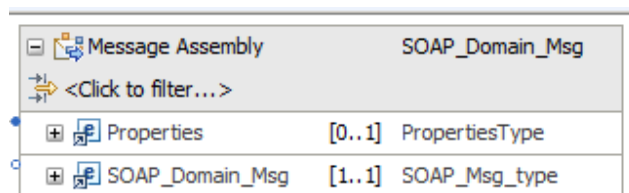
This final chapter shows you how to use the schemaless mapping tools in the Graphical Data Mapper to write data to the Environment tree.

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.

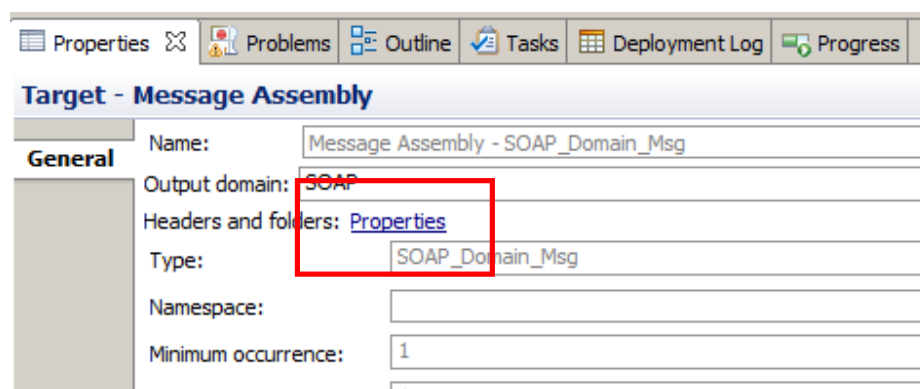
### 4.1 Configure the first mapping node

1. Reopen the first Mapping node, JSON\_to\_SOAP, and focus on the output message assembly.

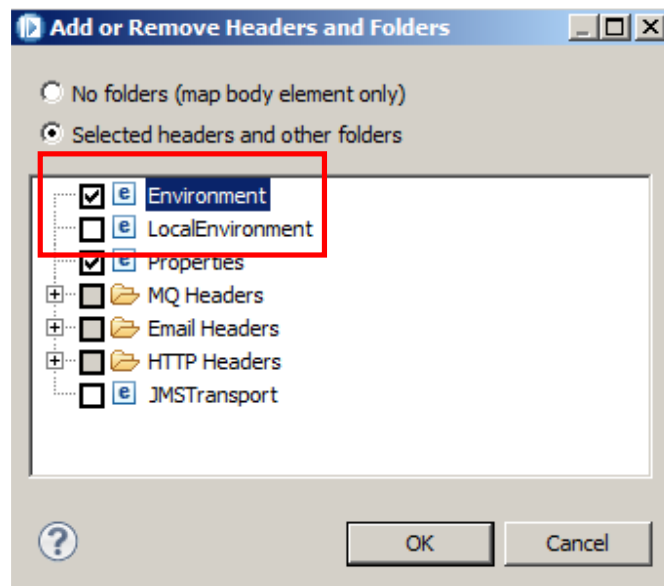


2. Highlight (click on) the output message assembly, and look at the Properties of the assembly.

Click the Headers and folders property (click the Properties item).

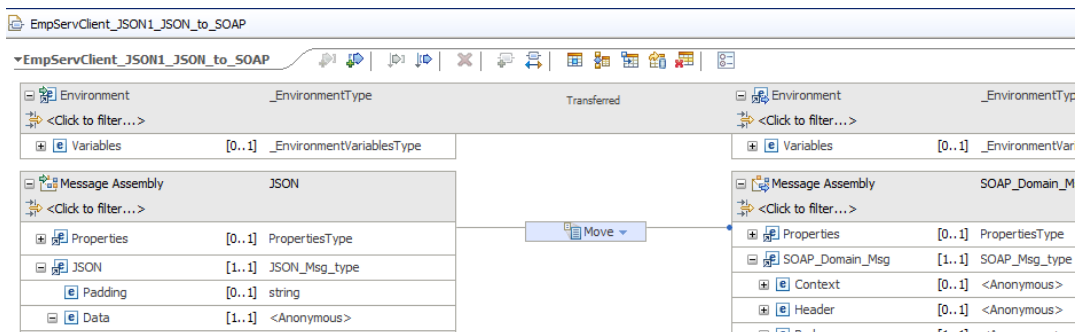


3. In the dialogue window, select the Environment, and click OK.

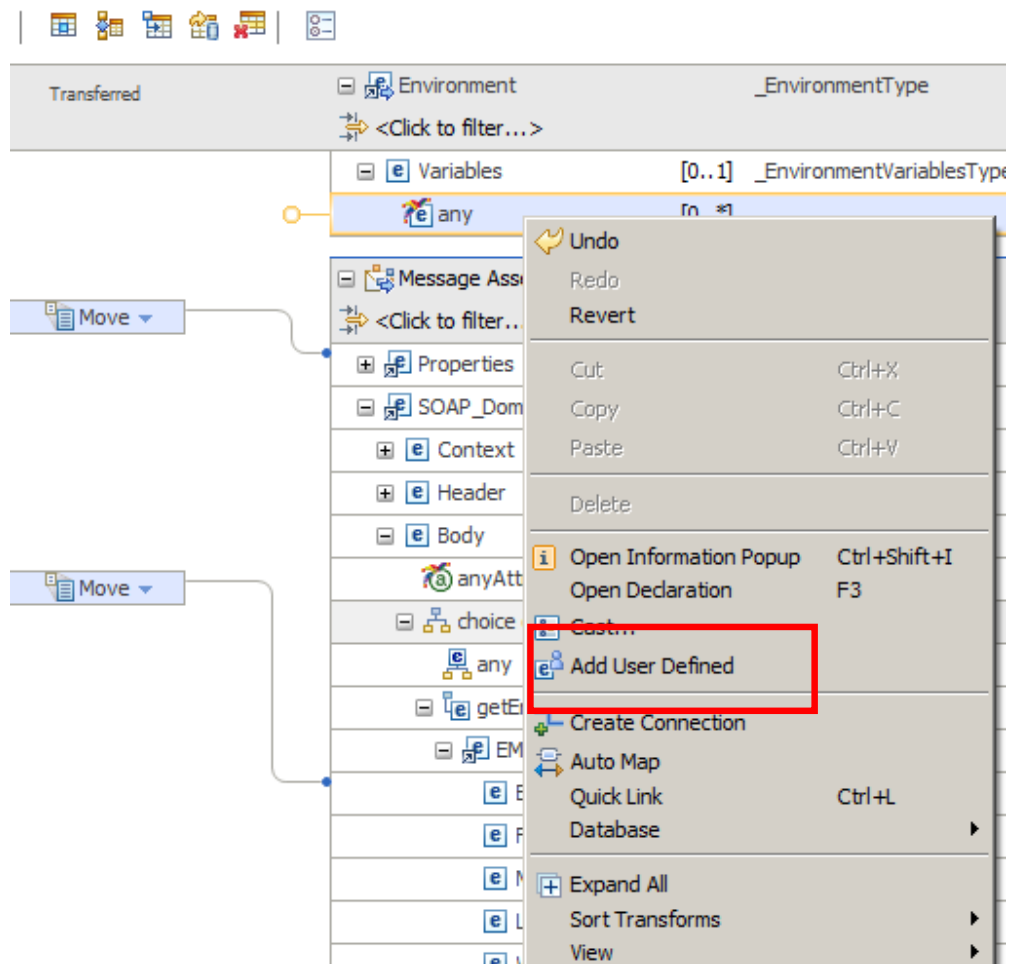


4. The output message assembly will have been updated to show the Environment.

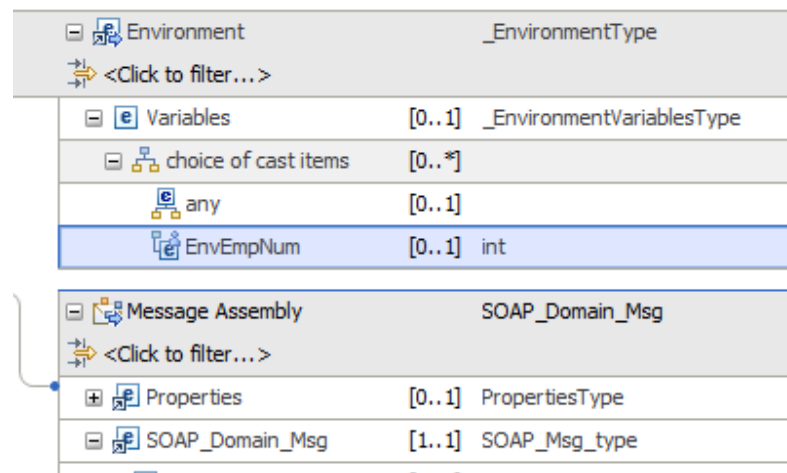
Note that because the Environment is a common area, this is displayed at the top of the map, bridging the input and output assemblies.



5. In the output Environment/Variables message assembly, right-click the "any" element, and select Add User Defined.



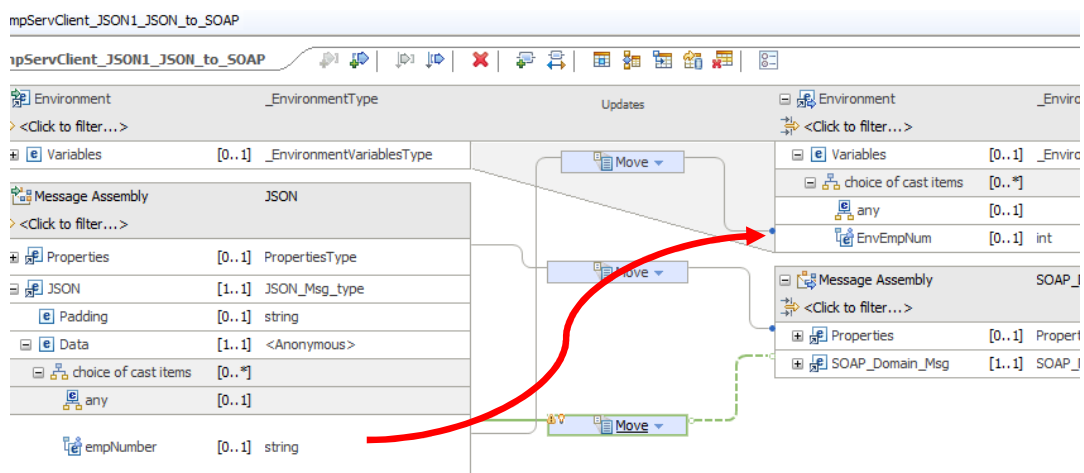
6. Name the new element EnvEmpNum and set the type to "int".



7. Map the input empNumber to the Environment element EnvEmpNum.

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.



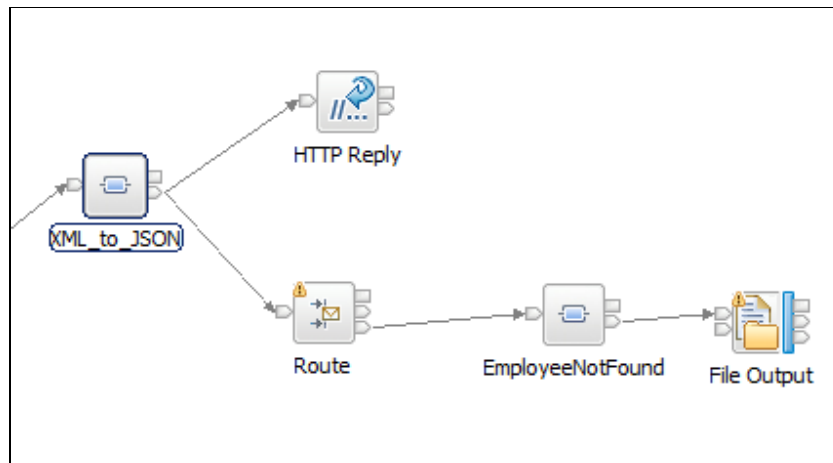
This map will now save the input employee number to the Environment tree for use later in the flow.

Save and close the Mapping node.

## 4.2 Add and Configure new nodes

8. Add three new nodes to the flow, and connect as shown.

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



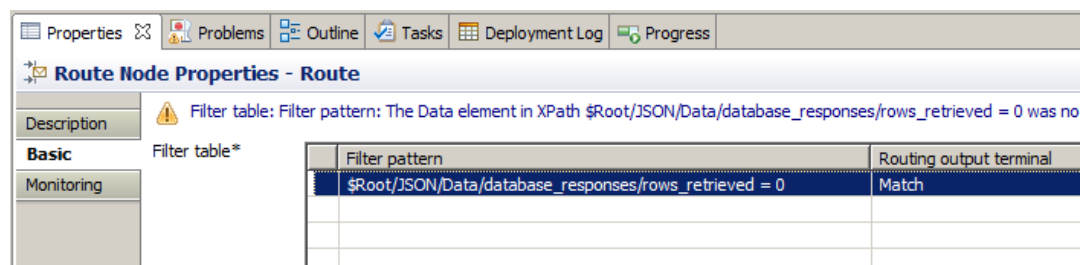
### 4.2.1 Configure the Route node

1. Configure the Route node. Highlight the node, and in the Properties, set the Filter pattern to **`$Root/JSON/Data/database_responses/rows_retrieved=0`**.

(Use the Add button to open a new filter pattern; you will manually type the above Filter. Note - this needs to exactly match the JSON elements that you created earlier).

Set the Routing output terminal to Match.

This test checks for `rows_retrieved=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.



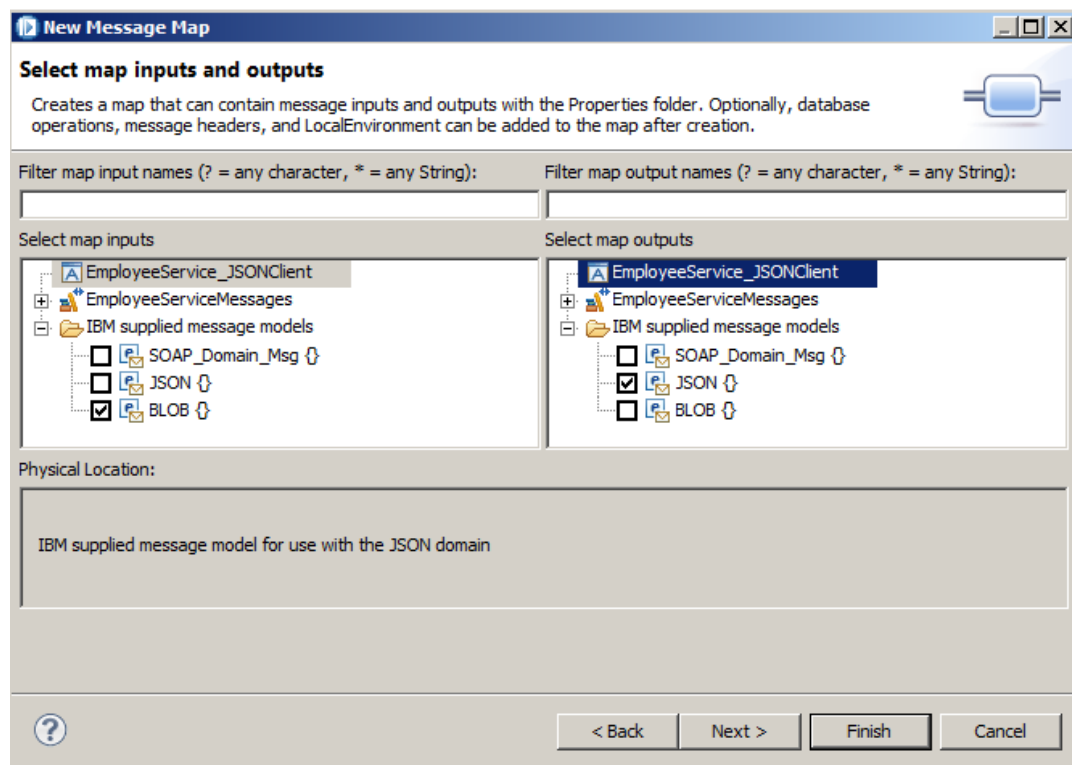
#### 4.2.2 Configure the new Mapping node

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

Set Input message assembly = BLOB

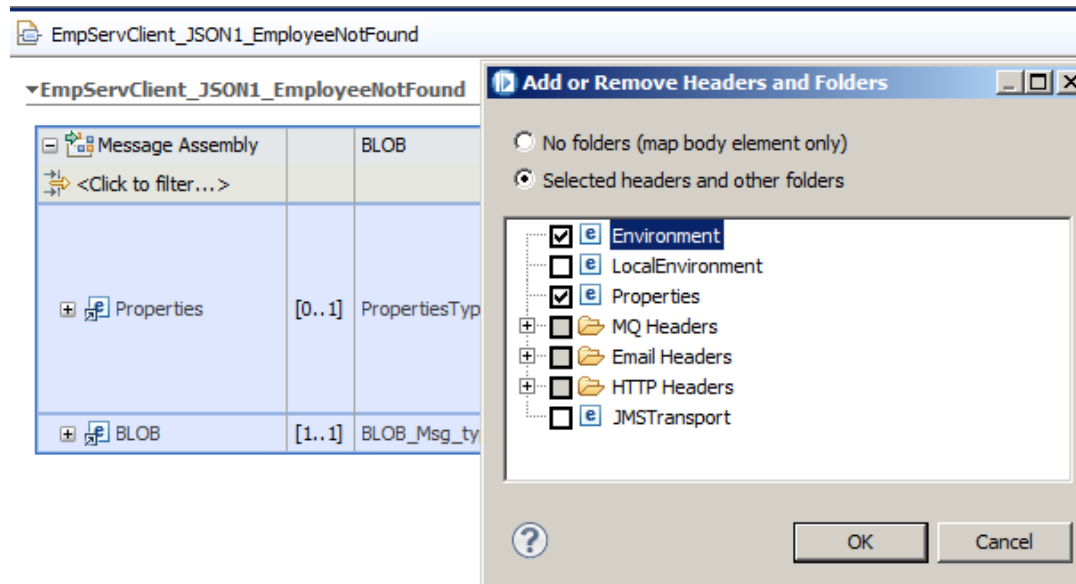
Set Output message assembly = JSON

Click Finish.

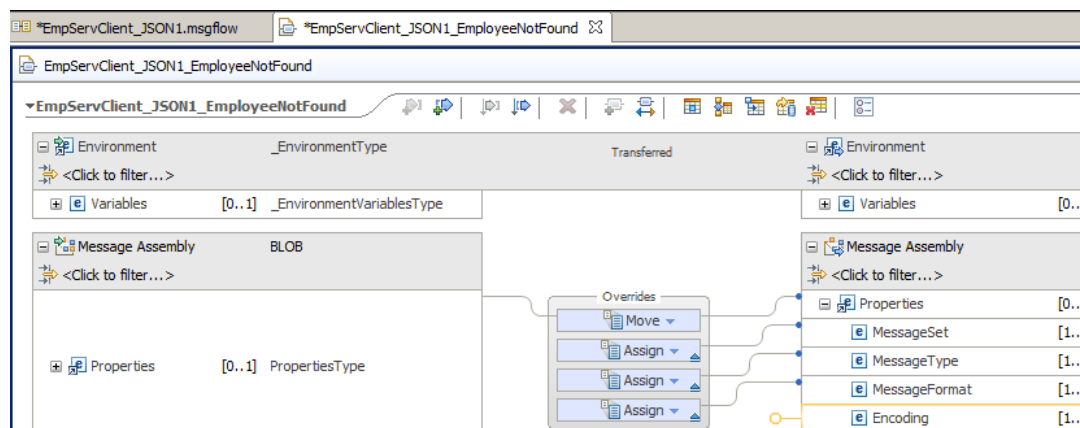


- In the map editor, focus on the input message assembly.

Add the Environment to the available Folders.

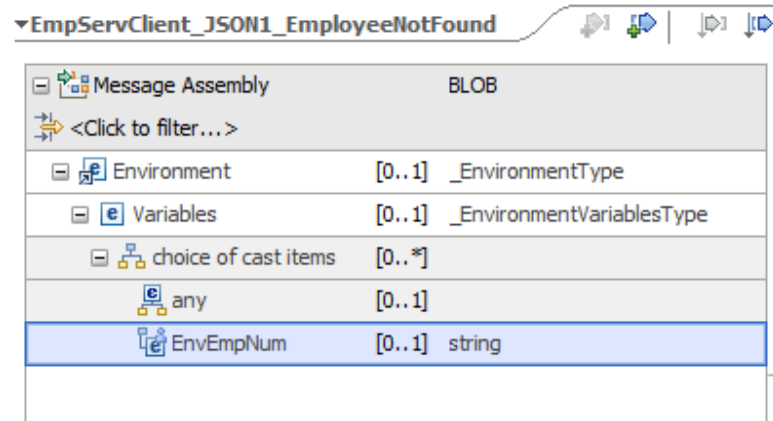


- This will show the Environment tree at the top of the map.





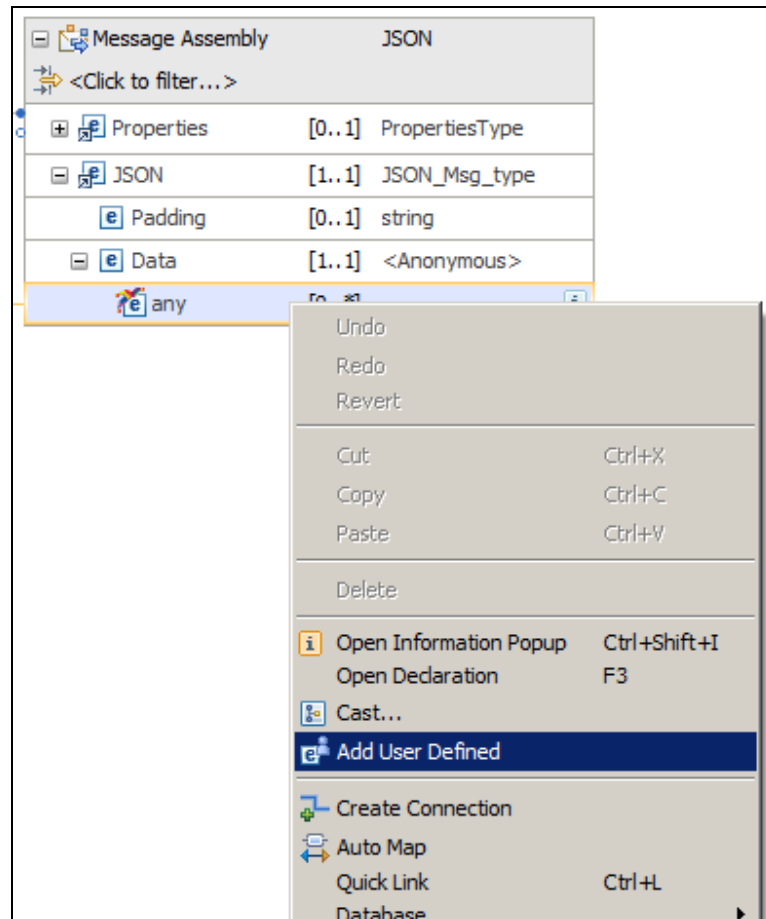
4. Using the same technique as earlier, on the input message, right-click "any" and add a User Defined element, named EnvEmpNum (ie. the one you added earlier).



▼EmpServClient\_JSON1\_EmployeeNotFound

|                      |                                  |
|----------------------|----------------------------------|
| Message Assembly     | BLOB                             |
| <Click to filter...> |                                  |
| Environment          | [0..1] _EnvironmentType          |
| Variables            | [0..1] _EnvironmentVariablesType |
| choice of cast items | [0..*]                           |
| any                  | [0..1]                           |
| EnvEmpNum            | [0..1] string                    |

5. In the map editor, expand the output message assembly. As above, add the following new elements to the JSON folder:
- outmessage (type = anonymous - will be set automatically)
    - text (type = string)
    - empNumber (type = string)
    - application (type = string)
    - flowname (type = string)
    - datetime (type = dateTime)



6. The result should be:

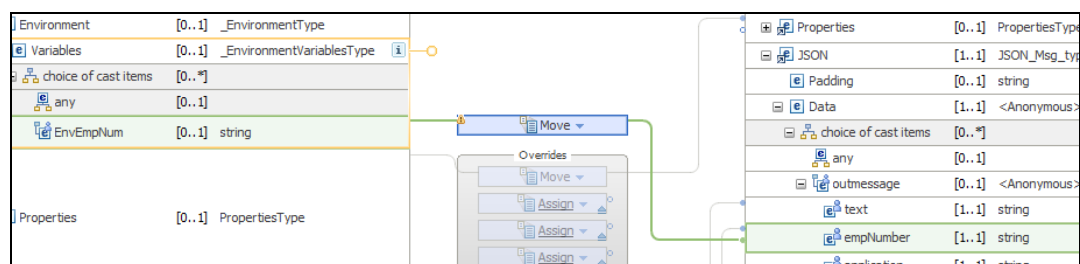
|                      |                       |
|----------------------|-----------------------|
| Message Assembly     | JSON                  |
| <Click to filter...> |                       |
| Properties           | [0..1] PropertiesType |
| JSON                 | [1..1] JSON_Msg_type  |
| Padding              | [0..1] string         |
| Data                 | [1..1] <Anonymous>    |
| choice of cast items | [0..*]                |
| any                  | [0..1]                |
| outmessage           | [0..1] <Anonymous>    |
| text                 | [1..1] string         |
| empNumber            | [1..1] string         |
| application          | [1..1] string         |
| flowname             | [1..1] string         |
| datetime             | [1..1] dateTime       |

#### 4.2.3 Create the mapping transforms

1. Create the transform for the new elements.

The "text" element. Use an 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 EnvEmpNum to output empNumber (a Move transform).



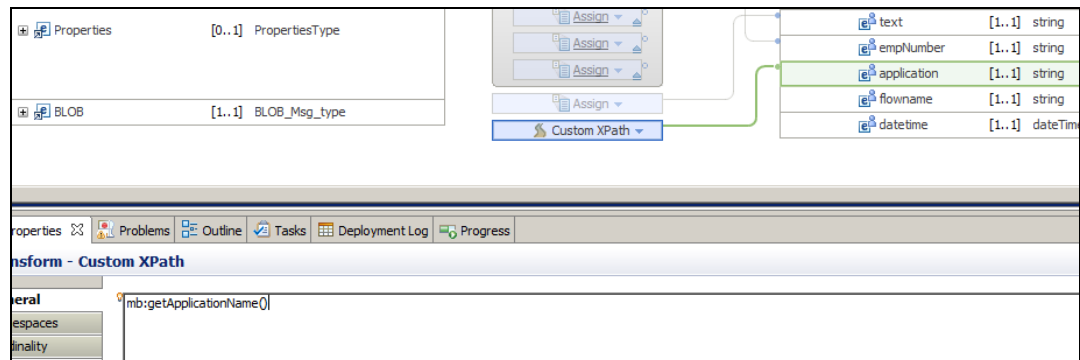
3. For the output "application". right-click "application" and select "Create Assign".

Change the Assign transform to Custom XPath.

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

**mb:getApplicationName()**

(Note - this value is not yet available using the content assist function in IIB V10).



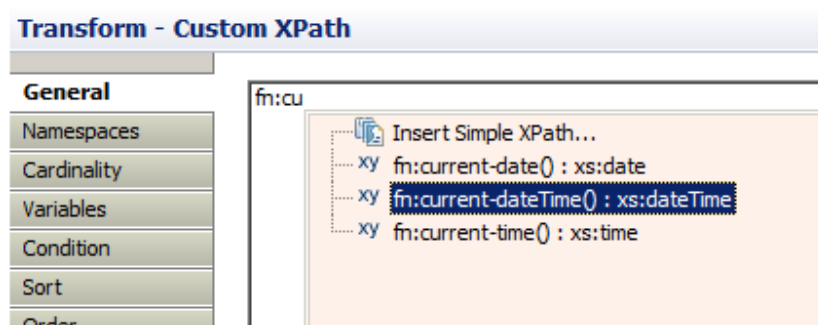
4. For the output "flowname", create a similar transform to application Set the XPath statement to

**mb:getMessageFlowName()**

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

This time, you can use the content assist function 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:\student10\integration\_service\temp
- File name = JSONAppError.txt
- File action = Write directly to output file

**File Output Node Properties - File Output**

|                      |  |
|----------------------|--|
| Description          |  |
| <b>Basic</b>         | Directory: c:\student10\integration_service\temp<br>File name or pattern: JSONAppError.txt<br>File action:<br>Mode for writing to file:<br><input checked="" type="radio"/> Write directly to the output file (append if file exists)<br><input type="radio"/> Stage in mqsitransit directory and move to output directory on "Finish file"<br>Action if file exists: Replace Existing File<br>Replace duplicate archive files: <input type="checkbox"/> |
| Request              |  |
| Records and Elements |  |
| Validation           |  |
| FTP                  |  |
| Monitoring           |  |

2. On the node properties Request tab, set:

- Data location = \$Root/JSON

Ignore the warning that the JSON element is not found in the XML schema.

**File Output Node Properties - File Output**

|                      |   |   |
|----------------------|---|---|
| Description          | Data location: The JSON element in XPath \$Root/JSON was not found in the XML Schema. |   |
| <b>Basic</b>         |   |   |
| <b>Request</b>       |   |   |
| Records and Elements | Request directory property location   | \$LocalEnvironment/Destination/File/Directory |
| Validation           | Request file name property location   | \$LocalEnvironment/Destination/File/Name      |
| FTP                  |   |   |
| Monitoring           |   |   |

### 4.3 Deploy and Retest

1. Save the flow. Redeploy the application (drag the application onto server1).

Test the application again, either with SOAPUI or the web browser client

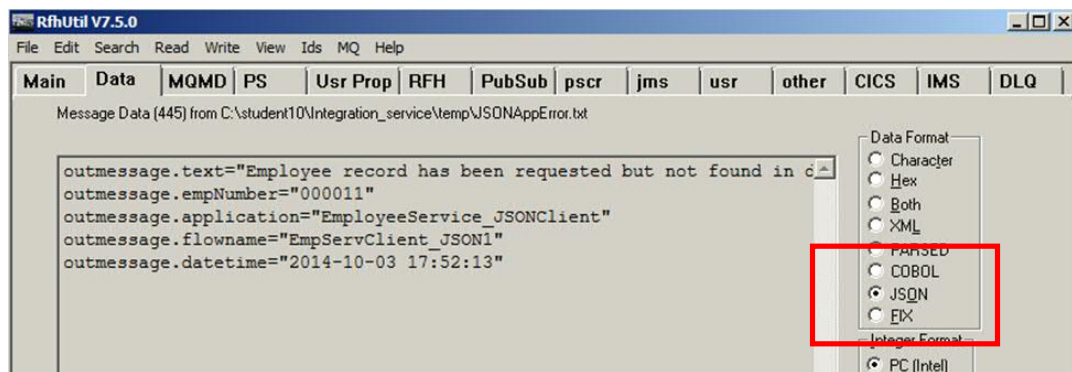
The response to the Test Client (or Nettool http client) will be the same as before.

Run the test again, making sure that you use a non-existent employee number, eg. 000011.

2. Open RFHUtil from the Start menu.

Open the file c:\student10\integration\_service\temp\JSONAppError.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.



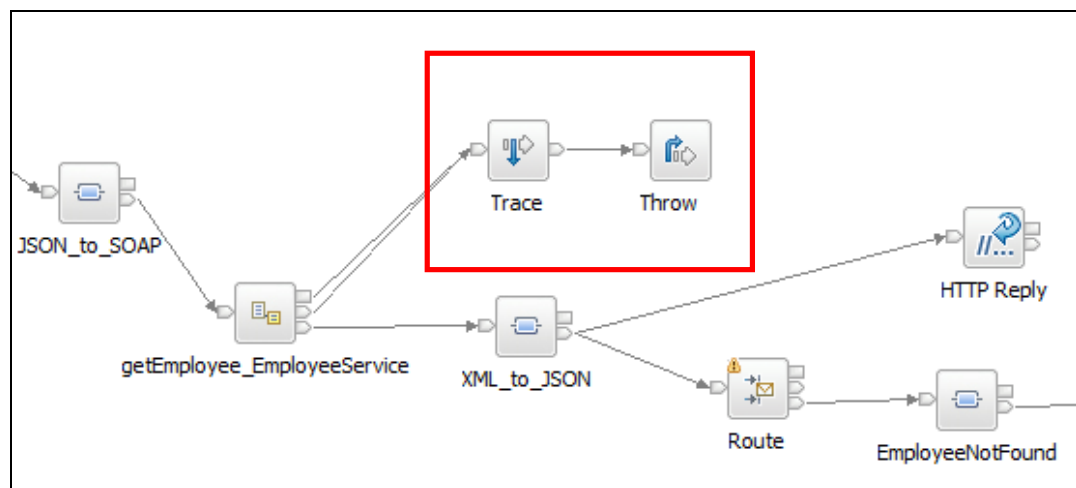
## 5. Add fault and failure handing (optional)

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.



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

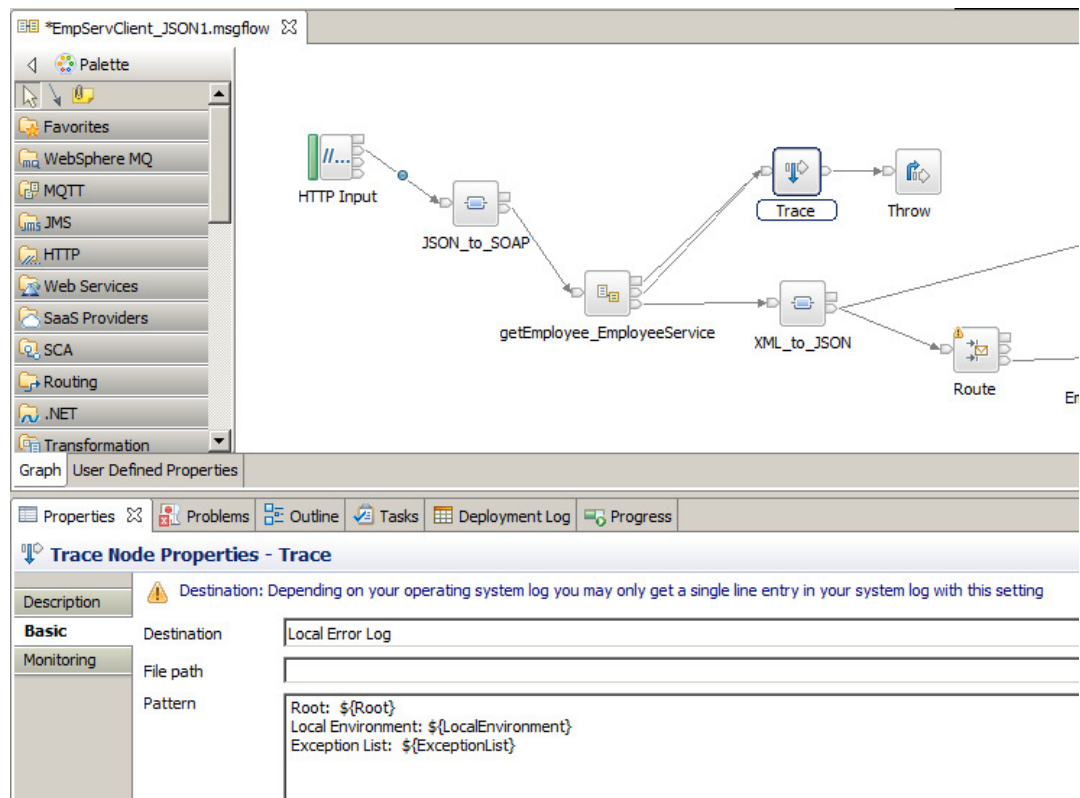
Set the Destination to "Local Error Log". (in your own environment, you may want to set this to a file or user log output).

Set the Pattern to:

**Root = \${Root}**

**Local Environment = \${LocalEnvironment}**

**Exception List = \${ExceptionList}**



- Save the flow, and redeploy the application.

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 Test Client. 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 Test Client, and in the Windows Event Log.



4. The IIB Error Log Console monitor will 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.

```
BIP2232E: < IB10NODE.server1 > Error detected whilst handling a previous error i
n node 'EmpServClient_JSON1.Throw'. [10/3/2014 5:57:14 PM]
BIP2230E: < IB10NODE.server1 > Error detected whilst processing a message in nod
e 'EmpServClient_JSON1.getEmployee_EmployeeService.Request'. [10/3/2014 5:57:1
4 PM]
BIP3754E: < IB10NODE.server1 > The SOAP Request Node or SOAP Async Request Node
'EmpServClient_JSON1.getEmployee_EmployeeService.Request' encountered an error w
hile processing the outbound SOAP request. [10/3/2014 5:57:14 PM]
BIP3162E: < IB10NODE.server1 > An HTTP error occurred. The HTTP Request-Line was
: 'POST /EmployeeService HTTP/1.1
'. [10/3/2014 5:57:14 PM]
BIP3152E: < IB10NODE.server1 > Socket error detected whilst invoking Web service
located at host 'localhost', port '7800', path '/EmployeeService'. [10/3/2014 5
:57:14 PM]
BIP3150E: < IB10NODE.server1 > A socket error occurred. Operation: '::connect::s
elect<>'. Error Code: '10061'. Error Text: 'No connection could be made because
the target machine actively refused it.
'. [10/3/2014 5:57:14 PM]
BIP3120E: < IB10NODE.server1 > Exception condition detected on input node 'EmpSe
rvClient_JSON1.HTTP Input'. [10/3/2014 5:57:14 PM]
BIP3001I: < IB10NODE.server1 > Exception thrown by throw node 'EmpServClient_JSO
N1.Throw'; text is ' '. [10/3/2014 5:57:14 PM]
```

## 6. Invoking EmployeeService integration service asynchronously (optional)

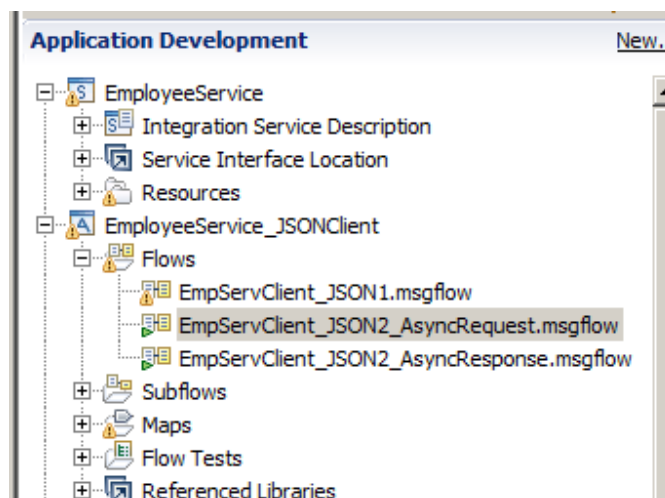
The first part of this lab invokes the EmployeeService synchronously. This is constructed by dropping the WSDL of the EmployeeService directly onto the flow editor of the JSON1 message flow. The Integration Studio uses this to include a SOAP Request node in the message flow (subflow).

At execution time, the flow invokes the service synchronously. If the service is not available, then the JSON1 application will throw an error, and the client will not receive the requested data.

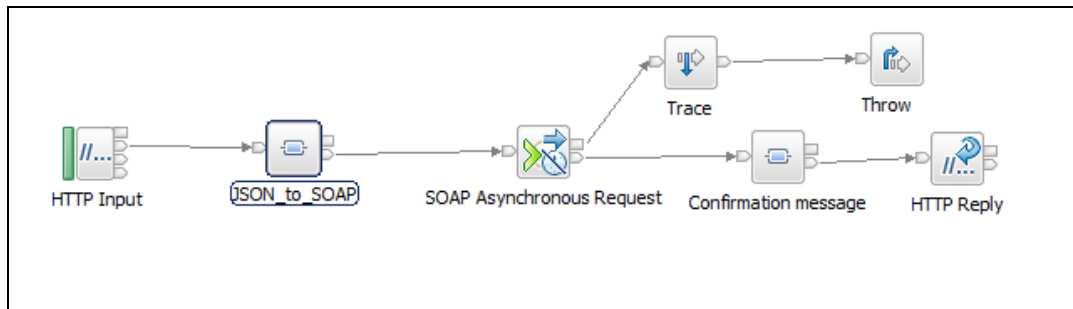
IBM Integration Bus provides the capability to invoke a web service asynchronously. In Version 8, this had to be done by using WS-Addressing, and in Version 9 web services can be invoked asynchronously by simply providing a local identifier string that the IIB runtime uses to correlate the request and 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:
  - EmpServClient\_JSON2\_AsyncRequest
  - EmpServClient\_JSON2\_AsyncResponse



2. For the JSON2\_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\_JSON2  
 Input Message Parsing: Message domain = JSON

#### Mapping node: JSON\_to\_SOAP

Basic: Mapping routine= {default}:EmpServClient\_JSON1\_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

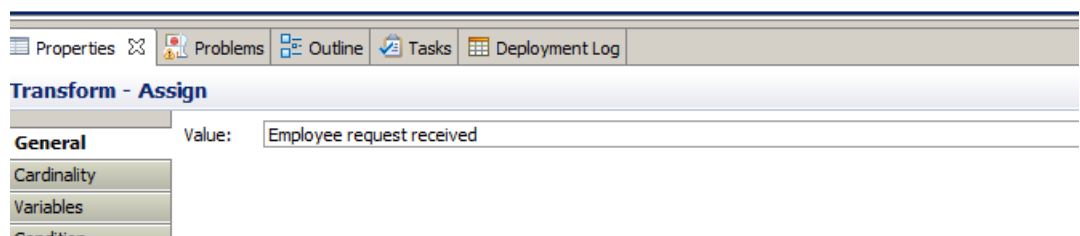
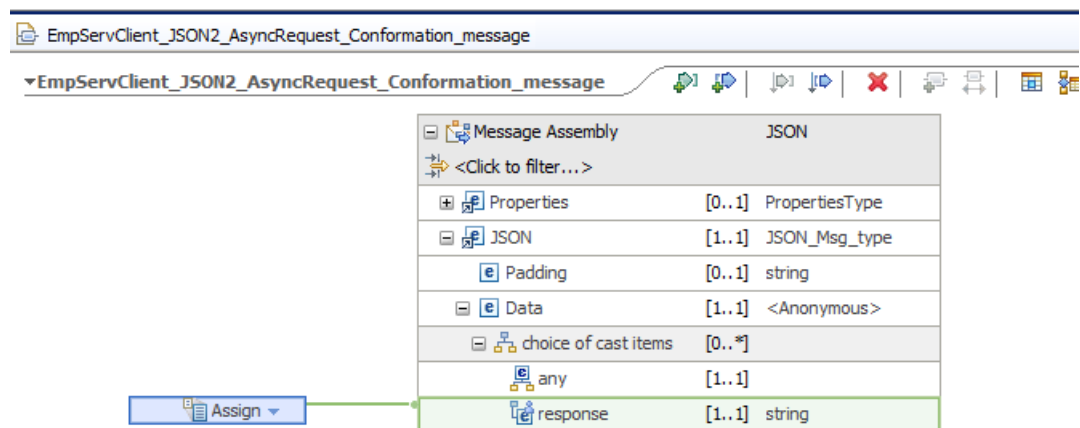
### 3. Configure the new mapping node: Confirmation Message

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

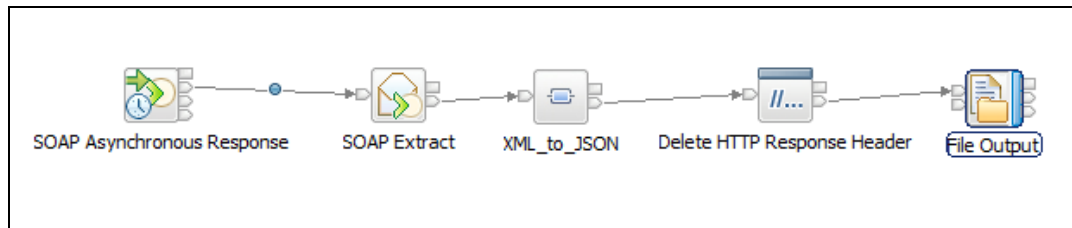
In the mapping editor, under the JSON/Data assembly, create a new element called "response".

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 JSON2\_AsyncResponse flow, construct the flow as shown here:



Set the node properties as follows:

SOAP Async Responset

Basic: Unique identifier = EmpServ123

Validation: Validate = None

Mapping node: XML\_to\_JSON

Basic: Mapping routine= {default}:EmpServClient\_JSON1\_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:\student10\integration\_service\temp

File name = JSON2.Async.out

File action: Write directly to output file

5. Deploy the EmpServClient\_JSON application in the usual way.

Ensure that the EmployeeService is Started.

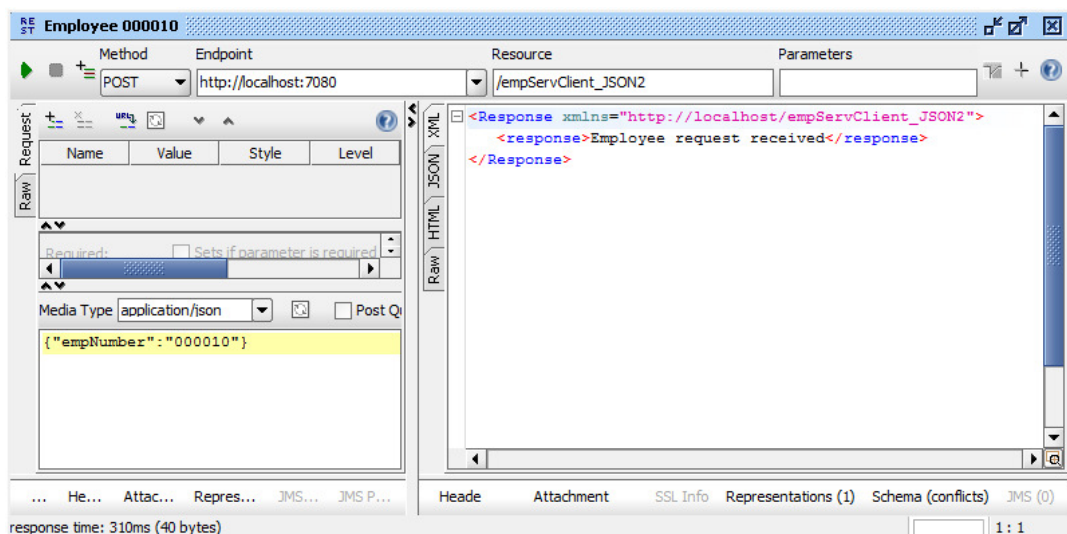
## 6. Test the new flows with SOAPUI.

In SOAPUI, expand the project JSON App - Async.

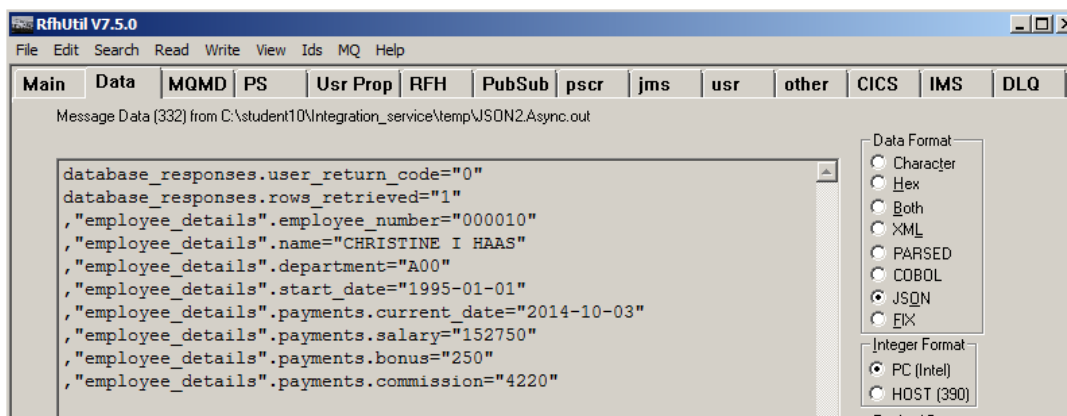
Open the request "Employee 000010".

Note that the URL for this project is set to /empServClient\_JSON2.

Click the green arrow, and see that the response data is simply the message "Employee request received".



## 7. In RFHUtil, open the file c:\student10\integration\_service\temp\JSON2.Async.out. On the Data tab, look at the data retrieved by the async web service.



This concludes the JSON Application lab.