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

1. Prepare the Provider Service

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

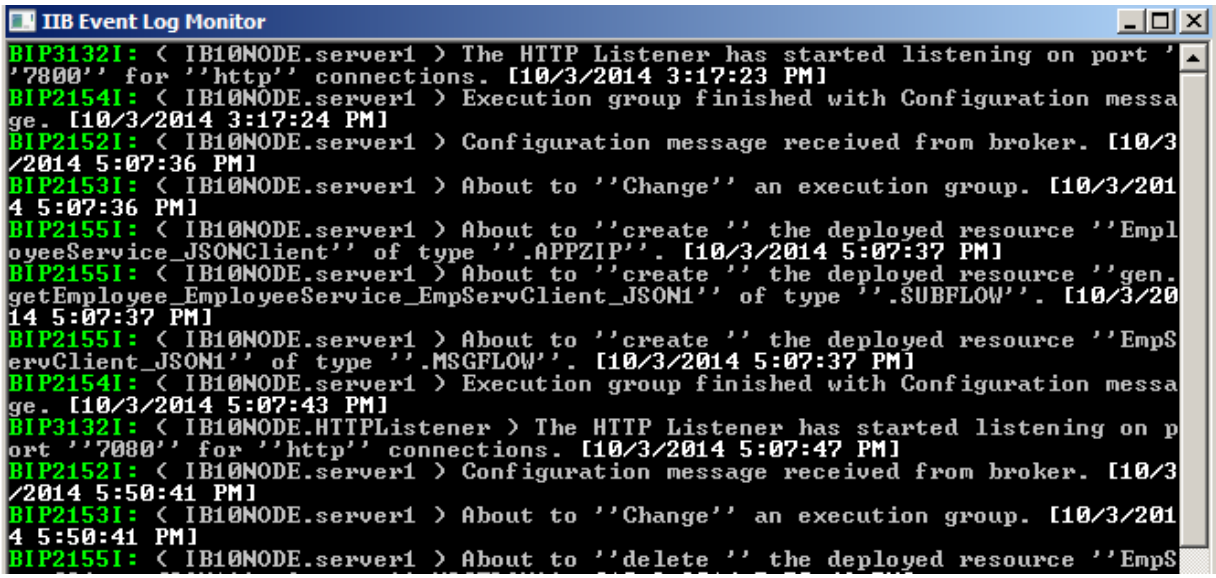
1. Use a web service connection to invoke an existing Integration Service (EmployeeService).
2. Use the Graphical Data Mapper to demonstrate schemaless mapping, mapping JSON input and output data.
3. Use the Graphical Data Mapper to write and read elements to the Environment tree.
4. Use the Graphical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



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.

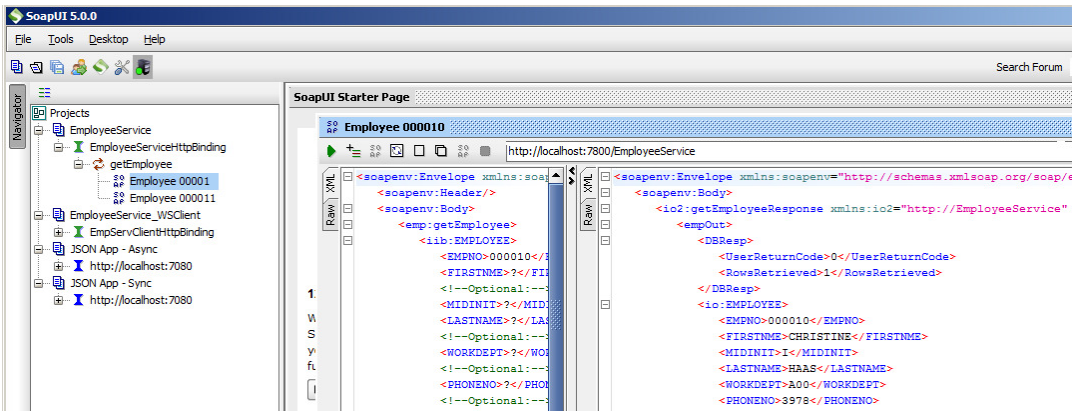
   This will overwrite any existing services of the same name in server1.

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.

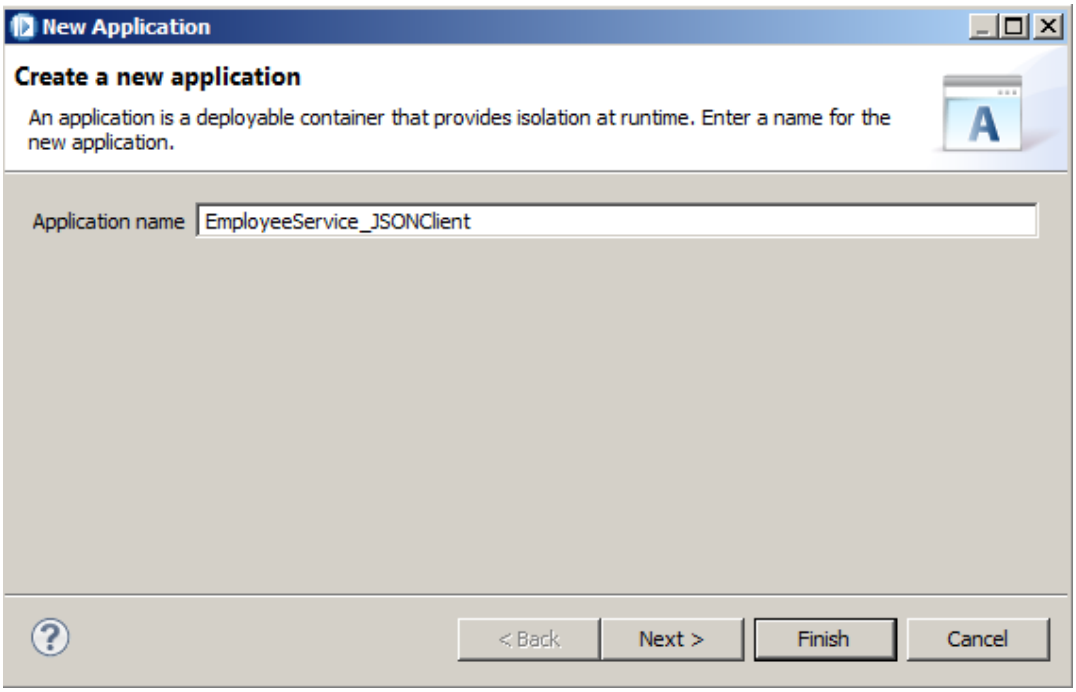
The EmployeeService uses the EMPLOYEE tables of the SAMPLE database, and has implemented the getEmployee operation.

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.

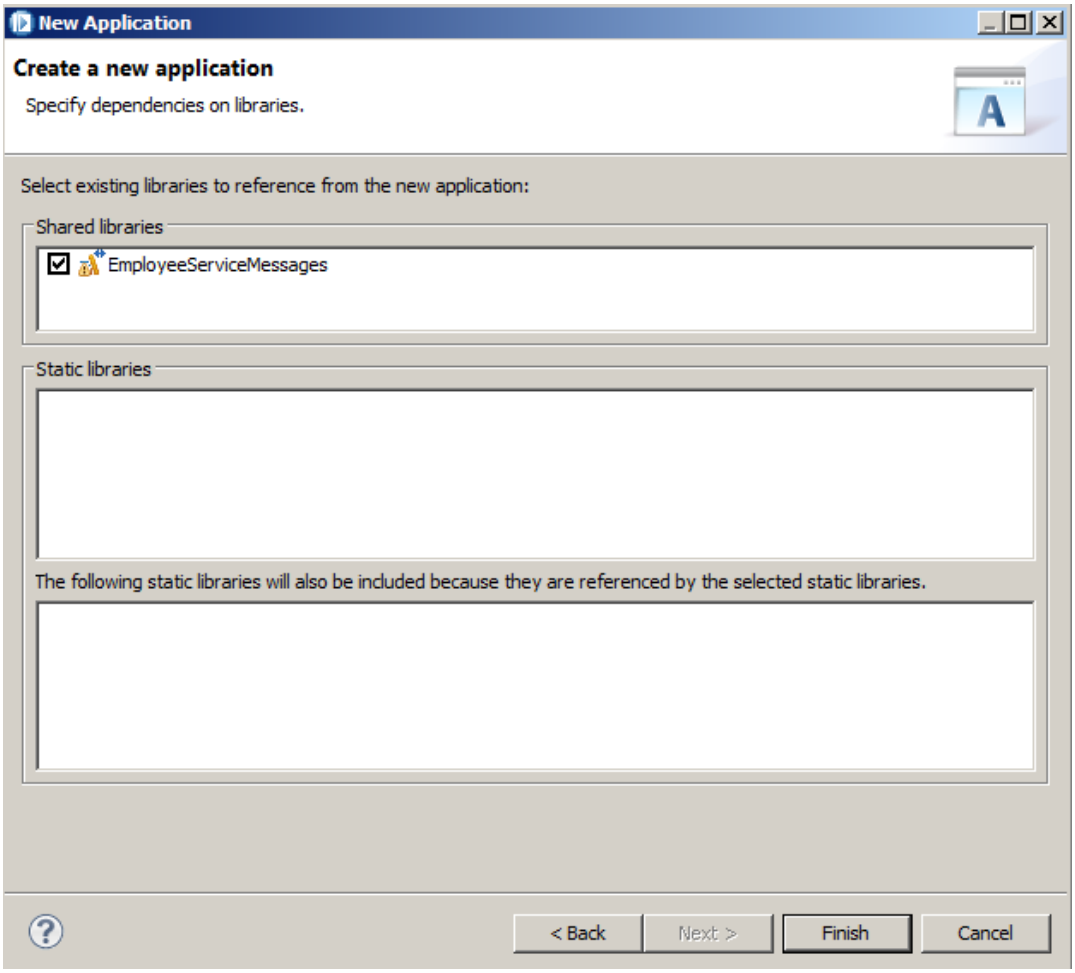
1. In the workspace, create a new Application named EmployeeService_JSONClient.

   Click Next.

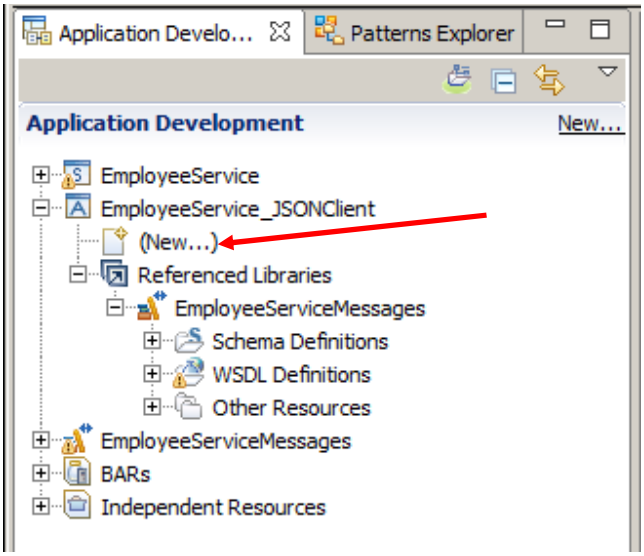


2. Select EmployeeServiceMessages as a referenced Shared Library.

   Click Finish.

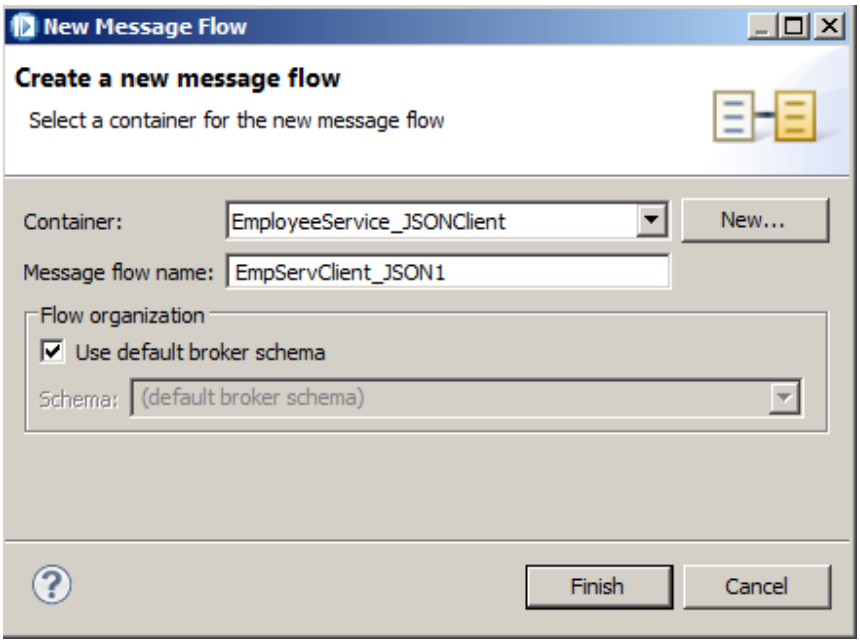


3. Click (New...) to create a new message flow.



Name the new flow EmpServClient_JSON1.

Click Finish.

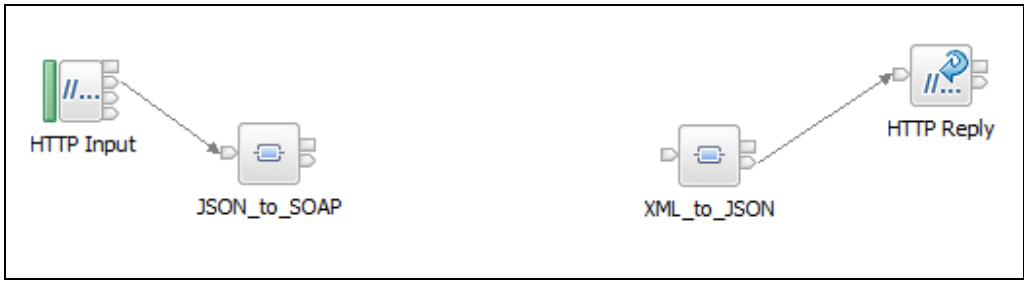


4. Drop the following nodes onto the flow editor, and name them as shown.

   - HTTP Input
   - Mapping Node - name it "JSON_to_SOAP"
   - Another Mapping Node - name it "XML_to_JSON"
   - HTTP Reply

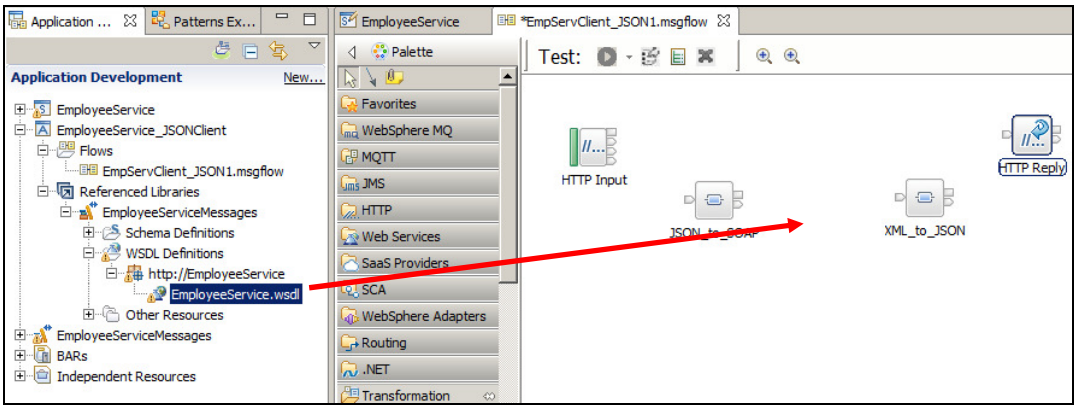
   For the HTTP Input node:

   - Set the Basic URL Path Suffix property to /**empServClient_JSON1** (the forward slash is important).
   - Set the Input Message Parsing Message Domain to **JSON**.



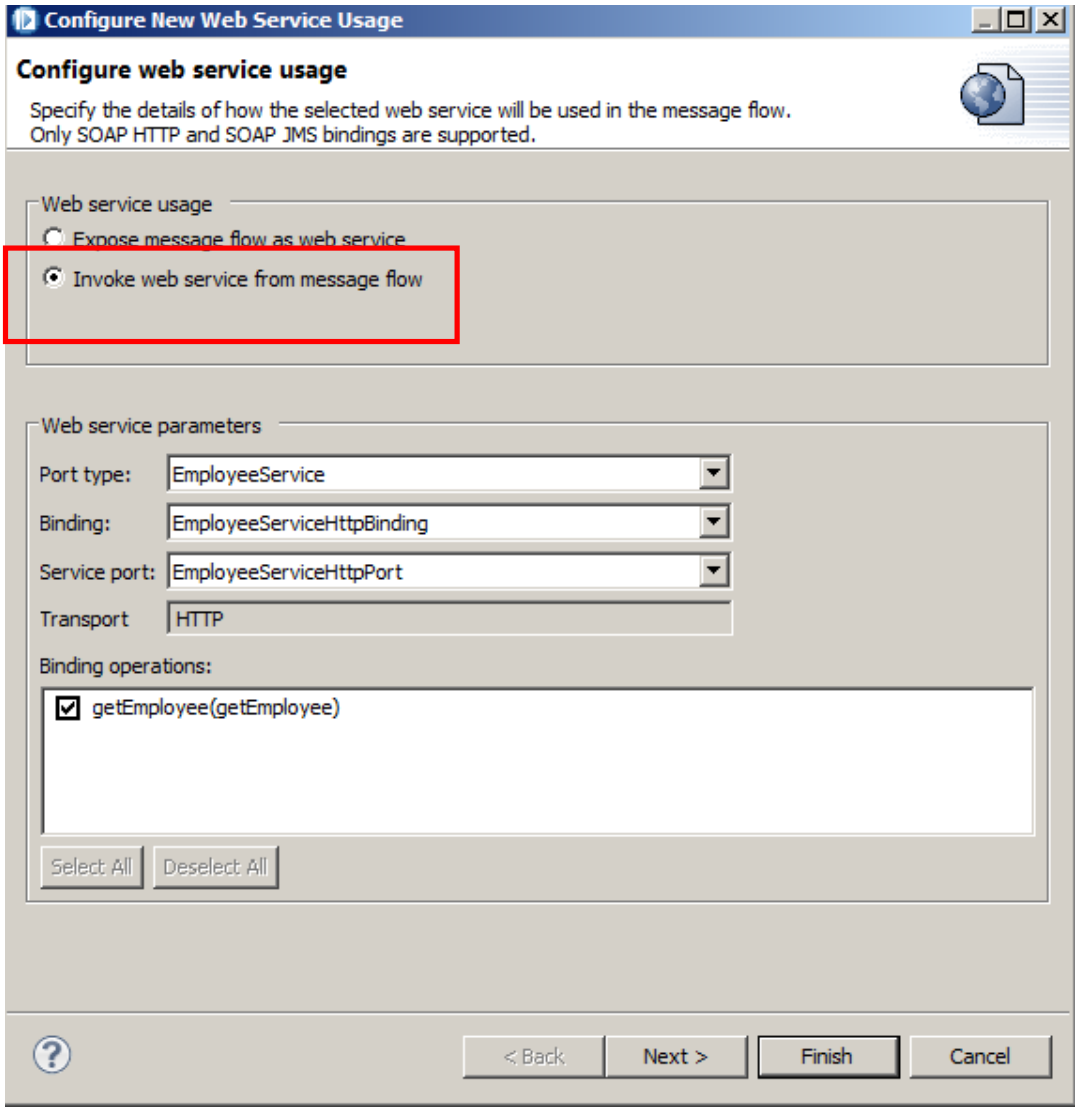
5. In the EmployeeService_JSONClient application, expand the Referenced Libraries folder, then the WSDL Definitions folder. You will see that it contains the WSDL of the EmployeeService.

   Drag and drop this wsdl onto the flow editor, between the two mapping nodes.



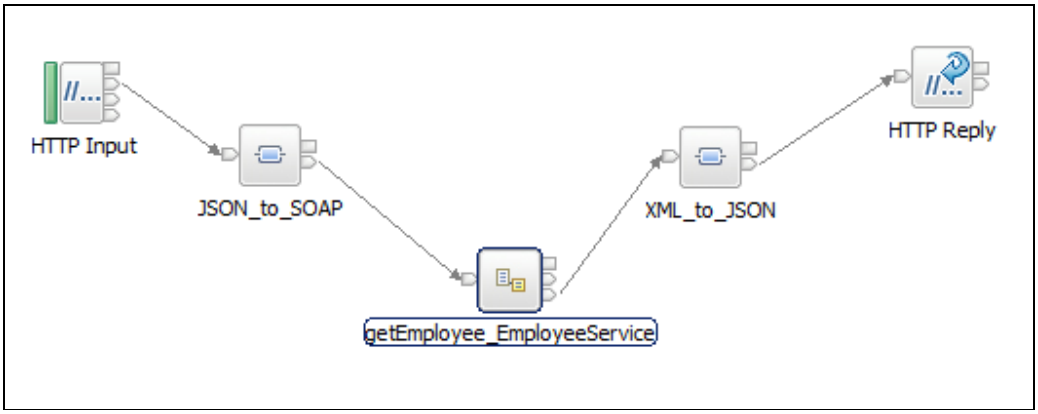
6. When you drop the wsdl onto the flow editor, the Configure New Web Service window will open, as shown below.

   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.



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.

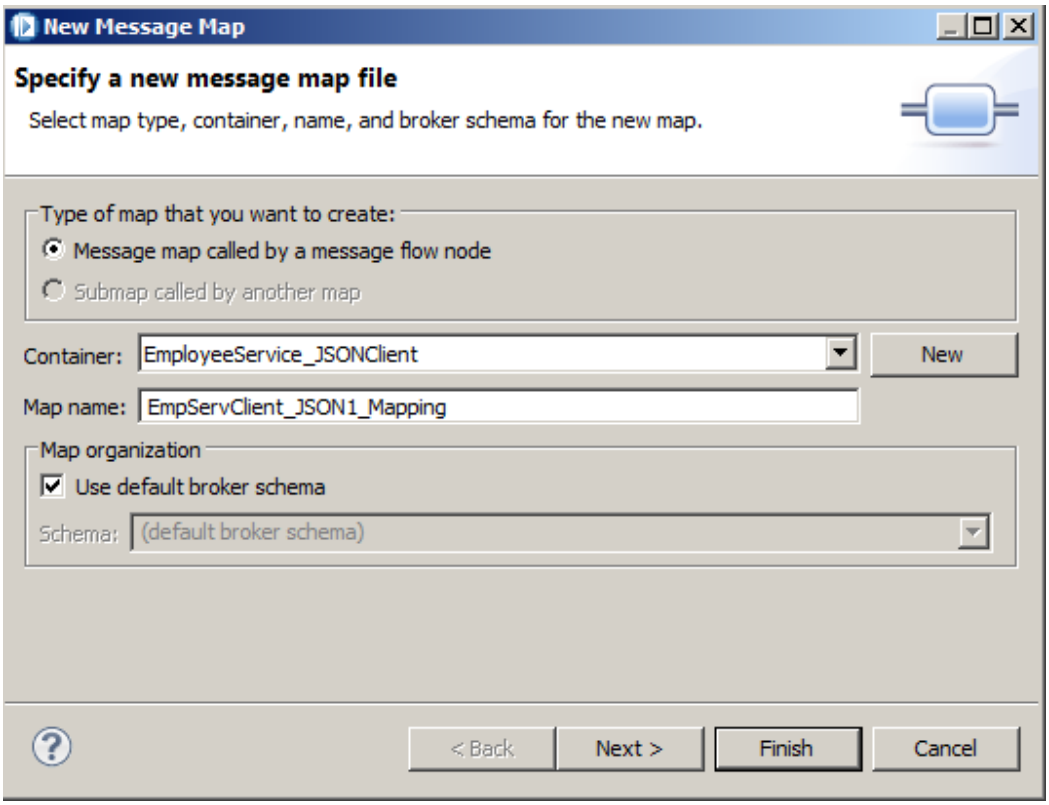


2.1 Implement the Mapping Nodes

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).

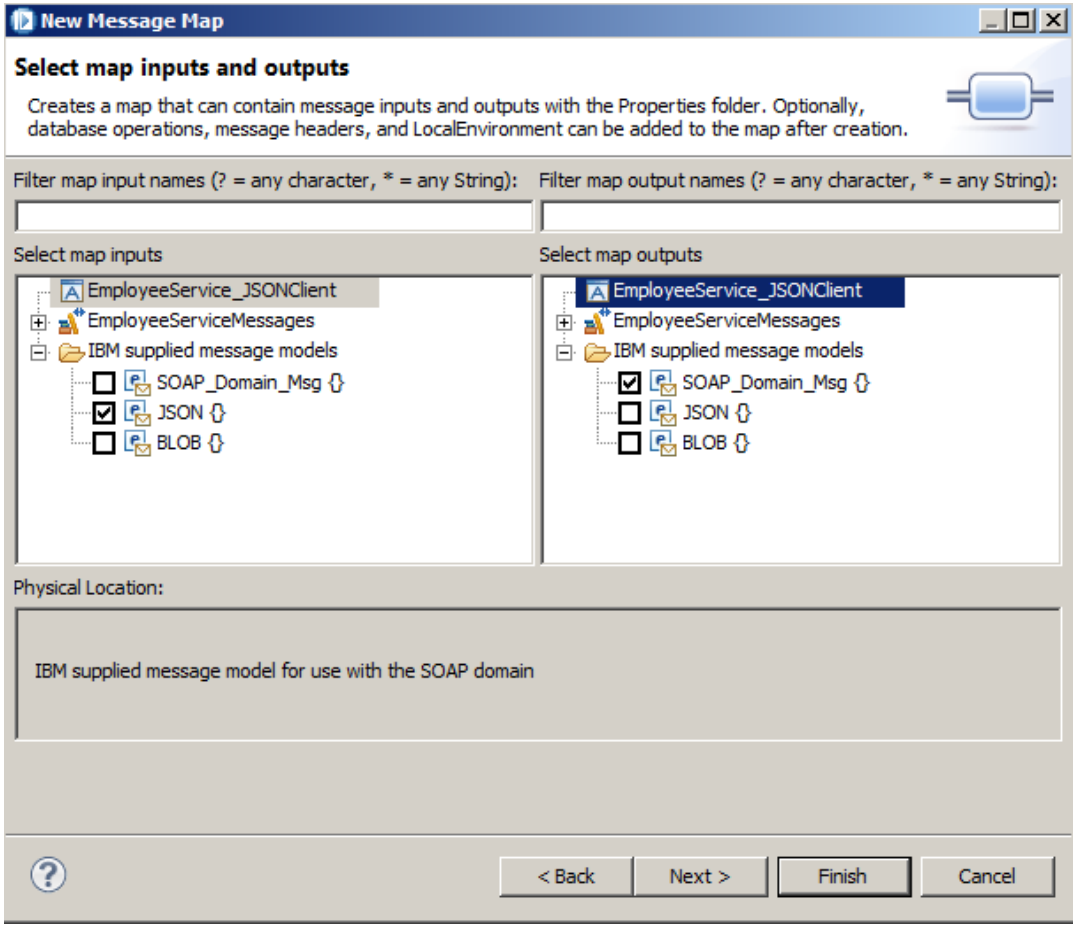
1. Double-click the first Mapping Node. At the first pop-up, click **Next**.



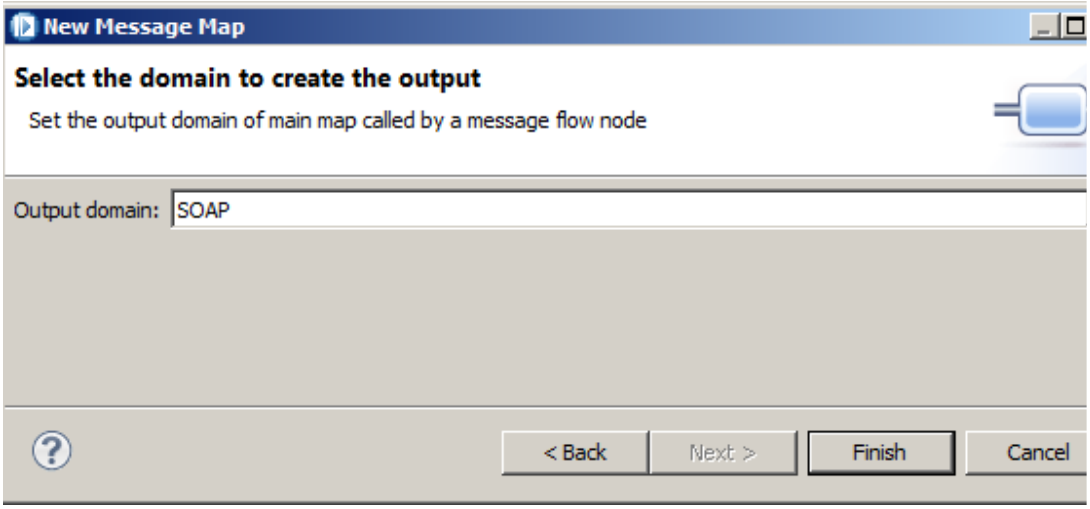
2. At the next window, select the map inputs and outputs.

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

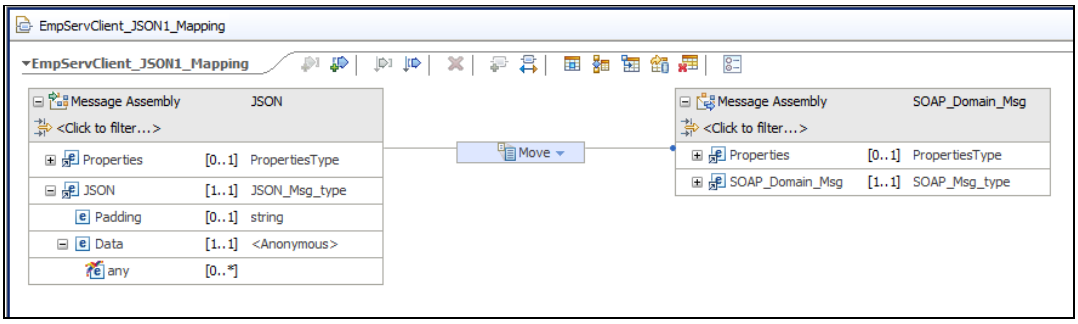
   Click **Next**.



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



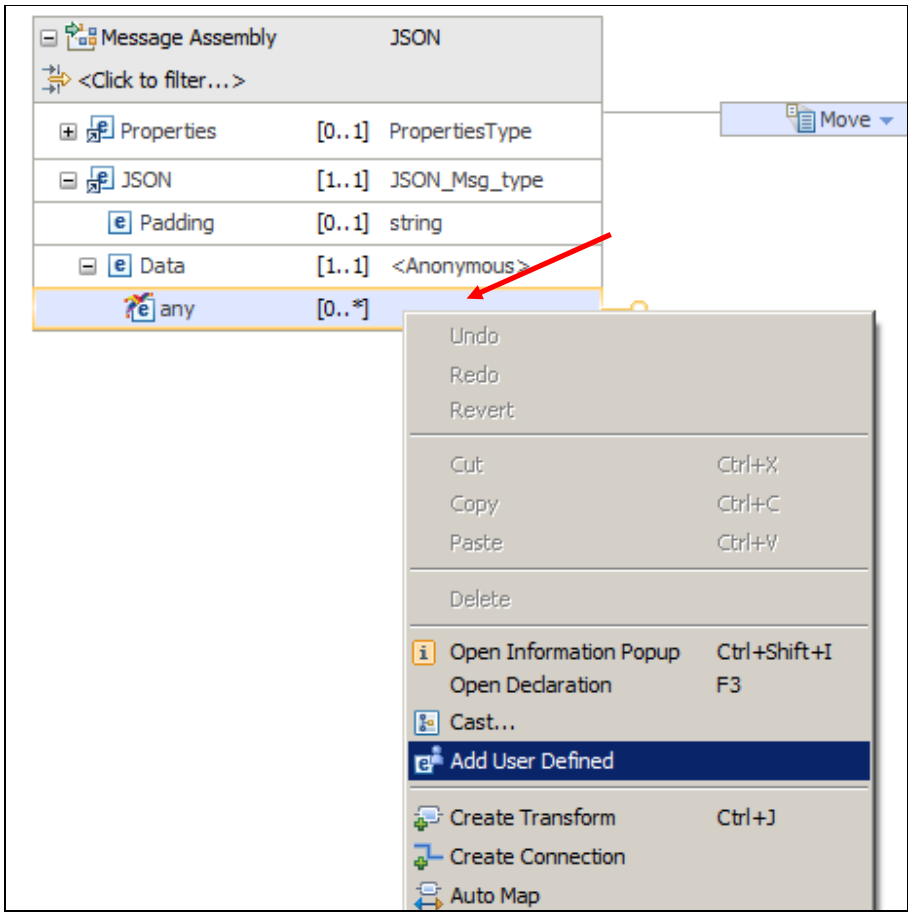
4. The mapping editor will load. Expand the Message Assembly for the input message.



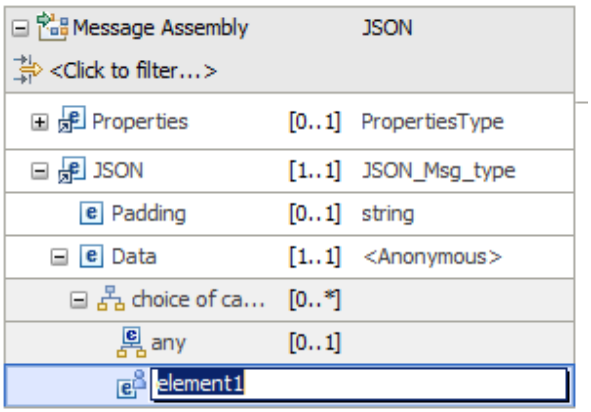
5. You are going to add a user-defined element to the JSON part of this input message assembly.

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

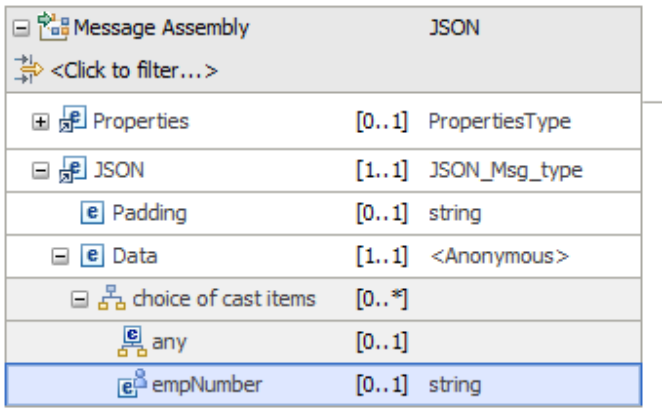
   Right-click the "any" element, and select "Add User Defined" from the context menu.



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



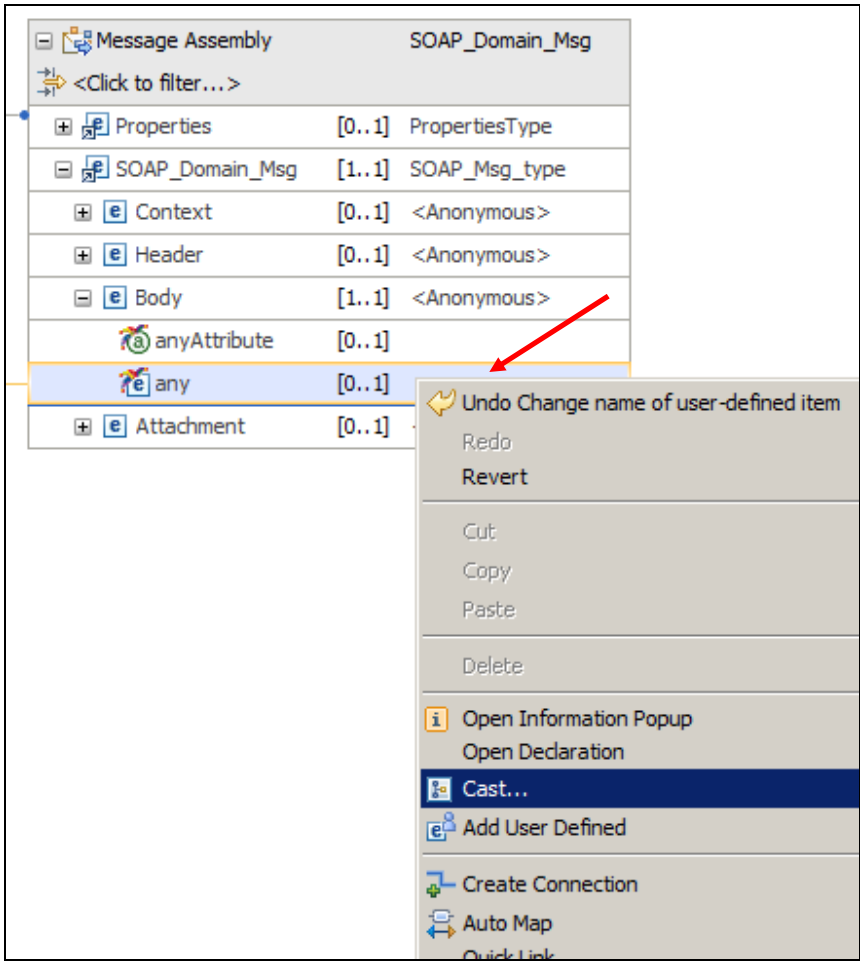
7. Change this to "empNumber" by overtyping "element1". Leave the element type as "string".



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 EmployeeService interface.

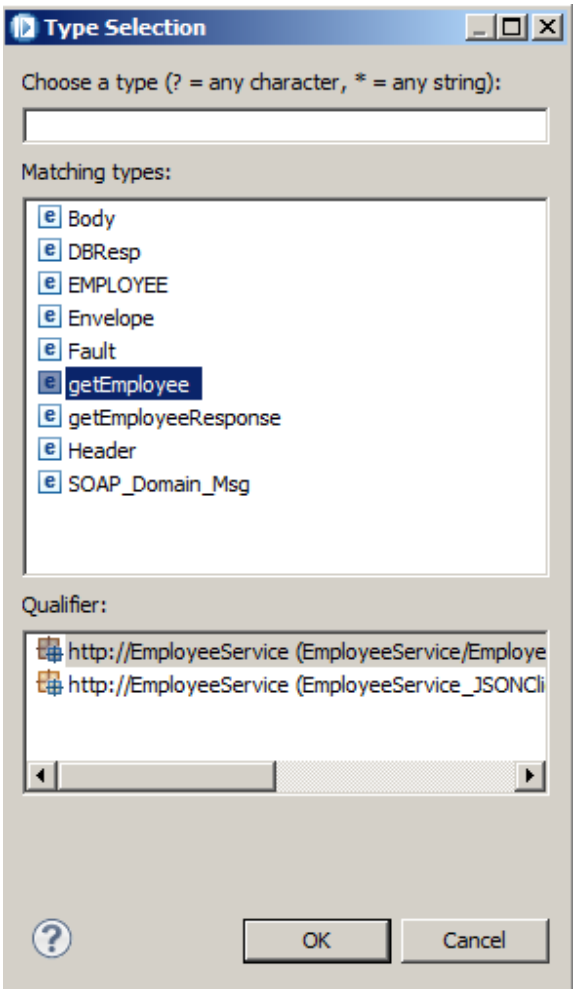
   In the output message assembly, expand the SOAP_Domain_Msg, then Body.

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



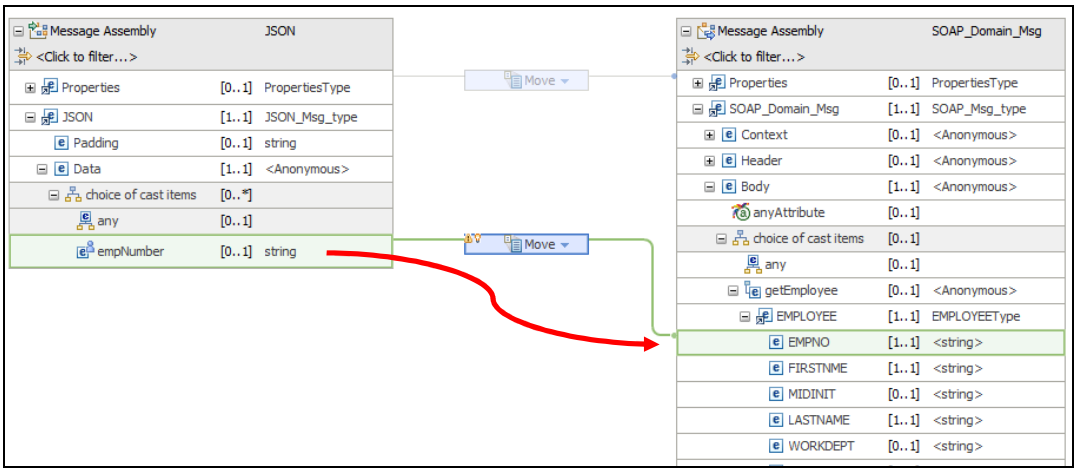
9. From the Type Selection window, highlight getEmployee, and click OK.

   (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.

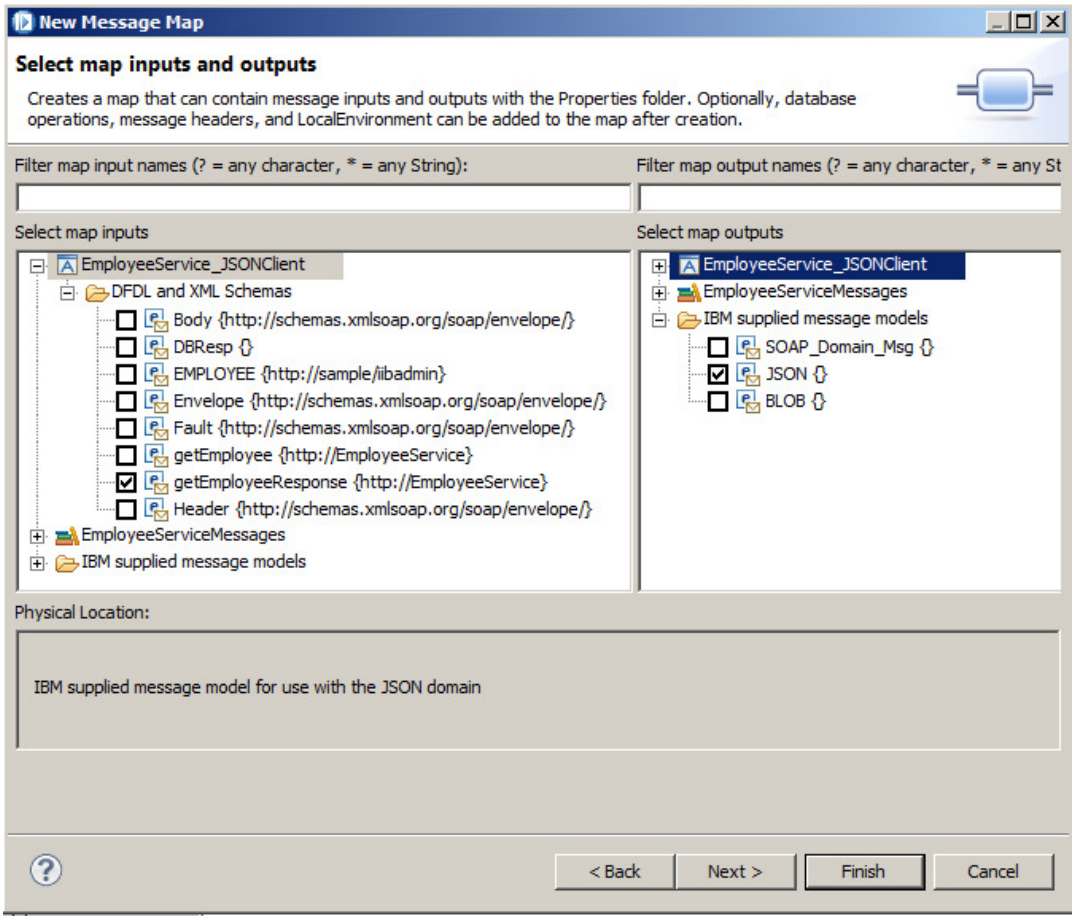
11. Configure the second mapping node (XML_to_JSON).

    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.

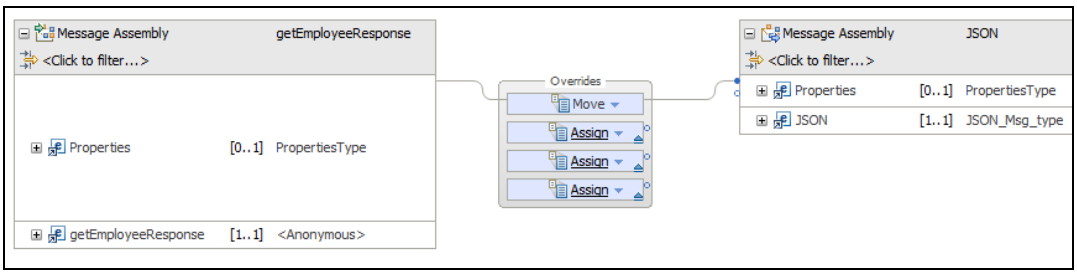
    - For the output message, expand IBM supplied message models, and select JSON.

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



12. The default map will be shown.

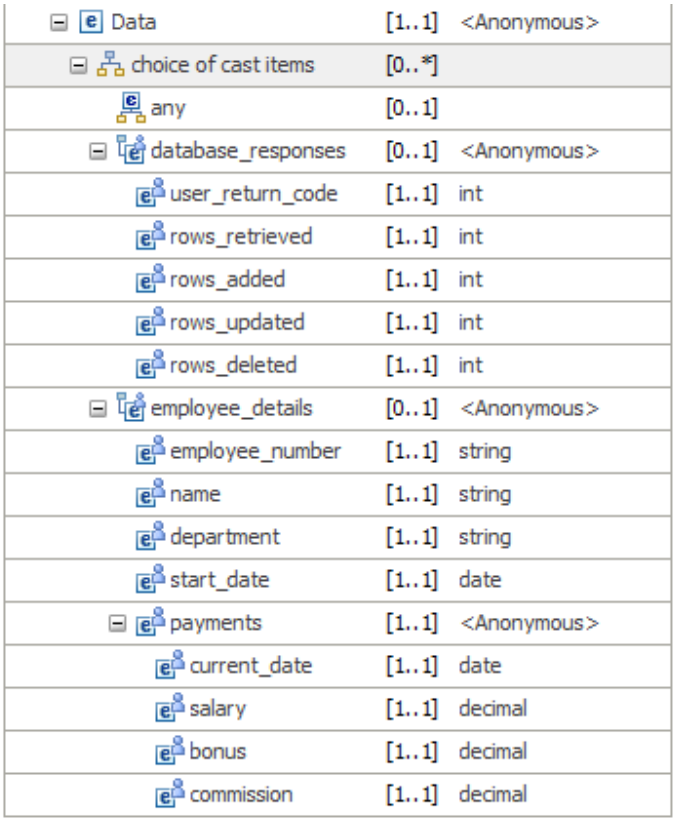
    The JSON domain does not require the use of MsgSet, MsgFormat and MsgType. The mapping editor initialises those values for you, so do not change the provided Assign transforms.



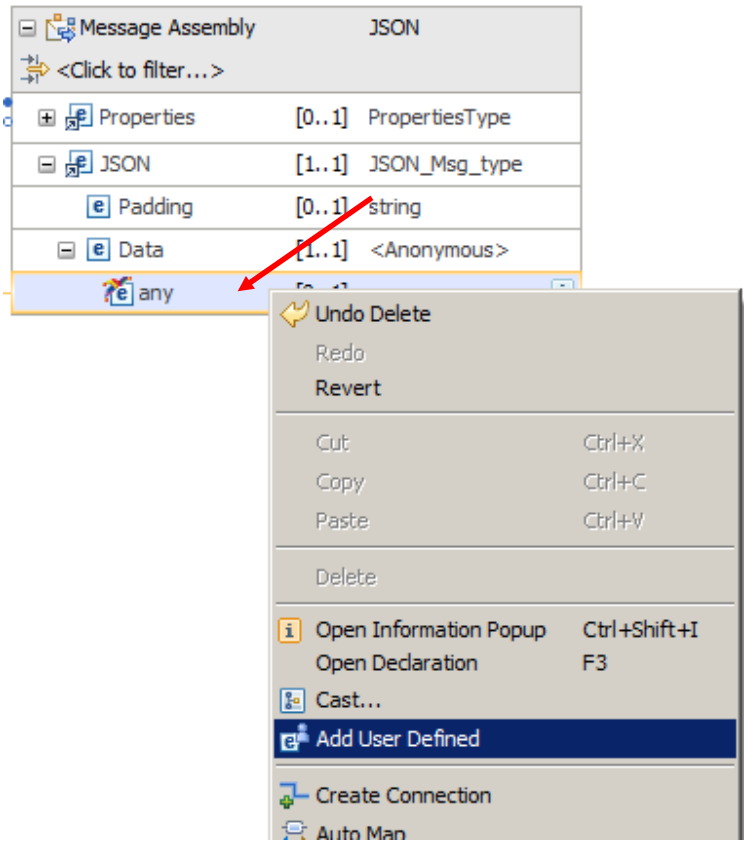
13. Before creating the map transformations, you are going to add a number of elements to the output message, using the JSON format, using the schemaless mapping tools in IIB v10.

    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.

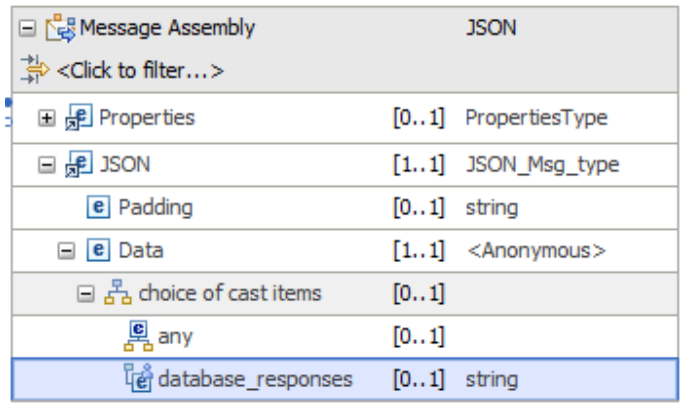
    When complete, the output message assembly will look like this:



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.



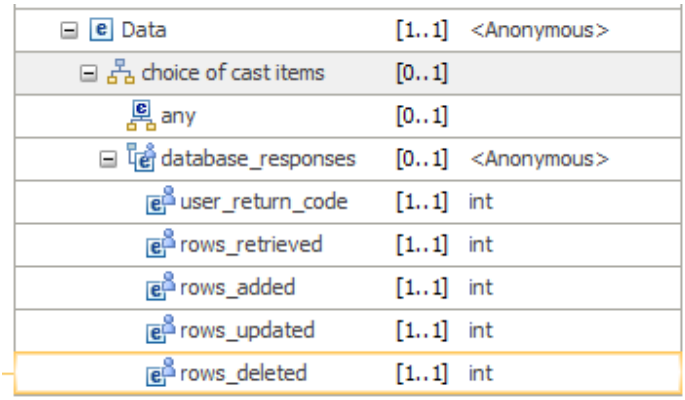
15. Overtype the name of the new element, naming it "database_responses". Leave the element type unchanged.



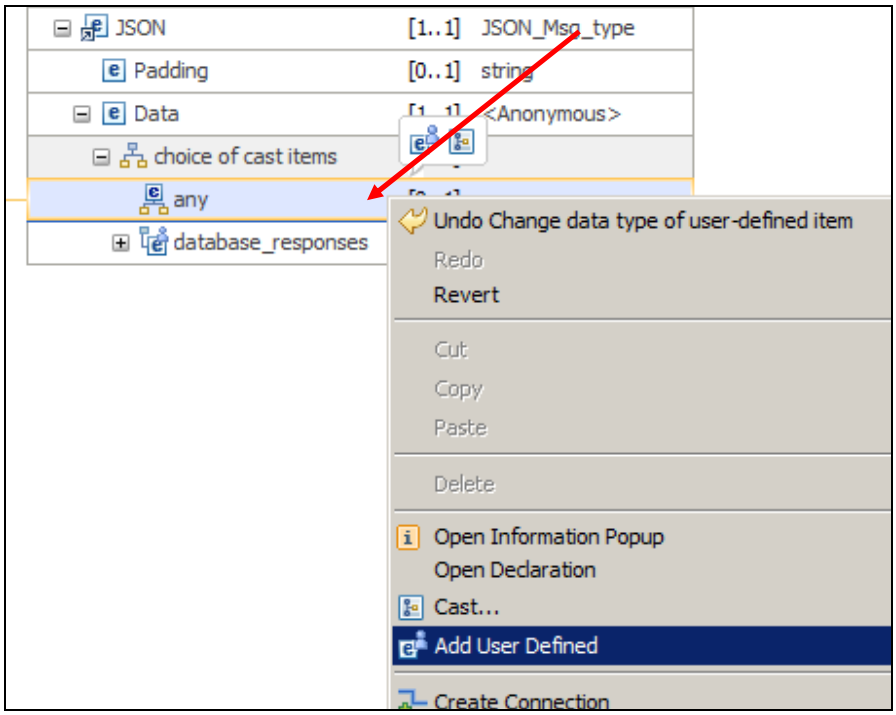
16. Now add some elements as child elements under database_responses. Do this by right-clicking database_responses for each new element, and select Add Child Element.

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

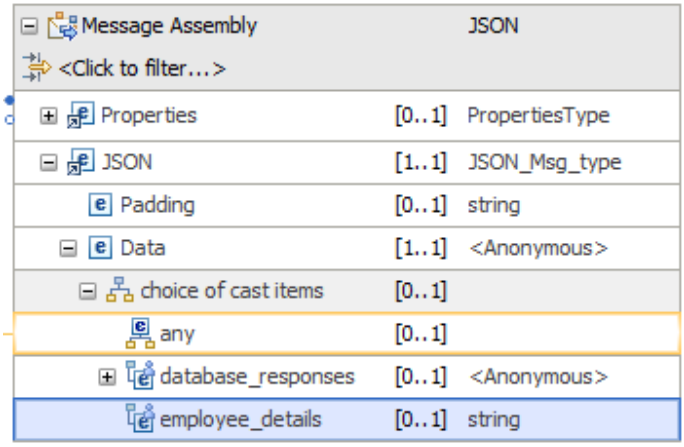
    - user_return_code
    - rows_retrieved
    - rows_added
    - rows_updated
    - rows_deleted



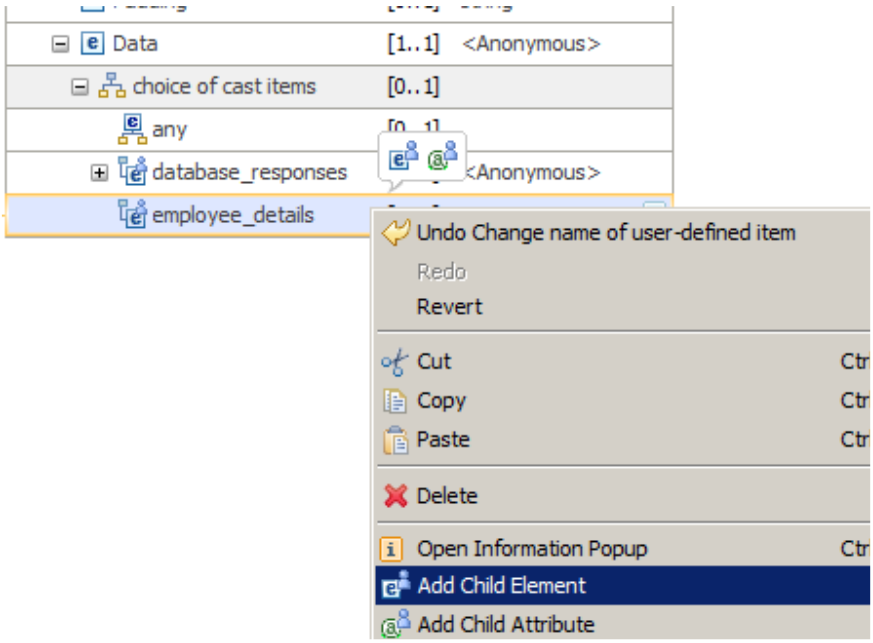
17. Now, add the elements required for the user data. Right-click the "any" element. Select "Add User Defined" from the context menu.



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



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

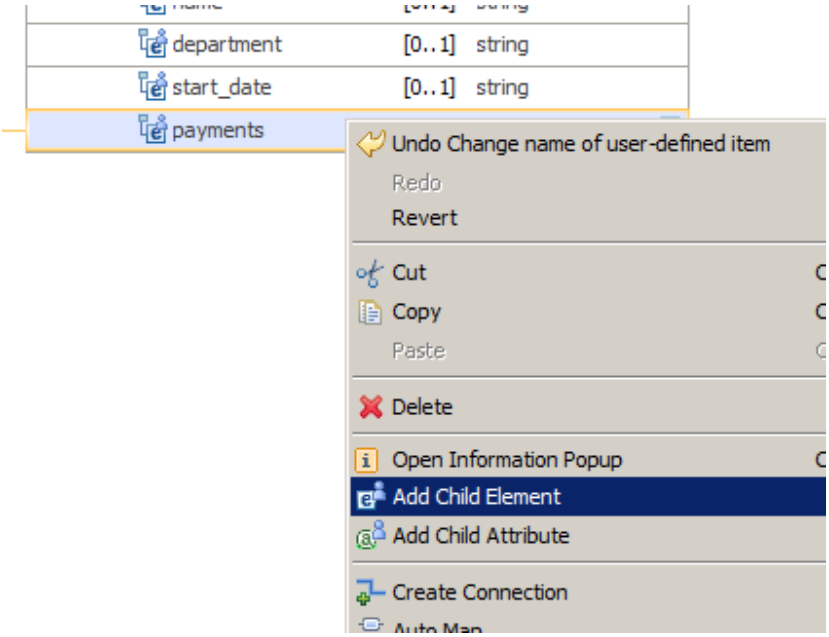
21. Add four more elements in the same way, by right-clicking the "employee_details" element (four times). Make sure these elements are all Child elements of employee_details.

    - name (type string)
    - department (type=string)
    - start_date (type=date)
    - payments (type=string)

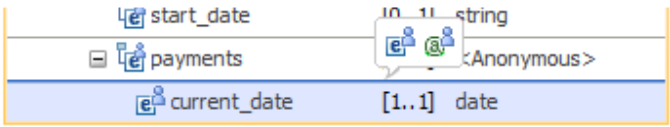
    At this point, the output message will look like this:

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.

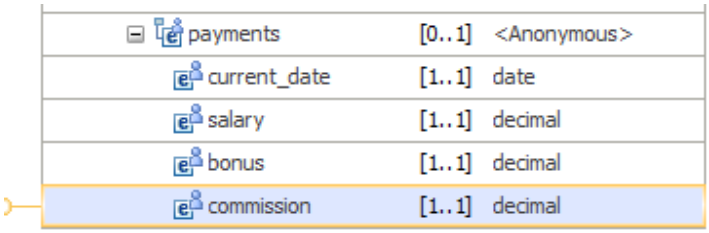


23. Name the new element "current_date", and set its type to "date".

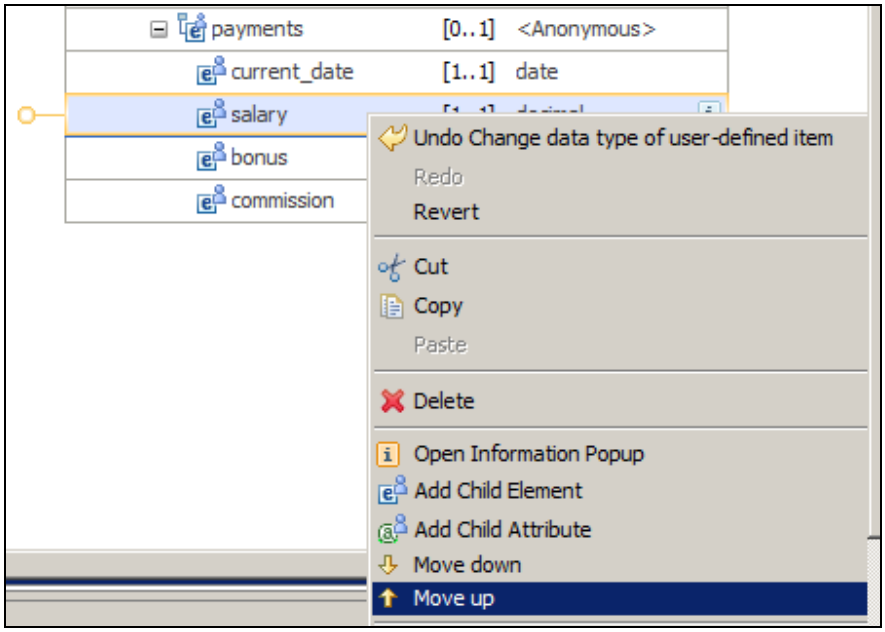


24. Add three further child elements under payments, all of type "decimal".

- salary
- bonus
- commission

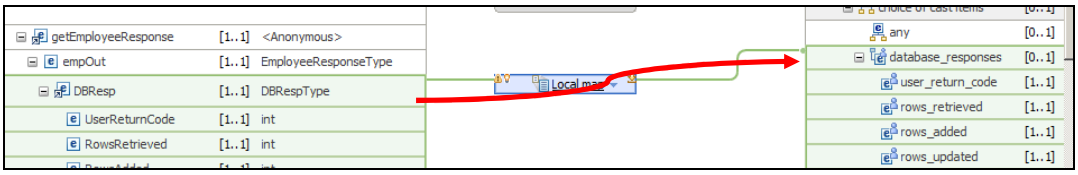


25. If you want to edit the name and position of the new elements, you can change the definitions you have just made. Clicking on the element name allows you to change its name. Right-clicking the elements gives you to tools to move the elements up or down in the output message.



2.1.1 Create the Mappings

1. First, map DBResp to database_responses.



This will initially create a Local map.

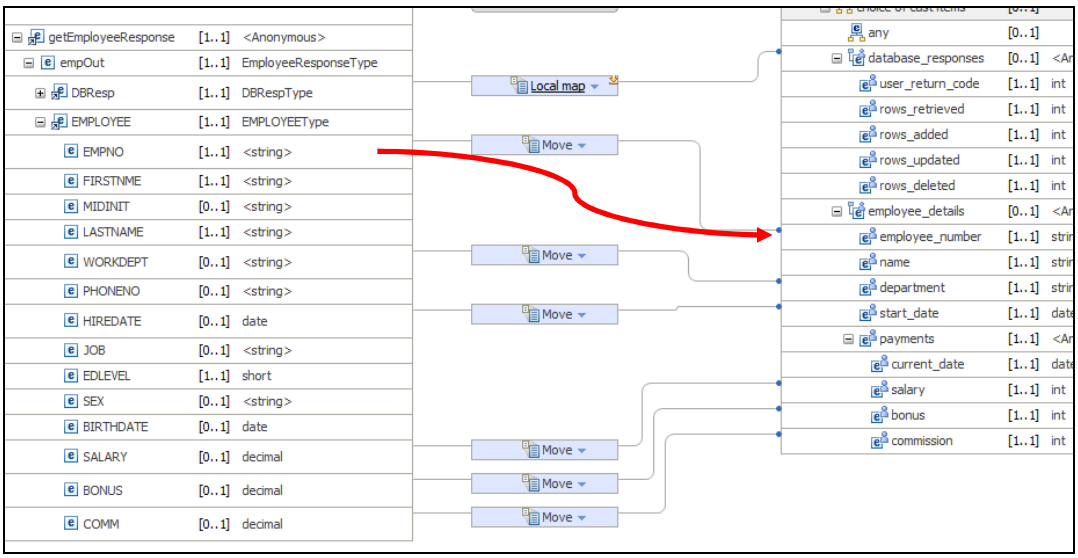
Click "Local map", and use the Auto Map function to map all the elements in DBResp (use the auto-map icon on the icon row in the editor).

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:



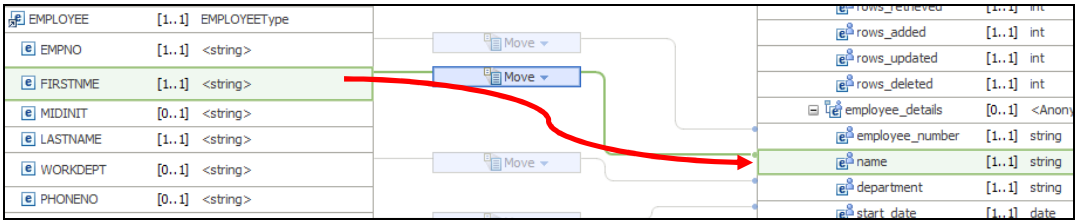
2.1.2 Optional extra mapping (Concatenate, Custom XPath, Retrieve flow details)

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.

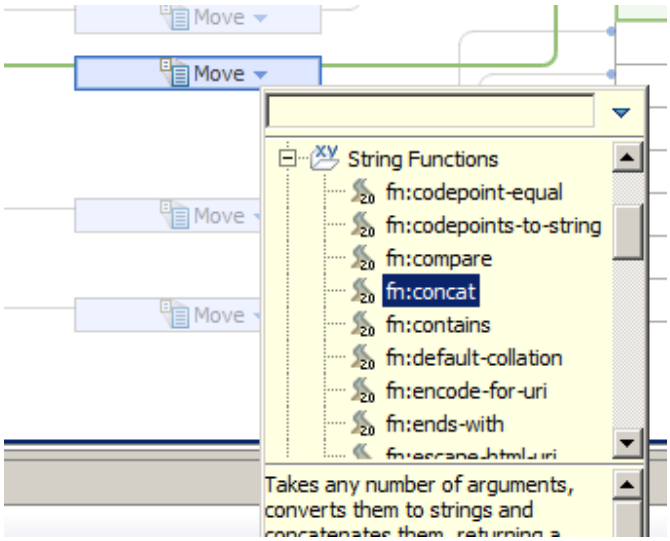
You may proceed direct to Complete the Message Flow on page 33.

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

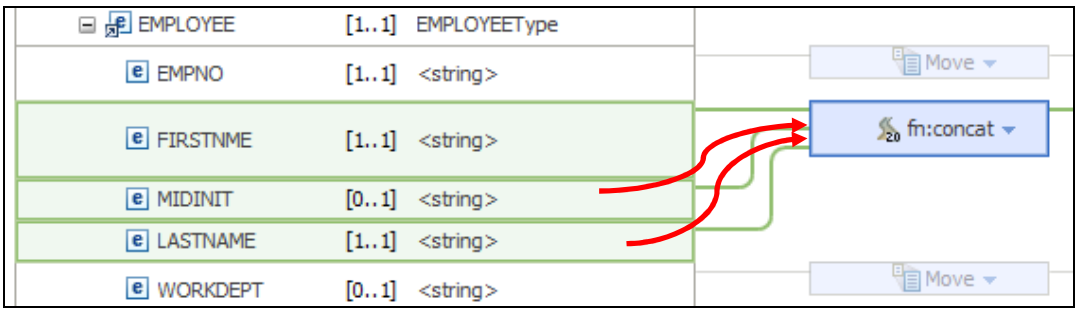
   Drag the input element FIRSTNME to the output element name.



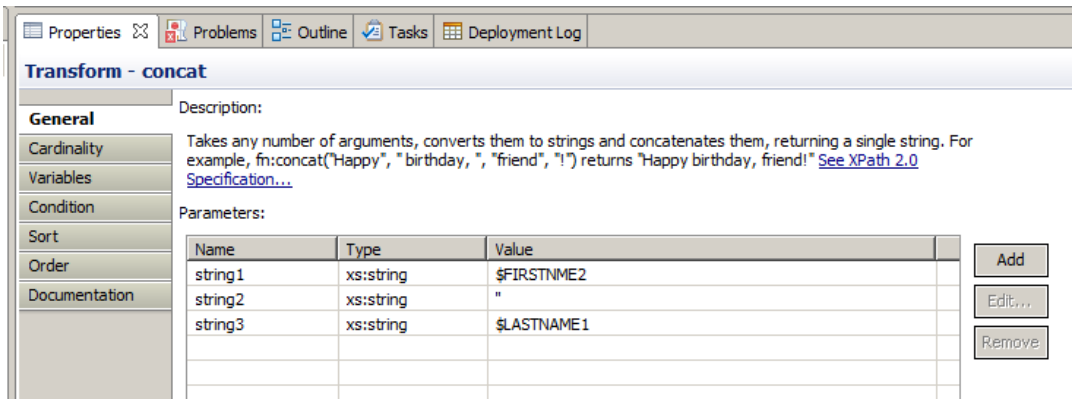
2. This will generate a Move transform. Using the drop-down arrow, change this to "fn:concat" (in the string Functions folder).



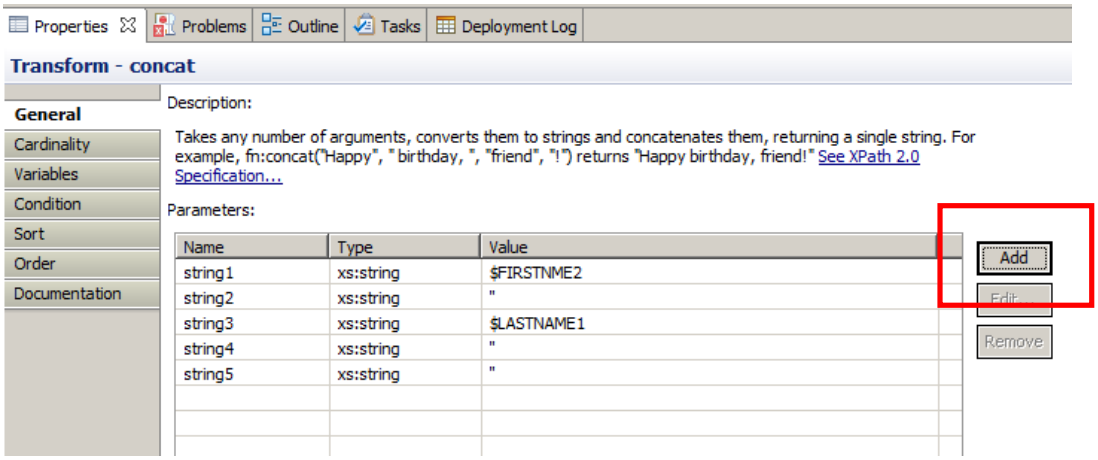
3. Connect the elements MIDINT and LASTNAME to the concat function box.



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



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



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

Parameters:

Name	Type	Value
string1	xs:string	$FIRSTNME
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	$FIRSTNME
string2	xs:string	"
string3	xs:string	$MIDINIT
string4	xs:string	"
string5	xs:string	$LASTNAME

Add
Edit...
Remove

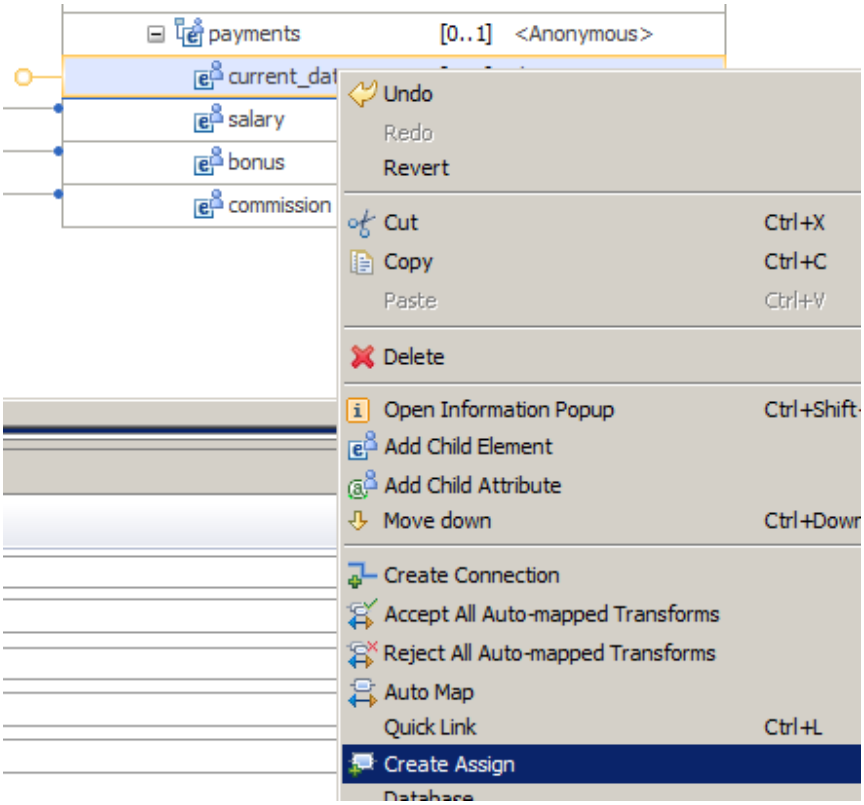
6. Change the values of string2 and string4 to add a space between the two apostrophes.

(Highlight each line and click Edit).

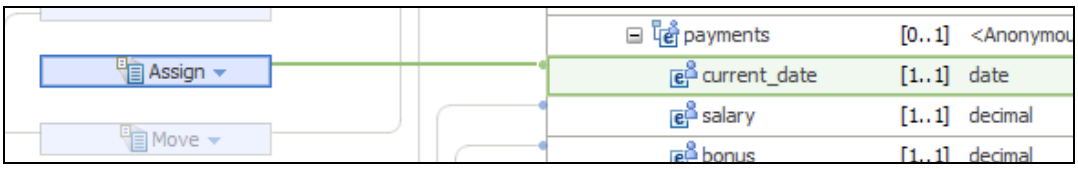
2.1.3 Custom XPath

Now use XPATH to set a value for the current_date element.

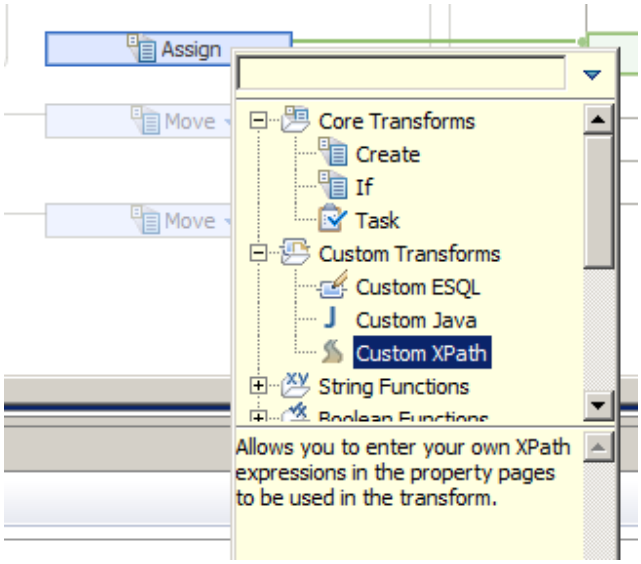
1. Under payments, right-click current_date, and select Create Assign.



An Assign function will be created.



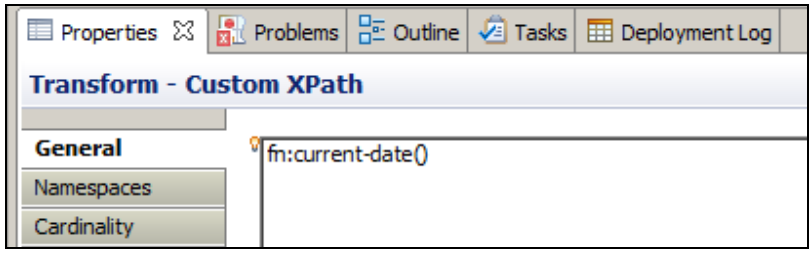
2. Using the drop-down arrow on the Assign transform, change this to a Custom XPath function (in the Custom Transforms folder).



3. 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).



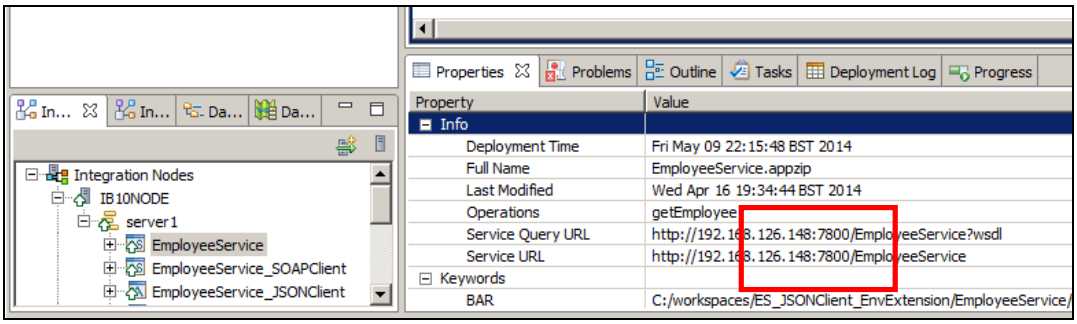
4. The second map is now complete, so save it and close the map editor.

2.2 Complete the Message Flow

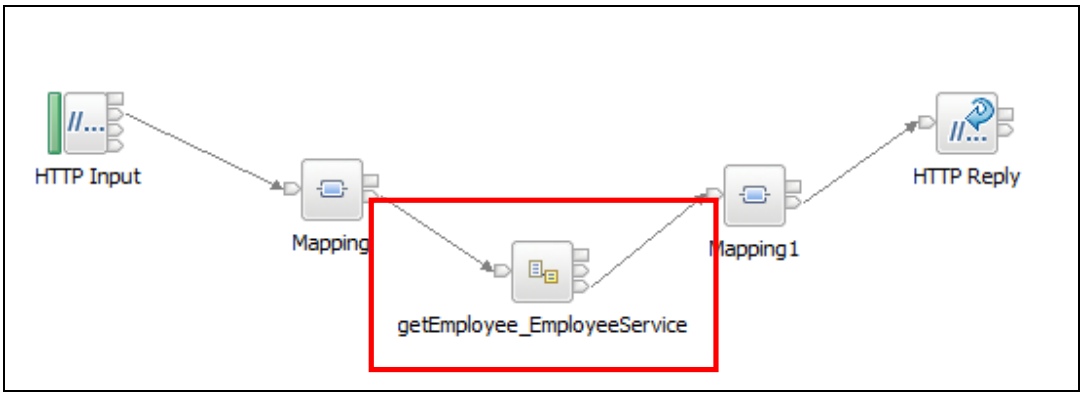
1. Two final changes are necessary. First, you must make sure that the new application is configured to use the correct port number of the EmployeeService.

   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.



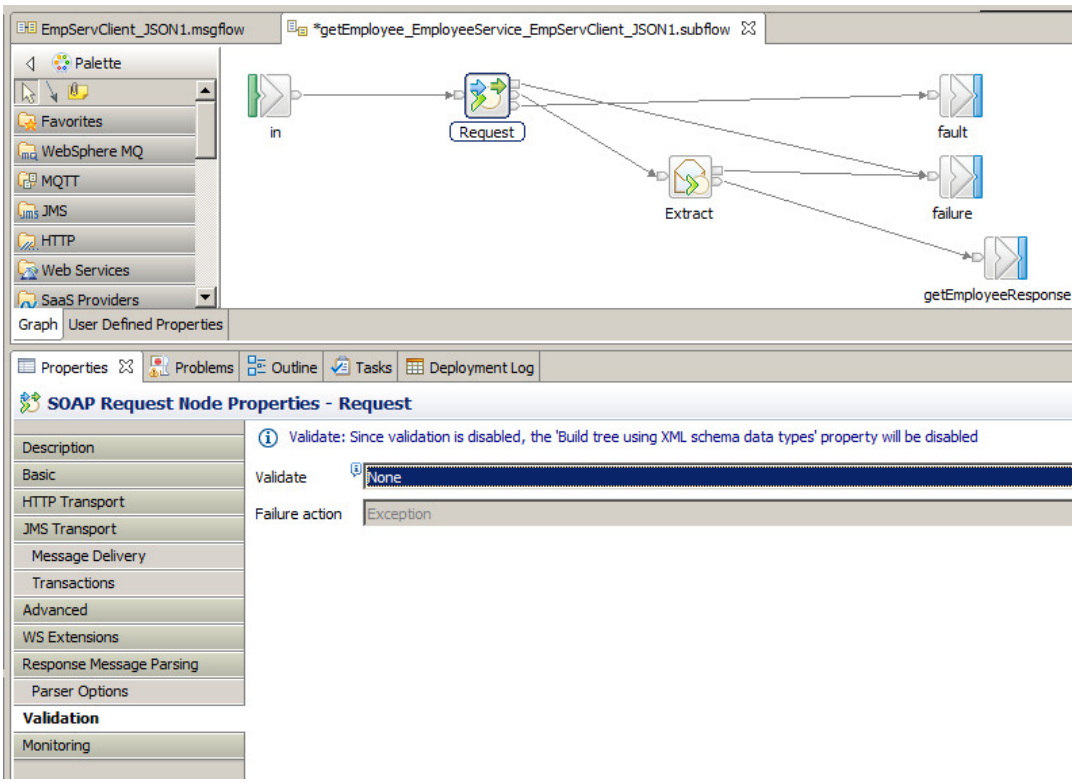
2. In the primary message flow, double-click the generated subflow getEmployee_EmployeeService.



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.

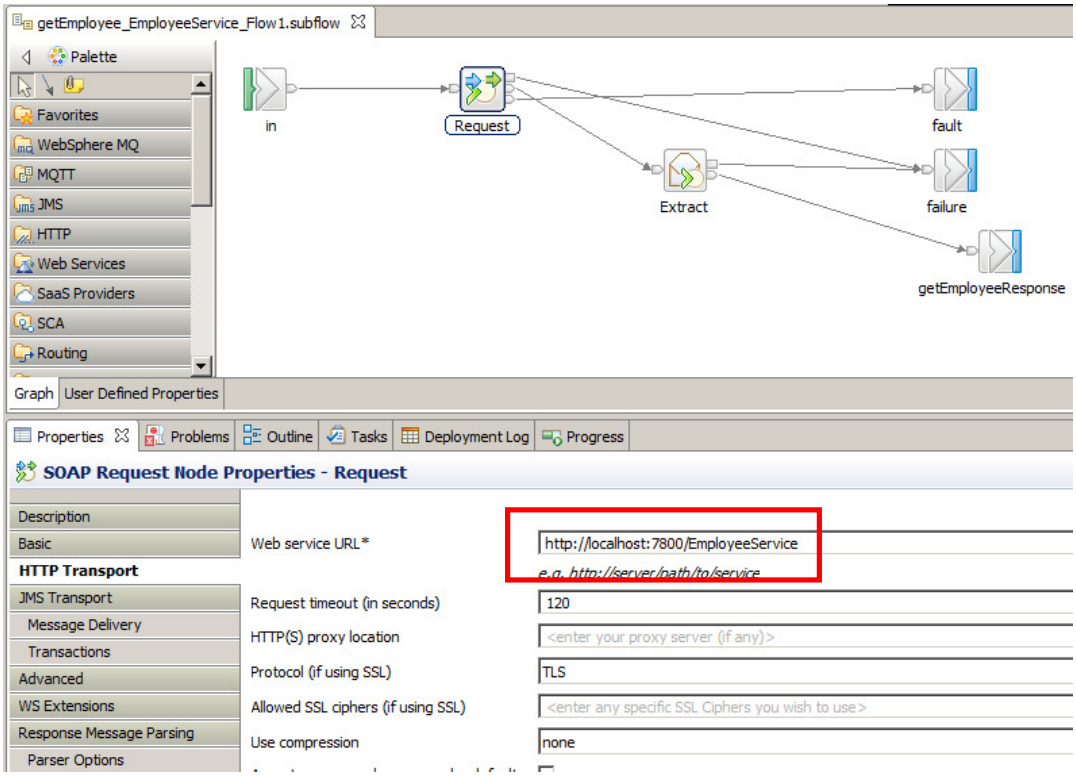
   Highlight the SOAP Request node, and in the node properties select the Validation tab. Set the Validate value to None.



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).

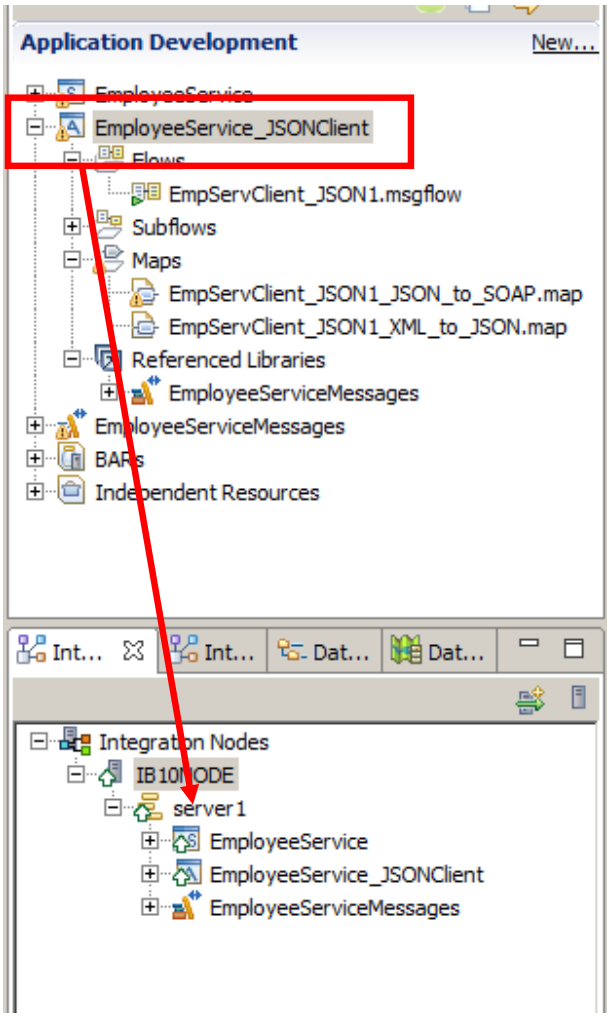
   (Note - a production environment would not normally hard-code a port number in the application, but this is done for ease of development in this lab).

   Save and close the subflow.



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

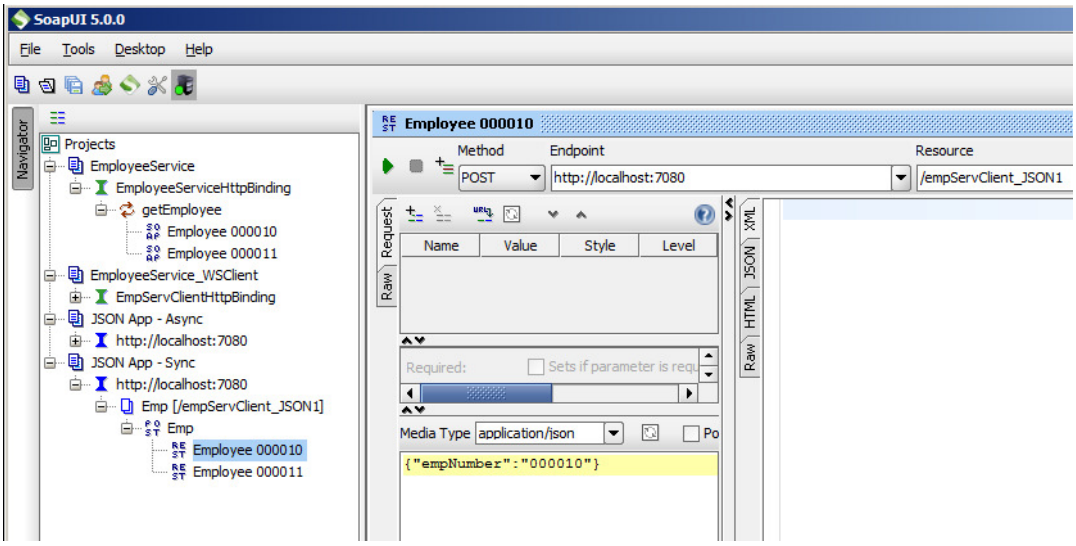
1. Open SOAPUI (you may already have this open from a previous lab).

   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:

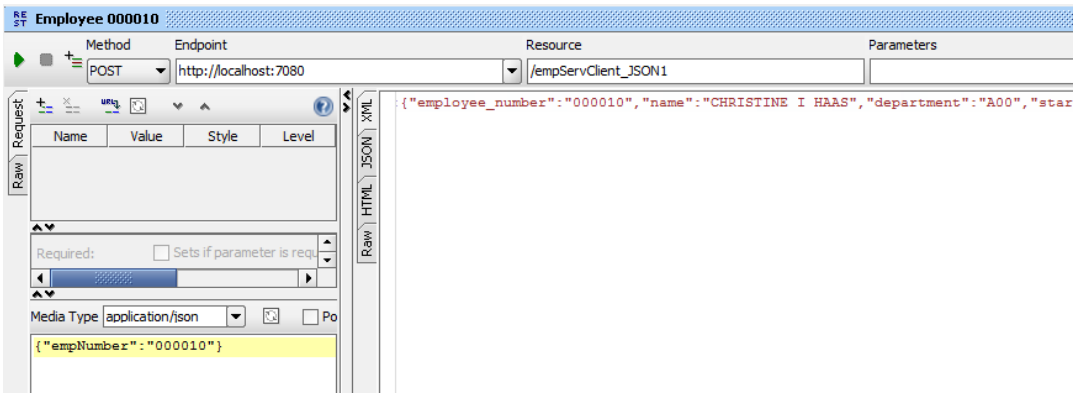
   {"empNumber":"000010"}

   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.

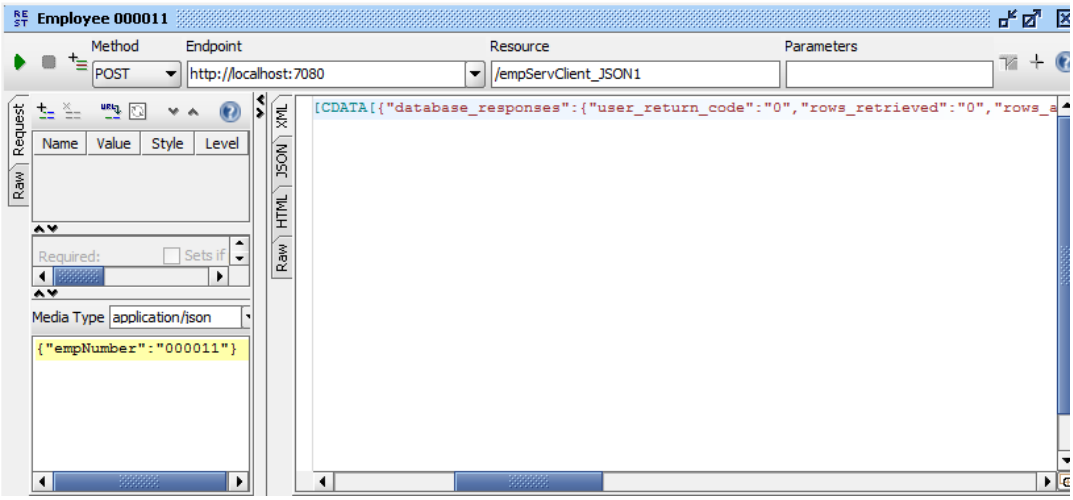


   (For info, this SOAPUI project was created by specifying "New REST Project" in the SOAPUI dialogue).

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.



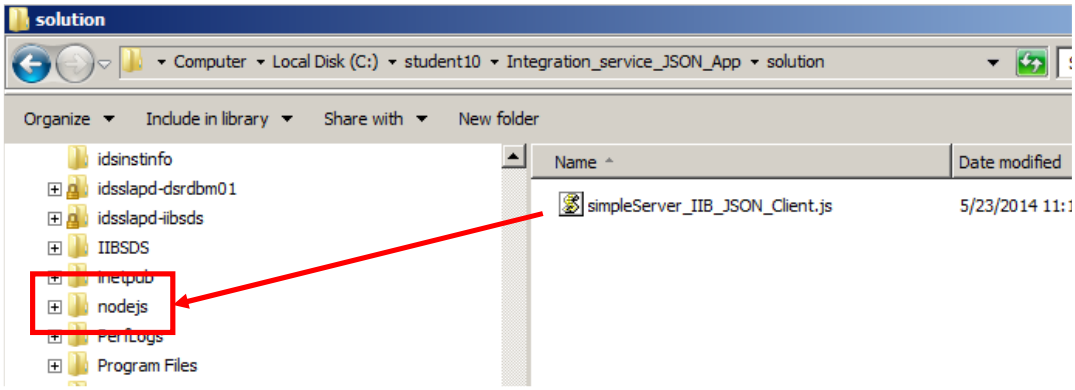
3.2 Test using Node.js Web Browser application (Optional)

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.



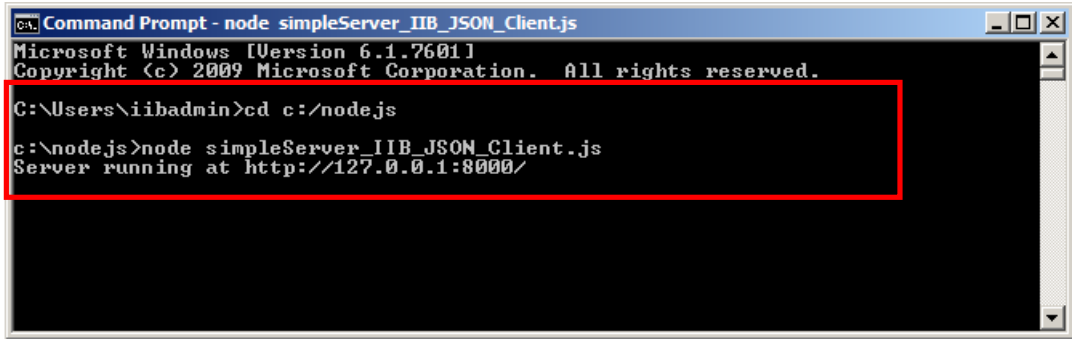
2. In a Windows Command Prompt, do the following:

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

```
node simpleServer_IIB_JSON_Client.js
```

   - View the message on the bottom of the Command Console:

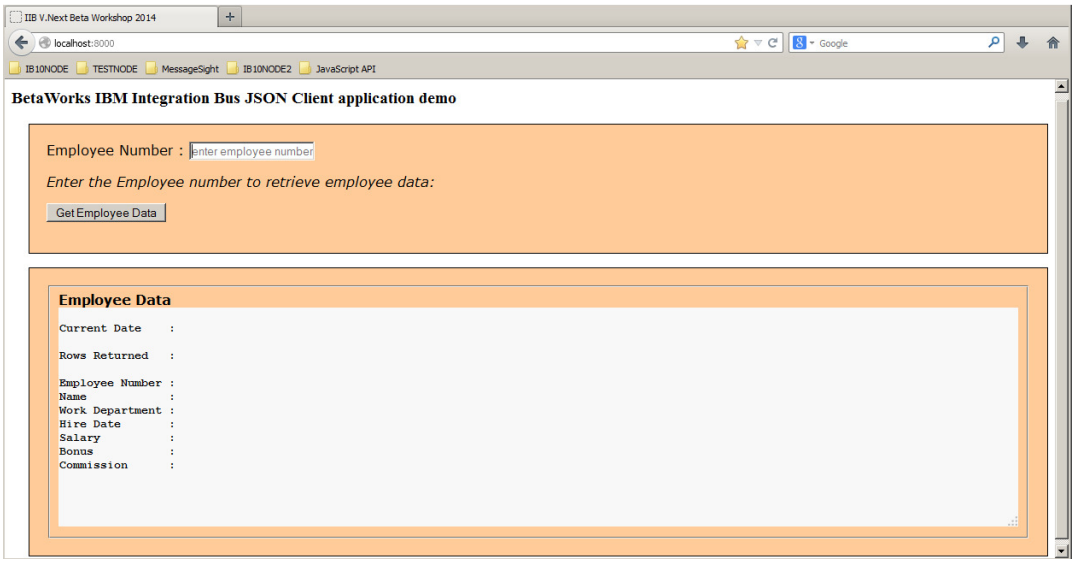
```
Server running at http://127.0.0.1:8000/
```



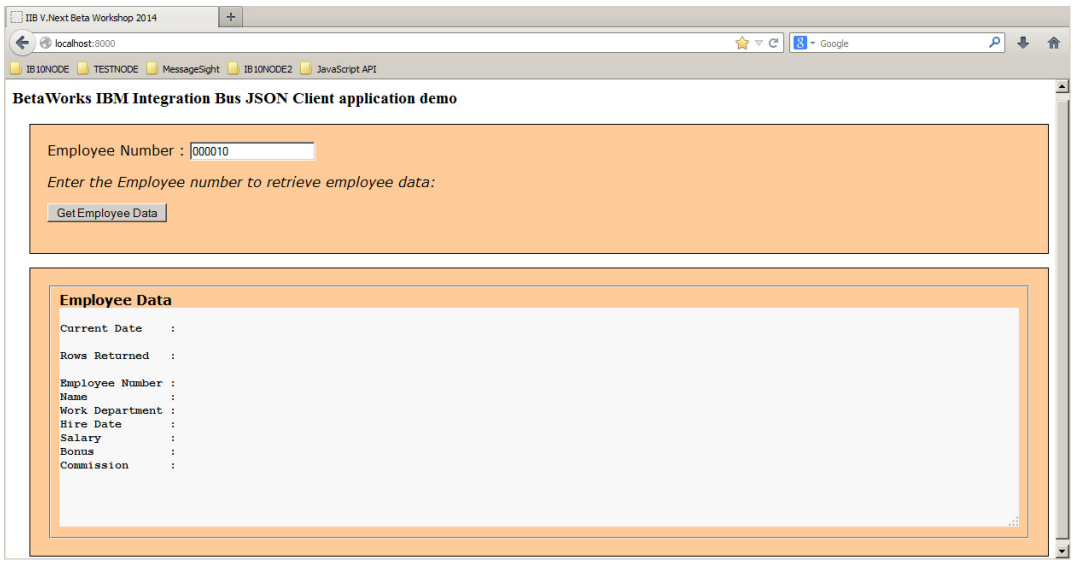
   - **Important: Please do not close the Command Prompt, as this will stop the server. You can however minimize the window.**

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

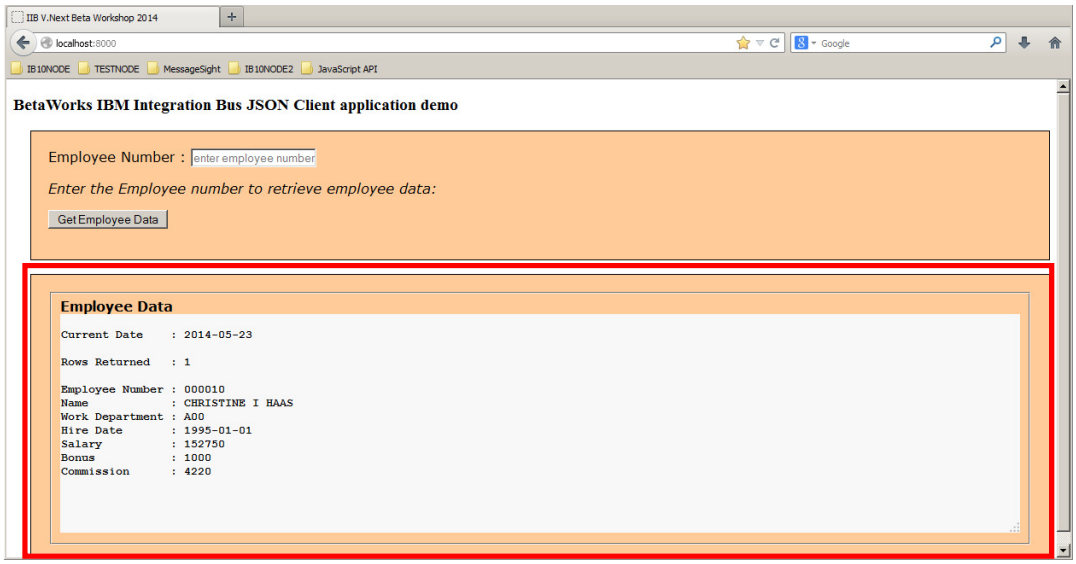
   This will open the Web page of the **'EmpServClient_JSON1'** Web application:



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



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



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)

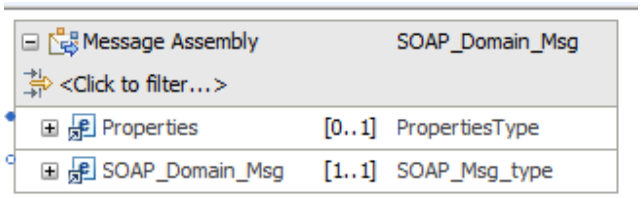
This final chapter shows you how to use the schemaless mapping tools in the Graphical Data Mapper to write data to the Environment tree.

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.

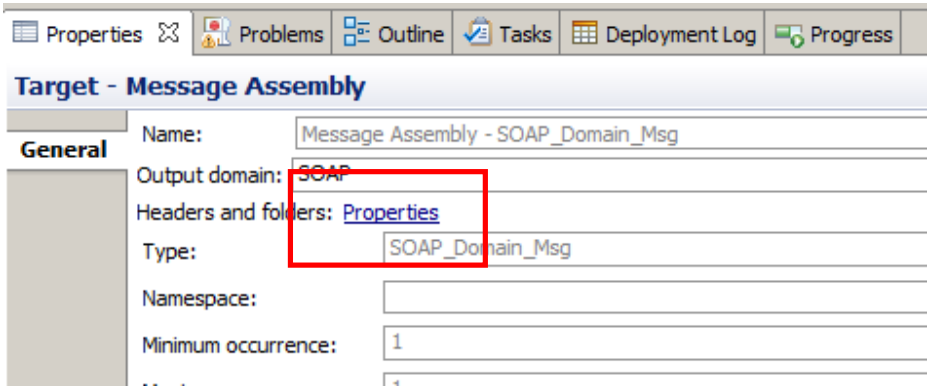
4.1 Configure the first mapping node

1. Reopen the first Mapping node, JSON_to_SOAP, and focus on the output message assembly.

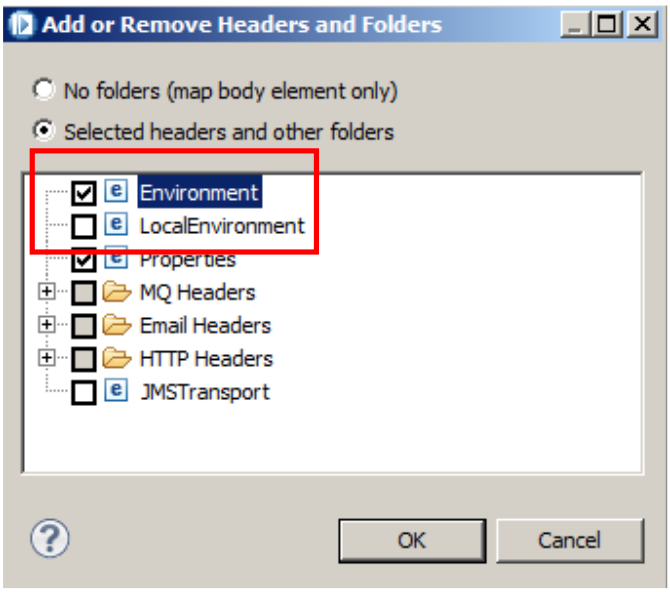


2. Highlight (click on) the output message assembly, and look at the Properties of the assembly.

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

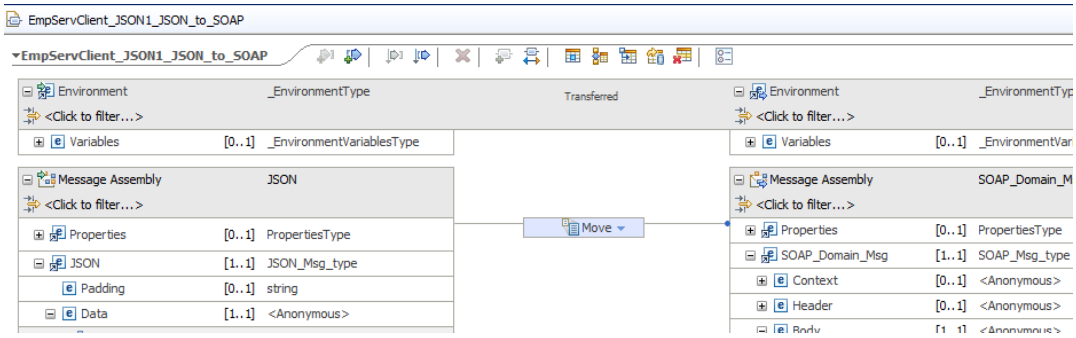


3. In the dialogue window, select the Environment, and click OK.

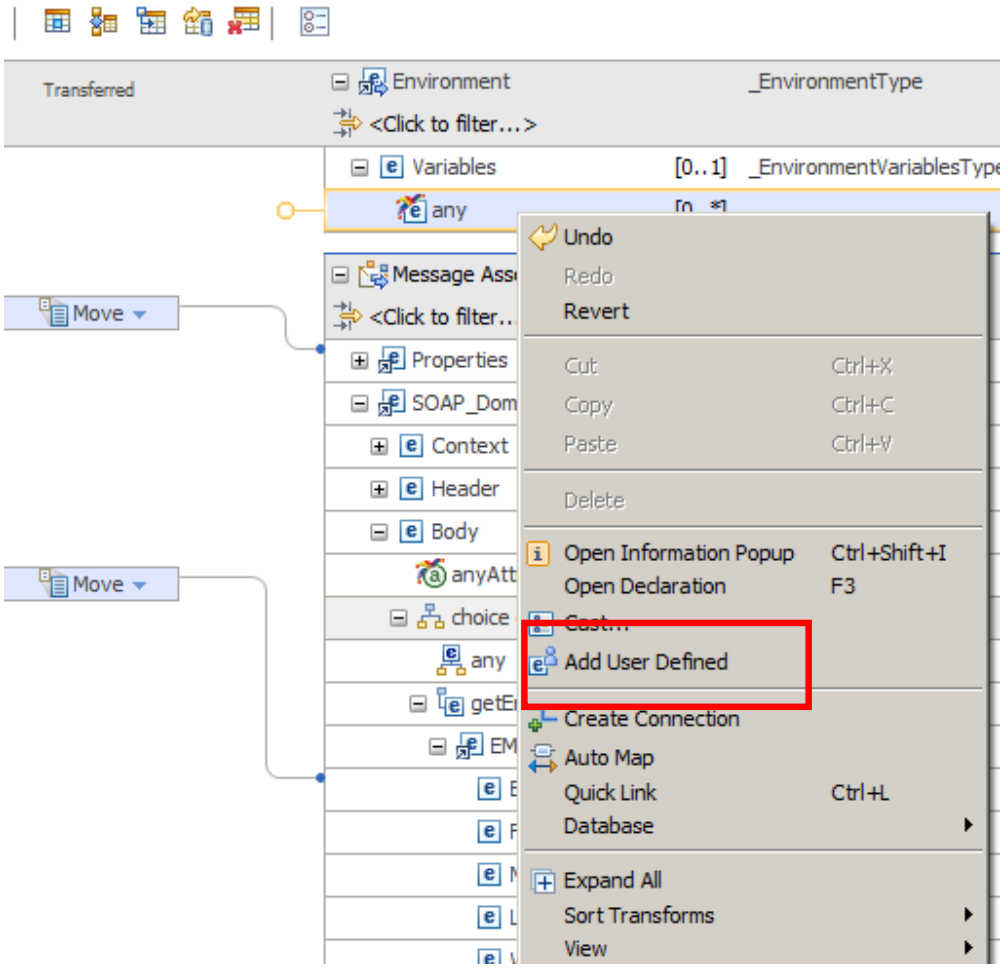


4. The output message assembly will have been updated to show the Environment.

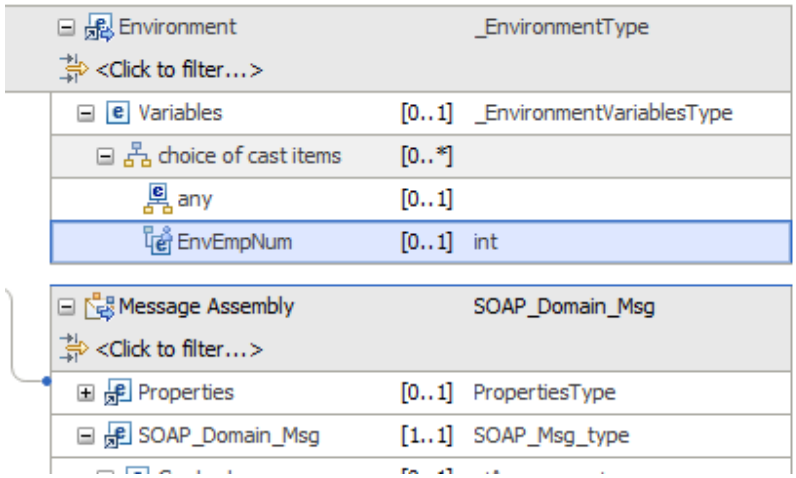
   Note that because the Environment is a common area, this is displayed at the top of the map, bridging the input and output assemblies.



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



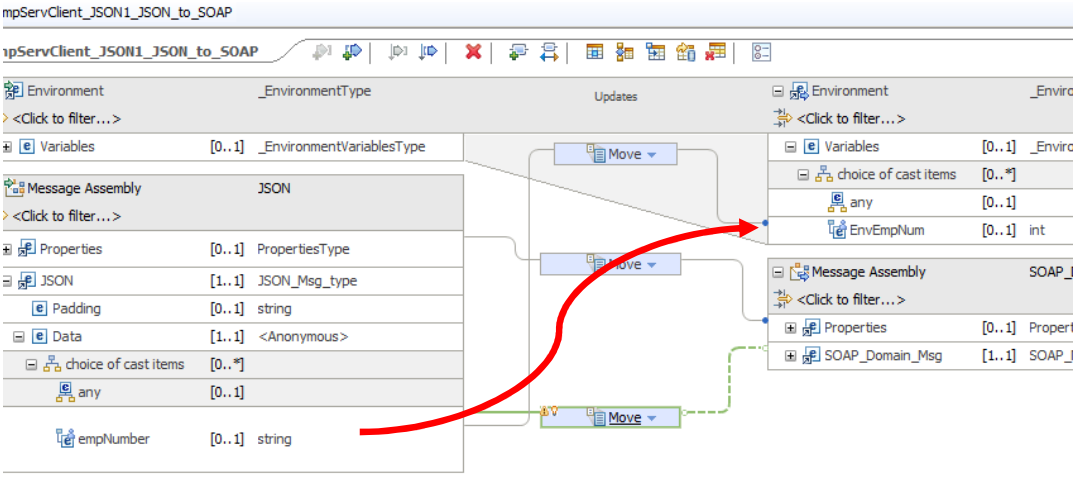
6. Name the new element EnvEmpNum and set the type to "int".



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.



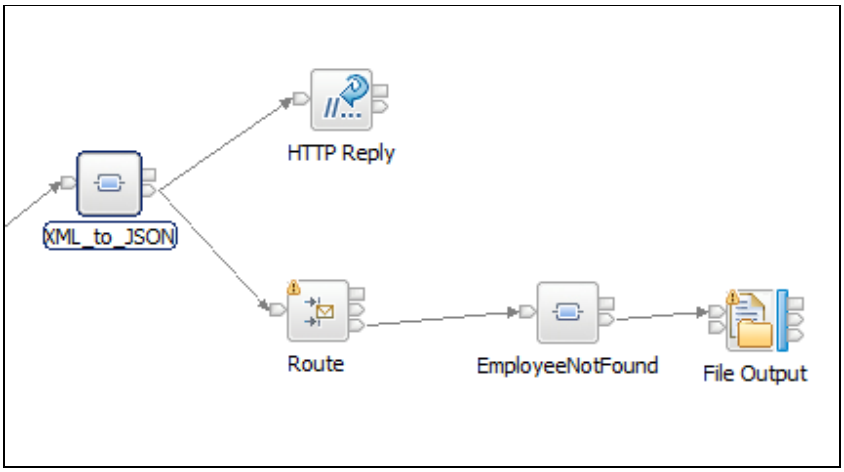
   This map will now save the input employee number to the Environment tree for use later in the flow.

   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



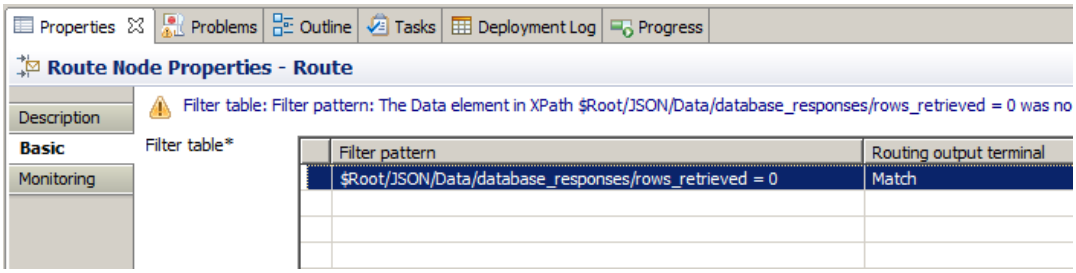
4.2.1 Configure the Route node

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.

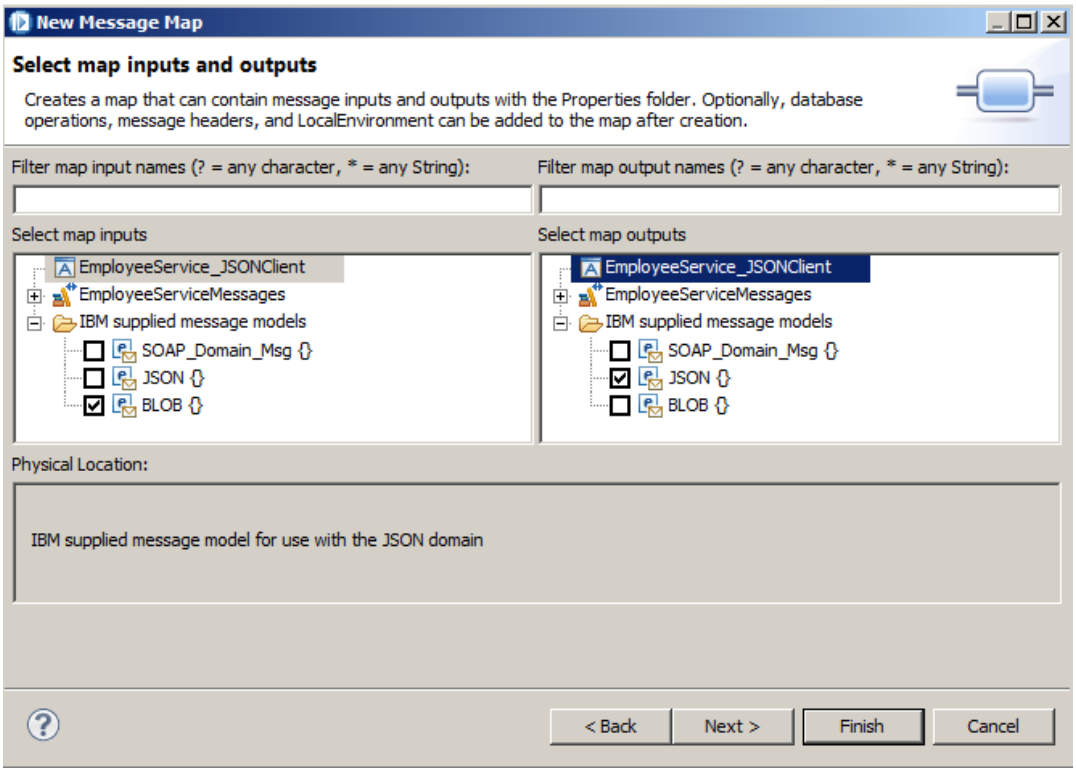


4.2.2 Configure the new Mapping node

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

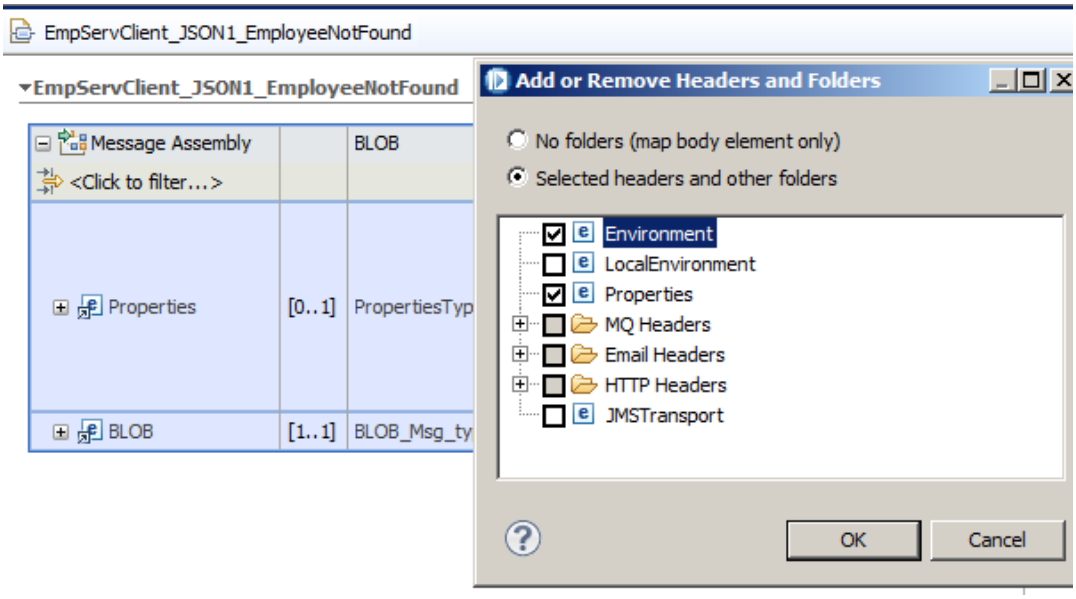
   Set Input message assembly = BLOB
   Set Output message assembly = JSON

   Click Finish.

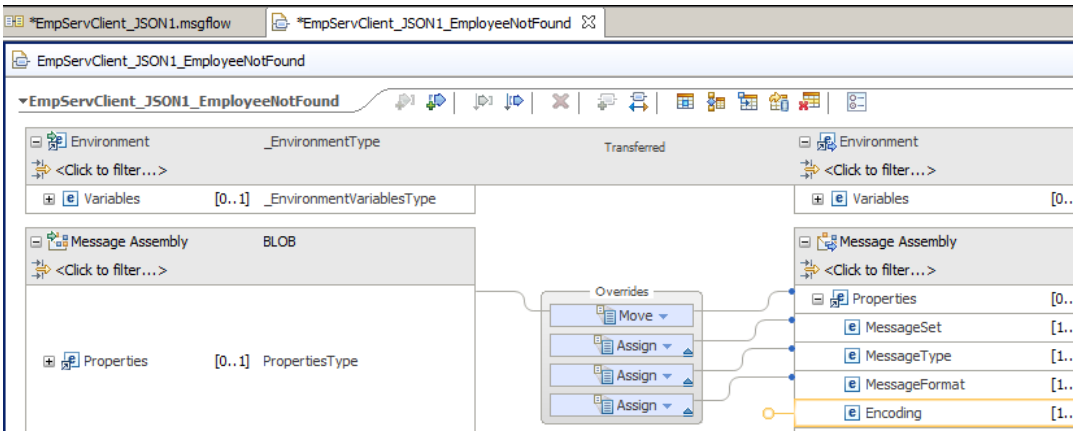


2. In the map editor, focus on the input message assembly.

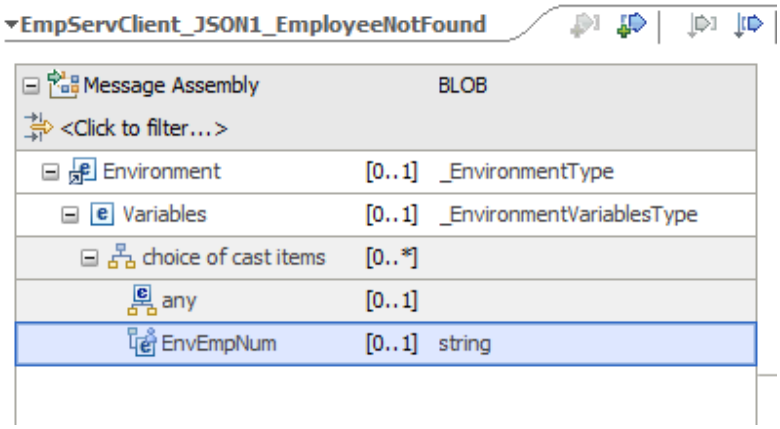
   Add the Environment to the available Folders.



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

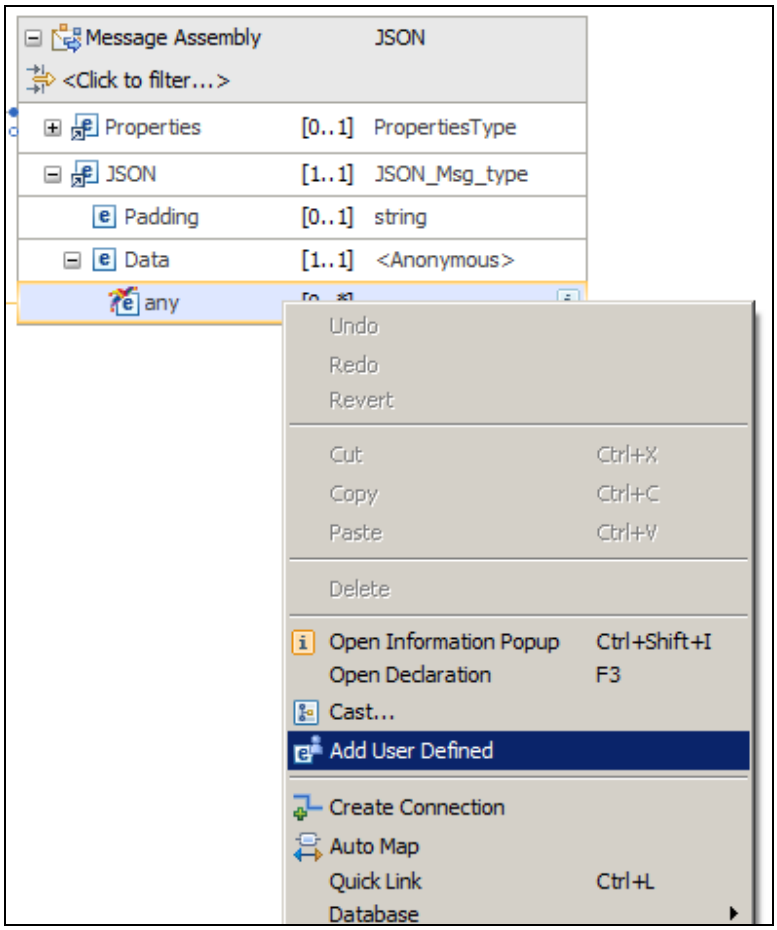


4. Using the same technique as earlier, on the input message, right-click "any" and add a User Defined element, named EnvEmpNum (ie. the one you added earlier).



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:

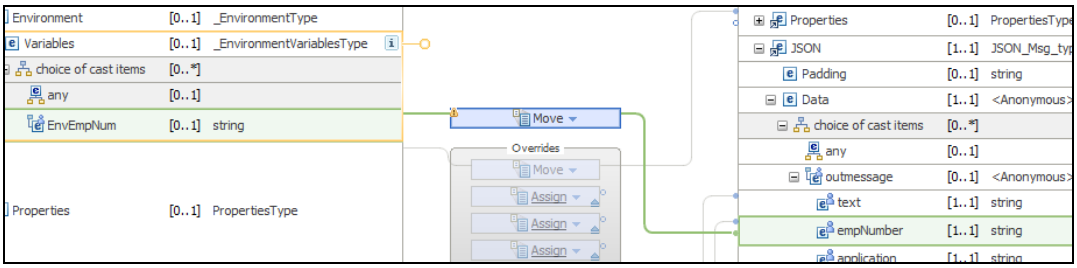
Message Assembly		JSON
<Click to filter...>		
Properties	[0..1]	PropertiesType
JSON	[1..1]	JSON_Msg_type
Padding	[0..1]	string
Data	[1..1]	<Anonymous>
choice of cast items	[0..*]	
any	[0..1]	
outmessage	[0..1]	<Anonymous>
text	[1..1]	string
empNumber	[1..1]	string
application	[1..1]	string
flowname	[1..1]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



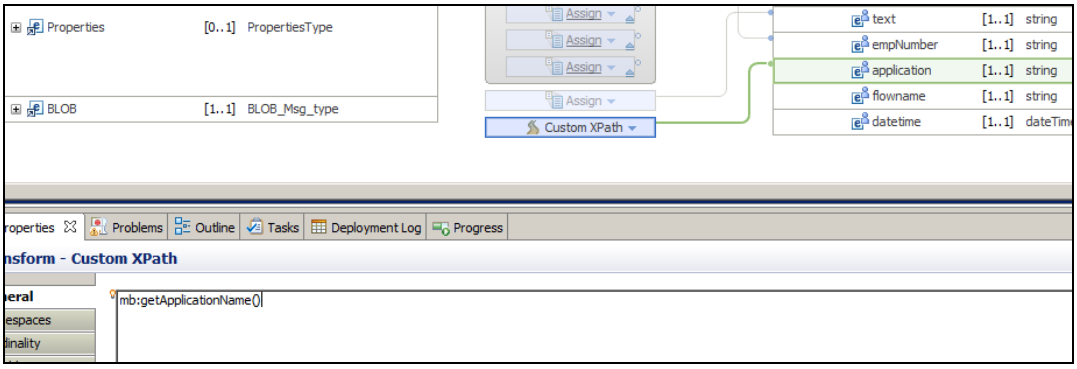
3. For the output "application". right-click "application" and select "Create Assign".

   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).



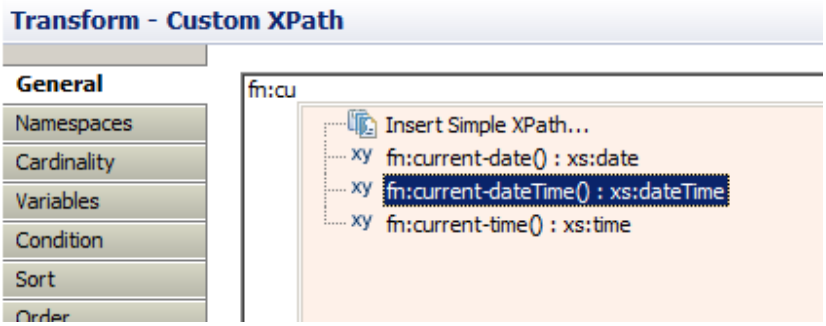
4. For the output "flowname", create a similar transform to application Set the XPath statement to

   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()

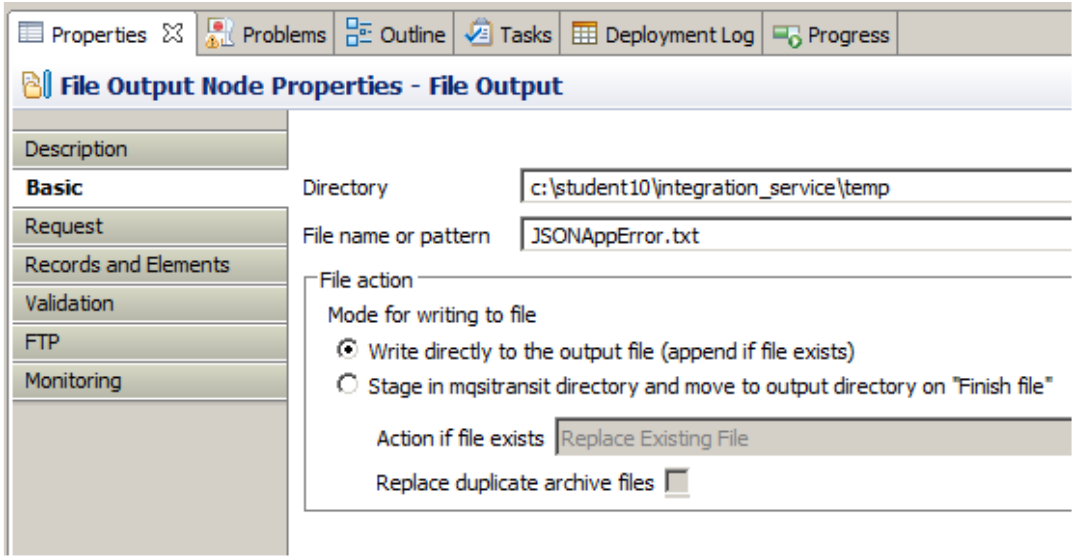


Save and close the map.

4.2.4 Configure the File Output node

1. On the node properties Basic tab, set:

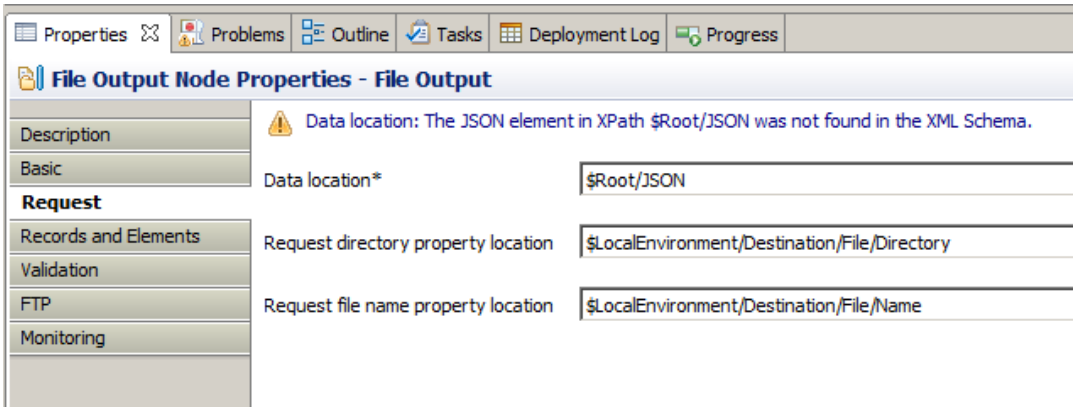
   - Directory = c:\student10\integration_service\temp
   - File name = JSONAppError.txt
   - File action = Write directly to output file



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.



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.

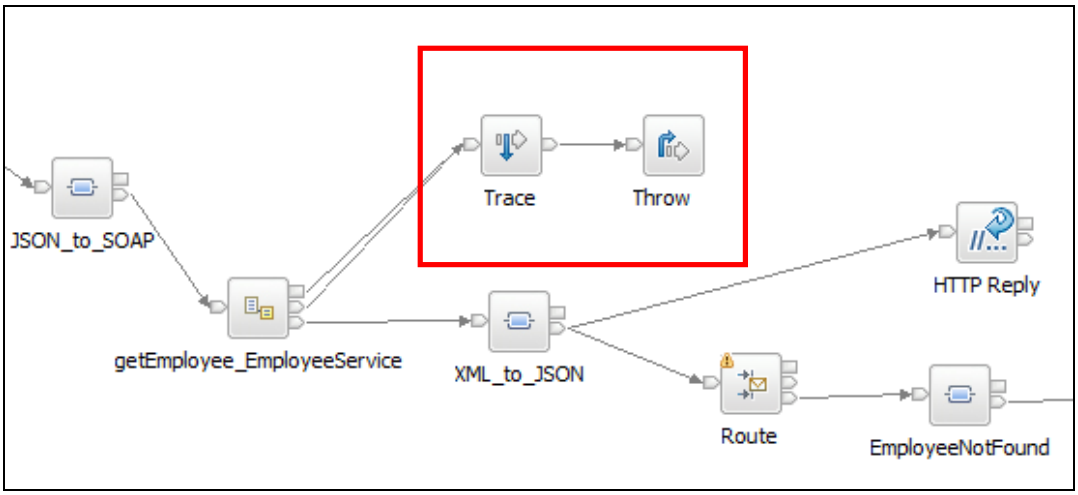
5. Add fault and failure handing (optional)

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.

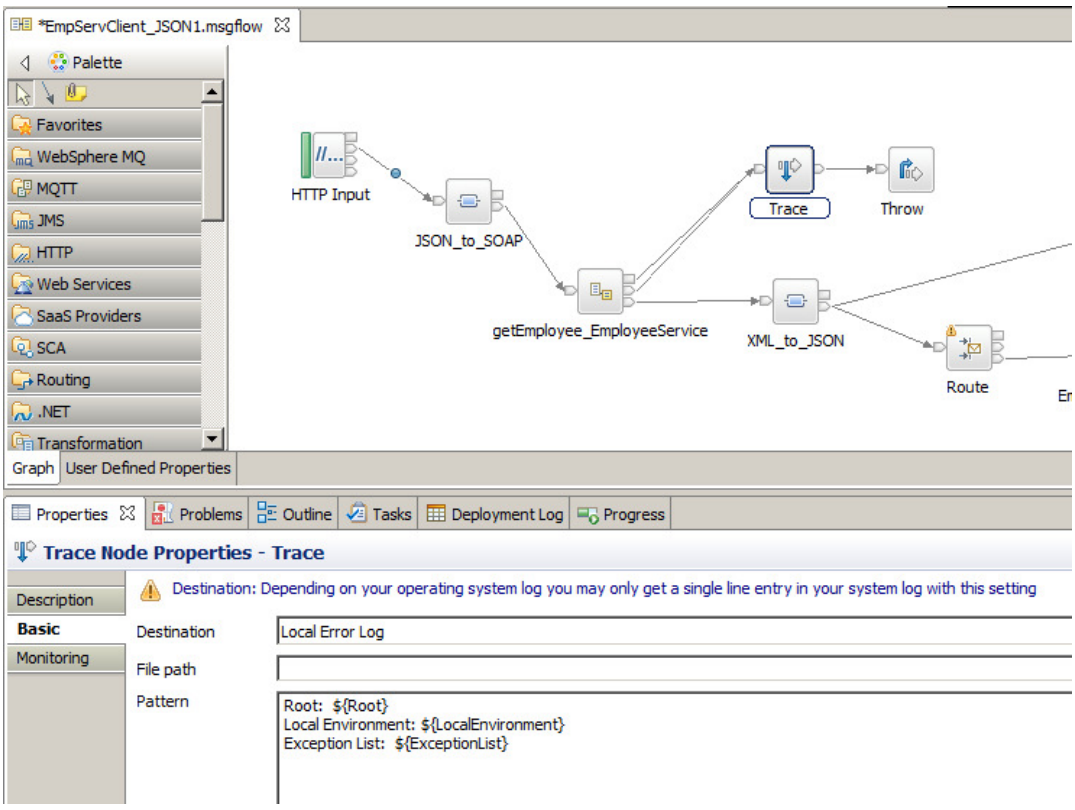
   Connect both the Failure and Fault terminals of the getEmployee_EmployeeService subflow to the Trace node.



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}



3. 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).

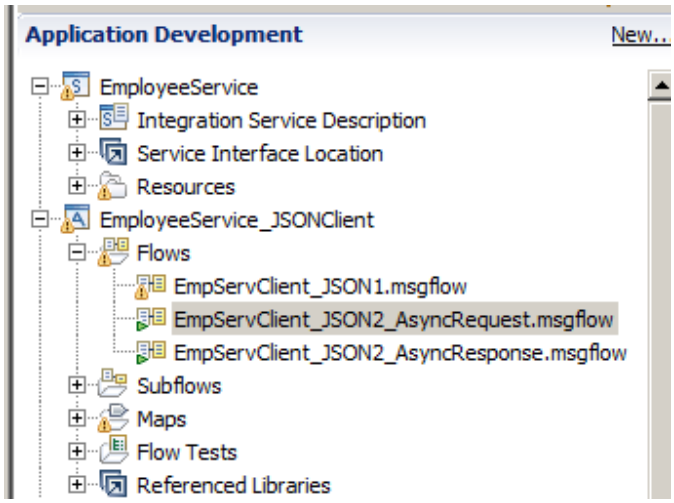
At execution time, the flow invokes the service synchronously. If the service is not available, then the JSON1 application will throw an error, and the client will not receive the requested data.

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.

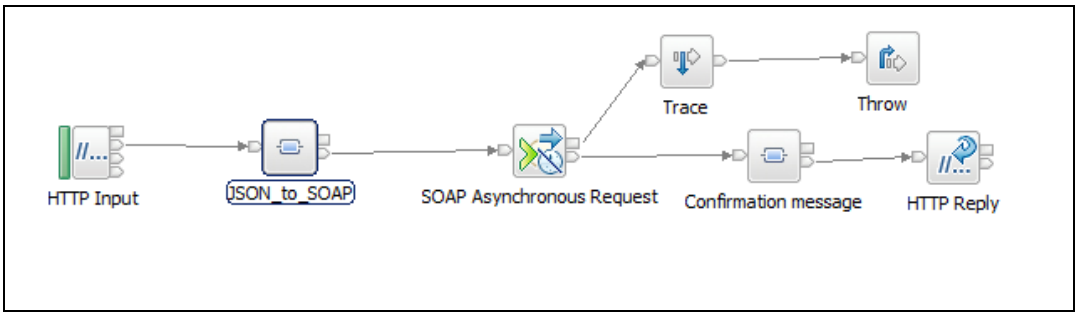
Since you are now an experienced IIB developer, we will omit some of the detailed screen captures.

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

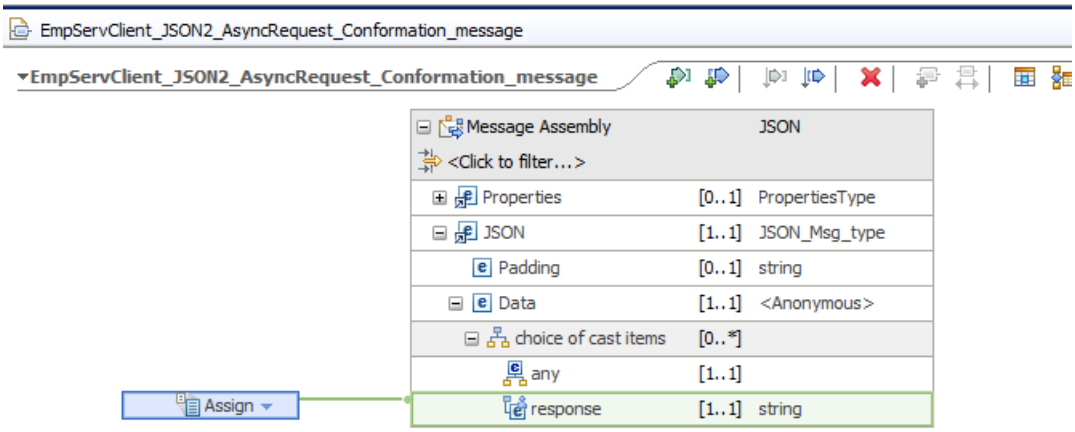
3. Configure the new mapping node: Confirmation Message

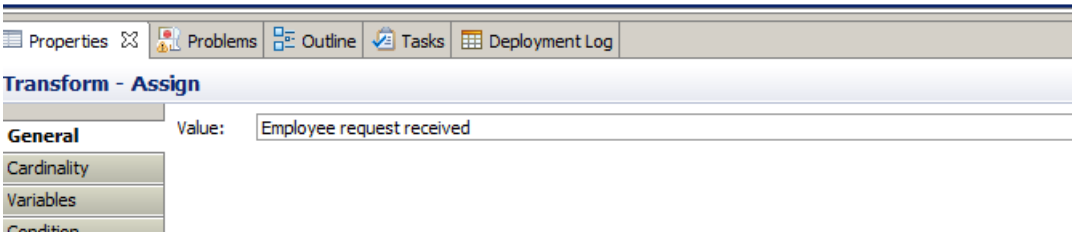
   Open this new map and select just an output message, of type JSON.

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

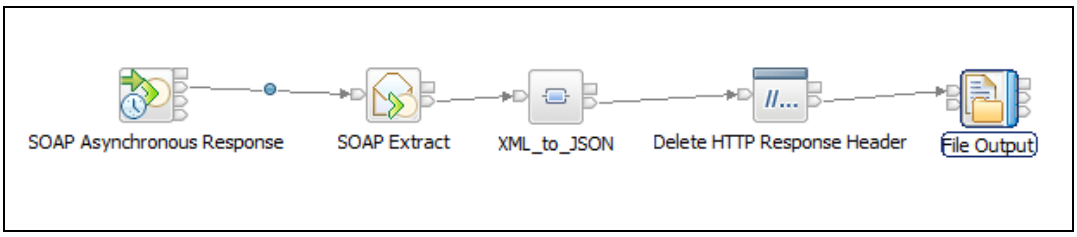
   Assign a value to this element using an Assign transform. Set the value to "Employee request received".

   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 identifer = 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.

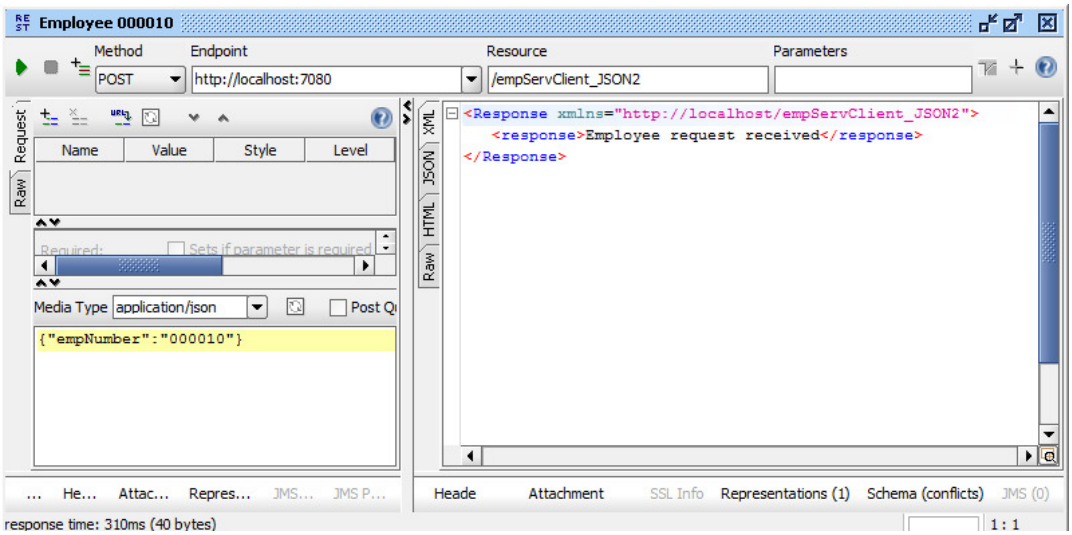
6. Test the new flows with SOAPUI.

   In SOAPUI, expand the project JSON App - Async.

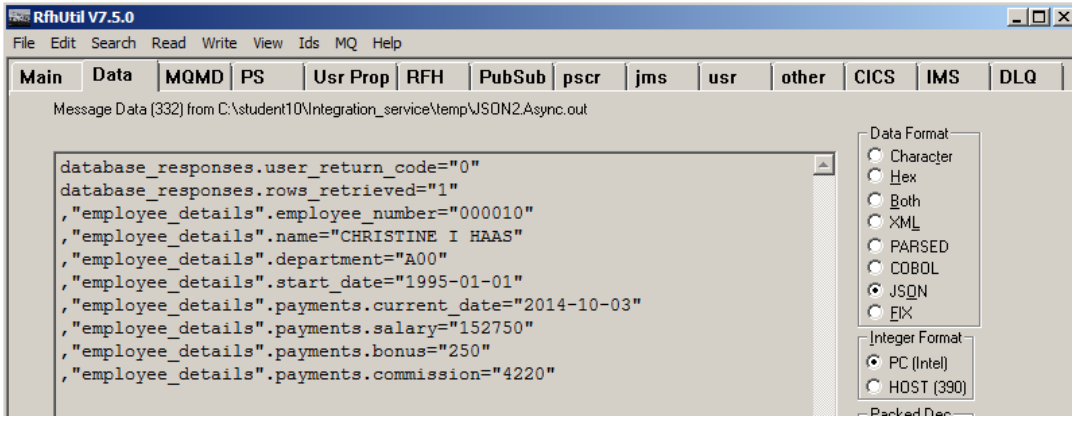
   Open the request "Employee 000010".

   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.