L01

IBM Integration Bus

Creating an Integration Service

Featuring:

Creating schemas from a database
Placing the schemas in a Shared Library
Using the schemas in an Integration Service
Placing a Mapping Node in a Shared Library
Referencing a Submap in a Shared Library
Using the Submap to read a database
Introduction to the Flow Exerciser

January 2016
Hands-on lab built at product
Version 10.0.0.3

1. Introduction

An integration service is a specialized application with a defined interface and structure that acts as a container for a Web Services solution.

You can integrate applications by using a service-oriented architecture (SOA). In an SOA, a service is often defined as a logical representation of a repeatable activity that has a specified outcome. Typically, a service is self-contained and its implementation is hidden from its consumers. To facilitate their reuse, services define a prescribed interface, which specifies how data is exchanged with the service.

In IBM Integration Bus, an integration service is a specialized application with a defined interface that acts as a container for a Web Services solution:

- It contains message flows to implement the specified web service operations.
- The interface is defined through a WSDL file.

Important note

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

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

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

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

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

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

1.1 Scenario Overview

In this lab, you will create an integration service and all the IBM integration resources required to implement the service. You will define the service data model and the WSDL that describes the service. You will implement and test the service operation.

The service accesses an external database and responds with data retrieved from the database. You will need to configure IBM Integration Bus to work with an external database system. In this lab, we are using DB2.

1.2 Outline of tasks

The tasks to complete in this lab are the following:

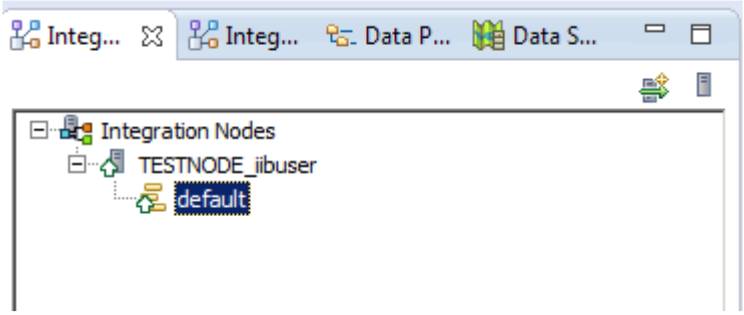
1. Configure the IBM Integration Bus default runtime node to work with DB2
    a. Runtime configuration: Configure a JDBCProvider configurable service. Configure the security credentials to work with the database.
    b. Development configuration: Create a Database Definition project, and the database connectivity configuration.
2. Create a Shared Library. This library will contain your message models, the WSDL that defines the new service, and the map and submap for the service.
3. Create a Message Model
    a. Create a message model based on a database table definition
    b. Extend the message model with other schema available in the library
4. Create an integration service, and define the WSDL associated with it
5. Implement the integration service operation
    a. Create the Submap that retrieves employee data from the EMPLOYEE table
    b. Create the main Message Map in the Shared Library
6. Test the service using the Integration Toolkit Flow Exerciser
7. Test the integration service using external tools (optional)
    a. Test the service using SOAPUI

1.3 Configure Integration Bus node to work with DB2

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

2. Start the IIB Toolkit, using the shortcut on the Windows Start menu. Accept the default name for the workspace (or you can specify your own location if desired).

   When you start the Toolkit, a new IIB node named TESTNODE_iibuser will be created. This node will be started automatically when the Toolkit starts. You will be able to see this node in the Integration Nodes view of the Toolkit.



   If the IIB Healthcare Pack product is installed, you will also see the Configurable Services folder.

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

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

   `c:\student10\Create_HR_database`

4. Run the command

   `3_Create_JDBC_for_HRDB`

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

5. Run the command

   `4_Create_HRDB_SecurityID`

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

   `mqsistop TESTNODE_iibuser`

   `mqsistart TESTNODE_iibuser`

1.3.1 Recreating the HRDB database and tables

> This section will not normally be required if you are running the lab scenarios on the pre-built VMWare workshop image. If you are creating your own environment, you will need to run the commands in this section. You may need to alter the database schemas and authorities in the DDL to reflect your own environment.

The HRDB database, and the EMPLOYEE and DEPARTMENT tables have already been created on the supplied VMWare image. If you wish to recreate your own instance of this database, do the following tasks:

1. Login with the user iibadmin (password=passw0rd)
2. Open an IIB Command Console (from the Start menu), and navigate to

   `c:\student10\Create_HR_database`

3. Run the commands

   `1_Create_HRDB_database`

   `2_Create_HRDB_Tables`

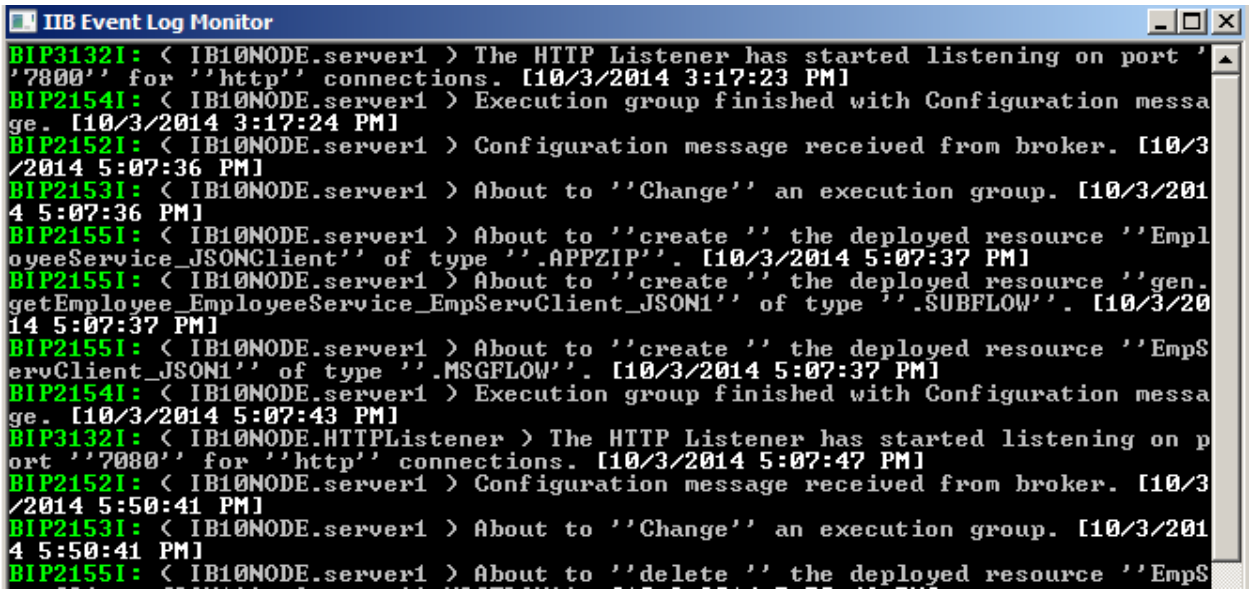
4. Logout user iibadmin.

Appropriate database permissions are included in the scripts to GRANT access to the user iibuser.

1.4 Open the Windows Log Monitor for IIB

A useful tool for IIB development on Windows is the IIB Log Monitor. This tool continuously monitors the Windows Event Log, and all messages from the log are displayed immediately. This tool is no art of the IIB product, and is currently only available with the IIB workshop system.

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



2. Create Database Definition, schemas and Submap

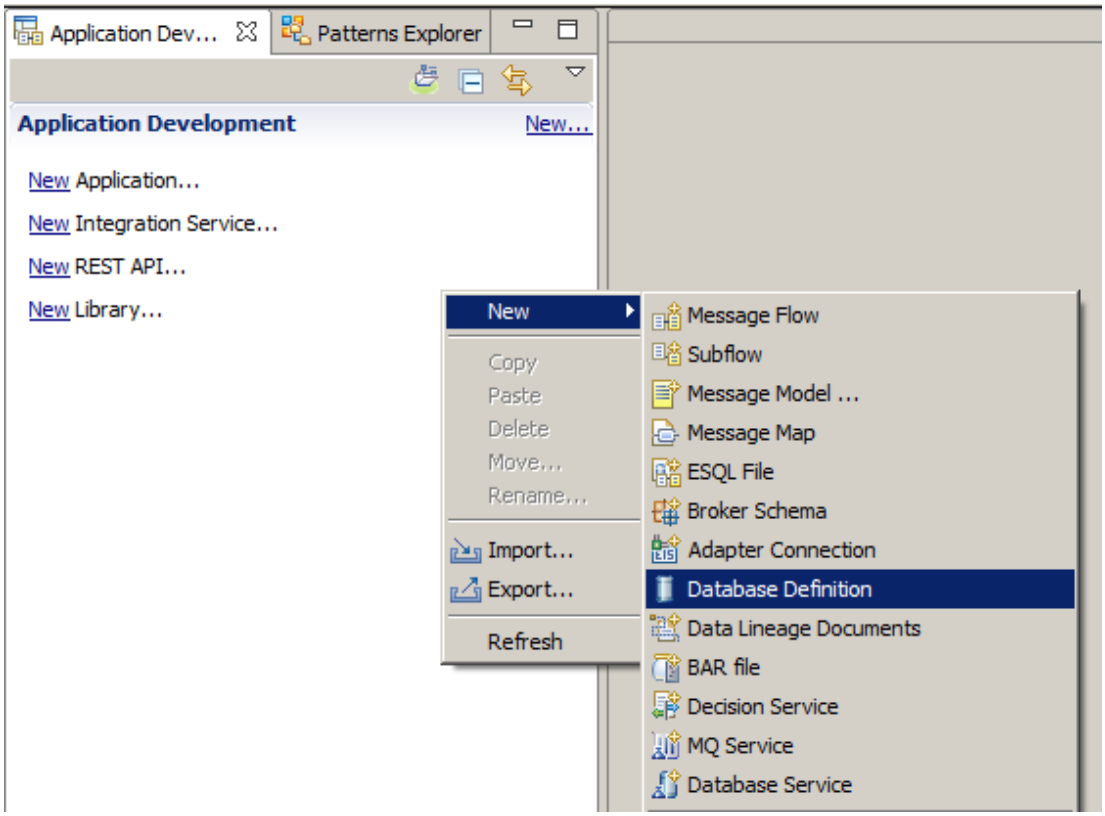
The IIB environment is now prepared, so move on to creating the artefacts that will be used in this lab. The first artefact is a Database Definition that represents the HRDB database.

2.1 Create a Database Definition for HRDB

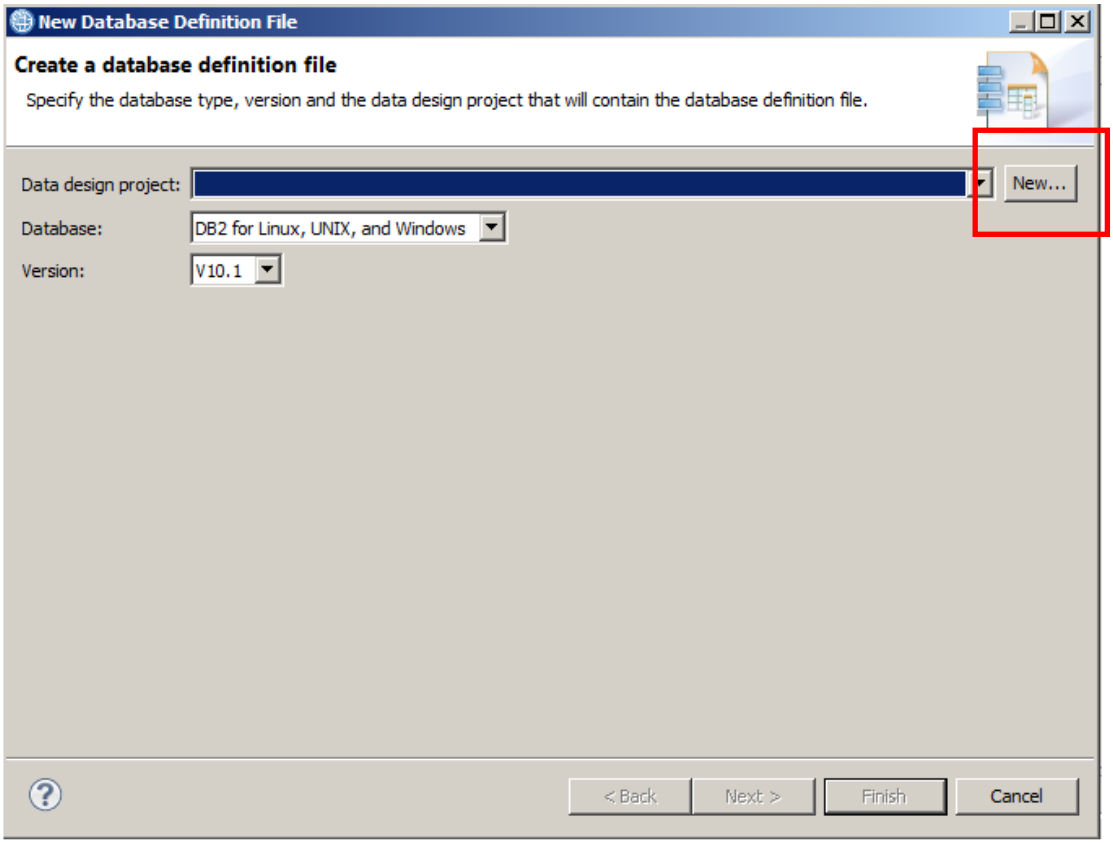
The first artefact is a Database Definition that represents the HRDB database. The database definition is required to enable the IIB Toolkit to be able to connect to the database, so that it can extract the required table and schema definitions, and use these to create an IIB message model.

Make sure you are logged in as "**iibuser**", and open the IIB Toolkit using the shortcut on the Windows Start menu.

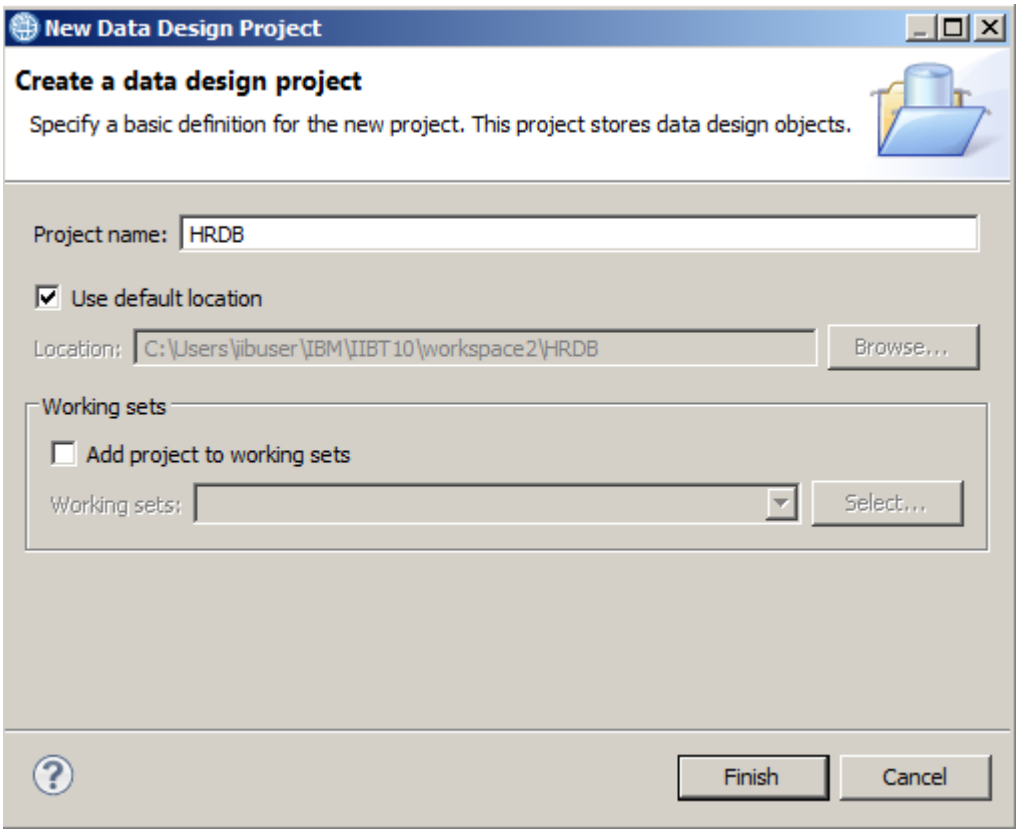
1. In the Toolkit navigator, right-click in blank space, select New, Database Definition.



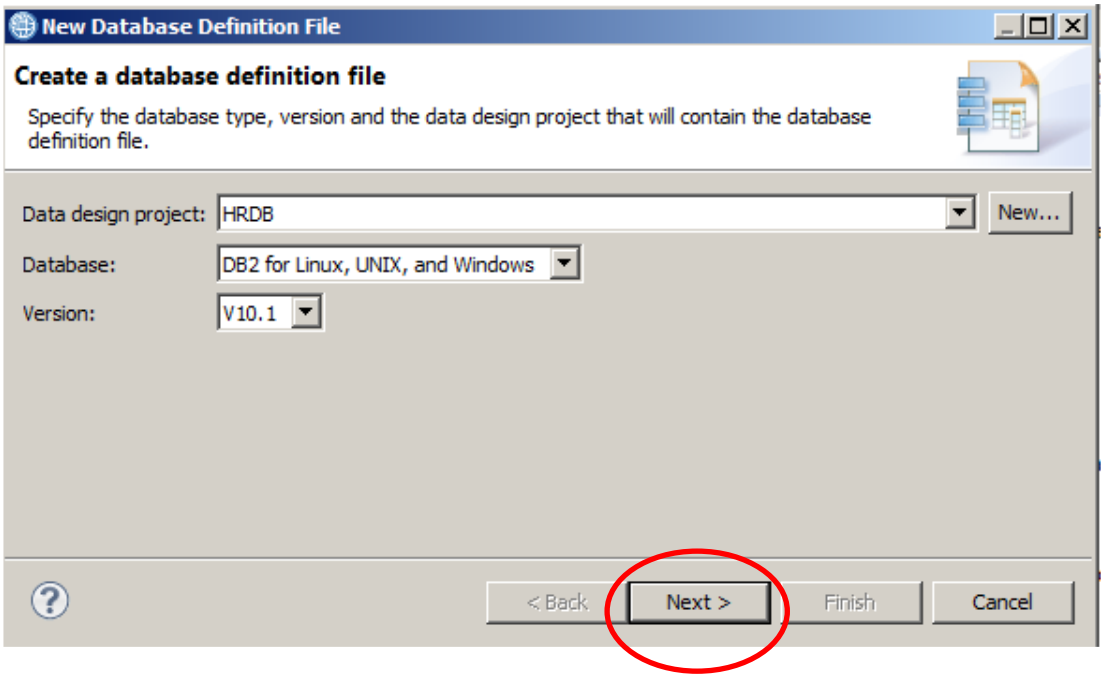
2. Click New to create a data design project.



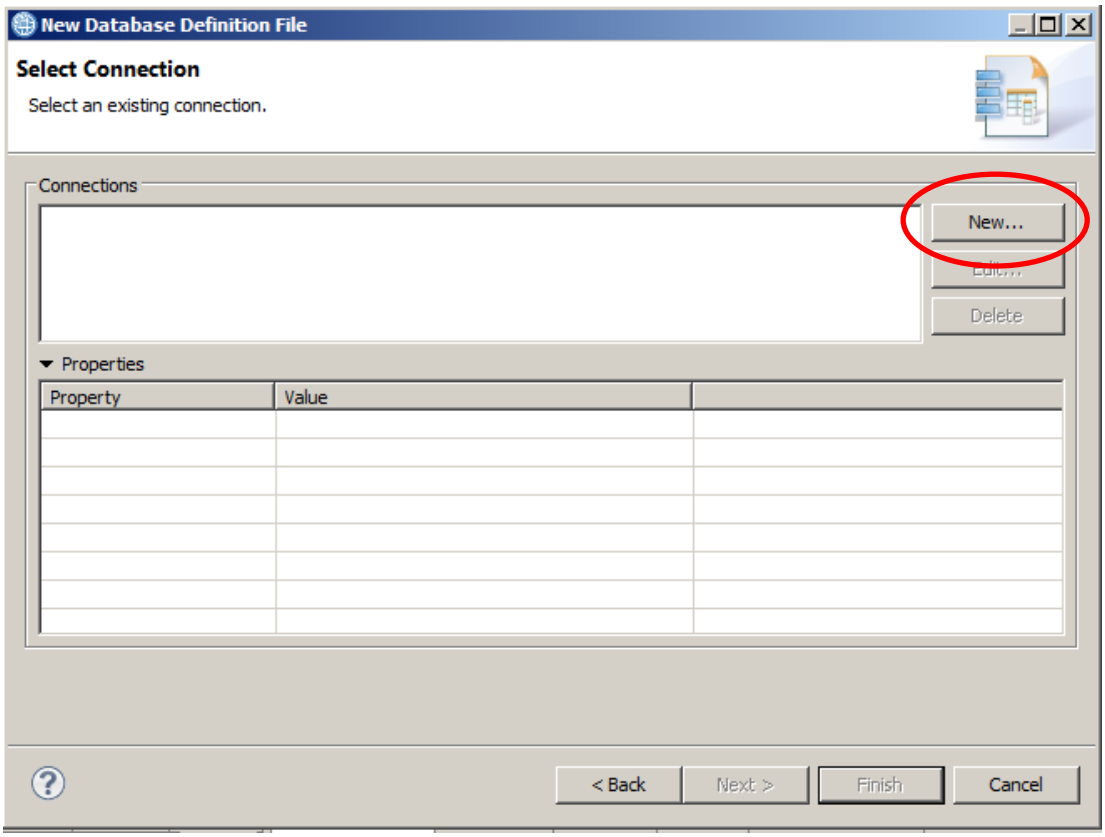
3. Name it HRDB. Click Finish.



4. Click Next.



5. Click New to create a new connection.



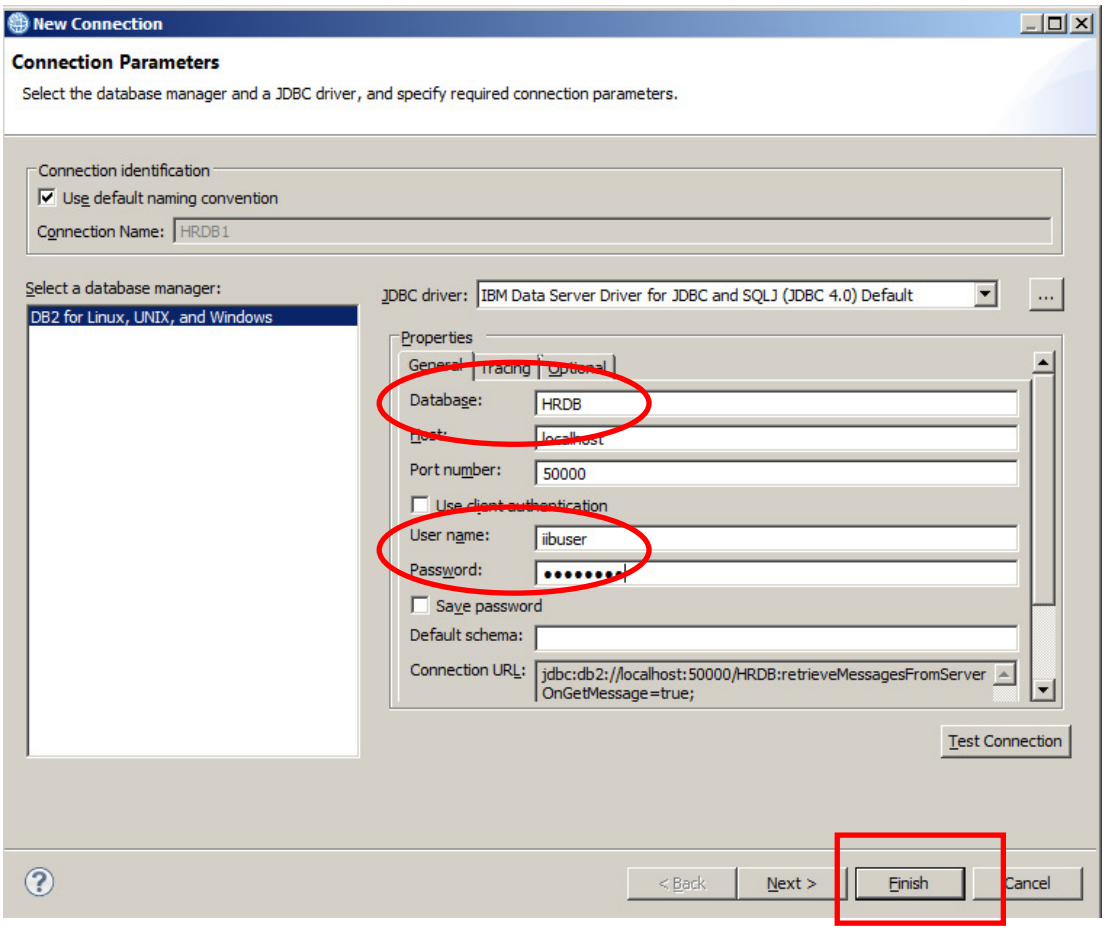
6. Set the Database name to HRDB.

   Specify connection credentials (user=**iibuser**, password=**passw0rd**),

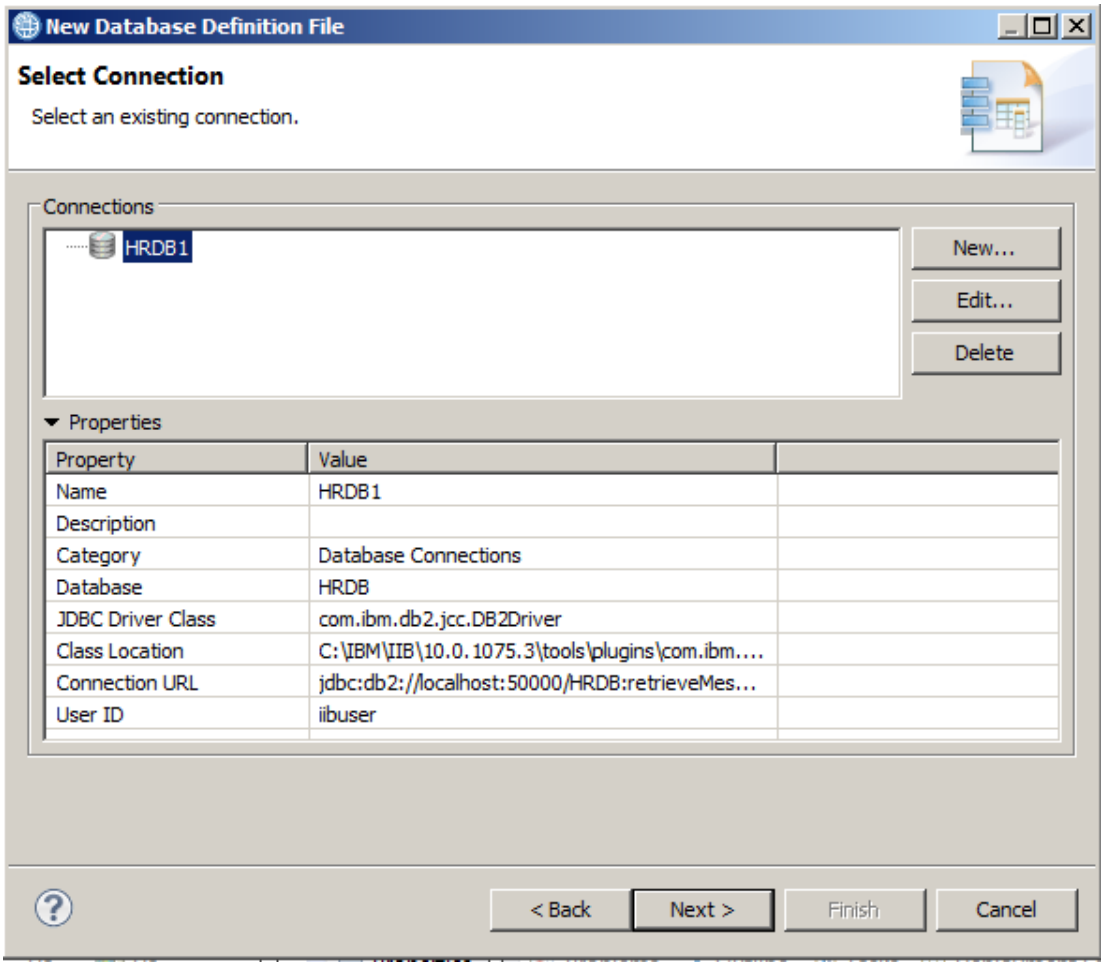
   If you wish, you can click Test Connection, to make sure you have specified the connection details correctly.

   Optionally, select save the password, and click Finish. (this is only required for subsequent Toolkit access to the database connection; it is not used by the IIB runtime).

   (You may need to expand the dialog window to see the Save Password checkbox).

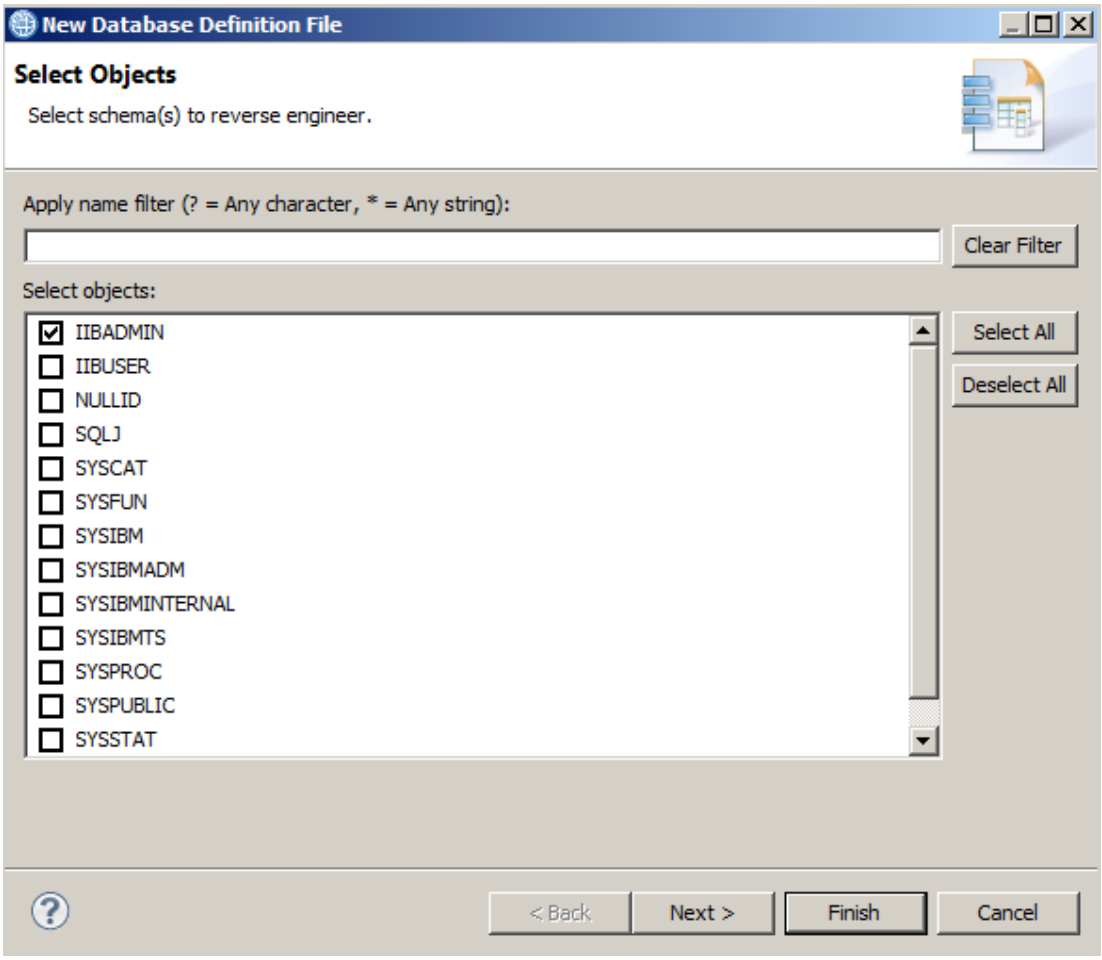


7. Highlight the new connection (HRDB1), and click Next.



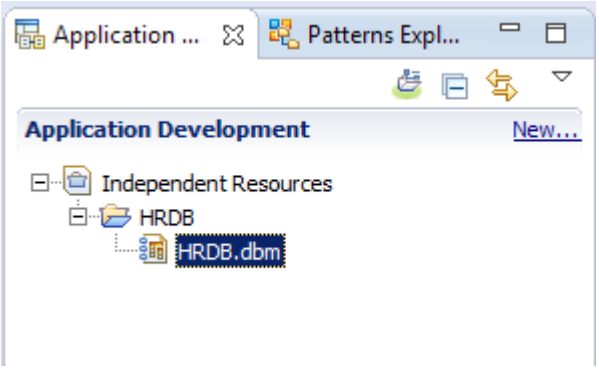
8. Select the iibadmin schema, and click Finish.

   (Note that the HRDB database has been defined with the IIBADMIN schema. "IIBUSER" has been granted permissions on this database. For reference, this is shown in the script 2_Create_HRDB_Tables.cmd).



9. The database definition for the HRDB database is now present as a project in the Independent Resources part of the navigator.

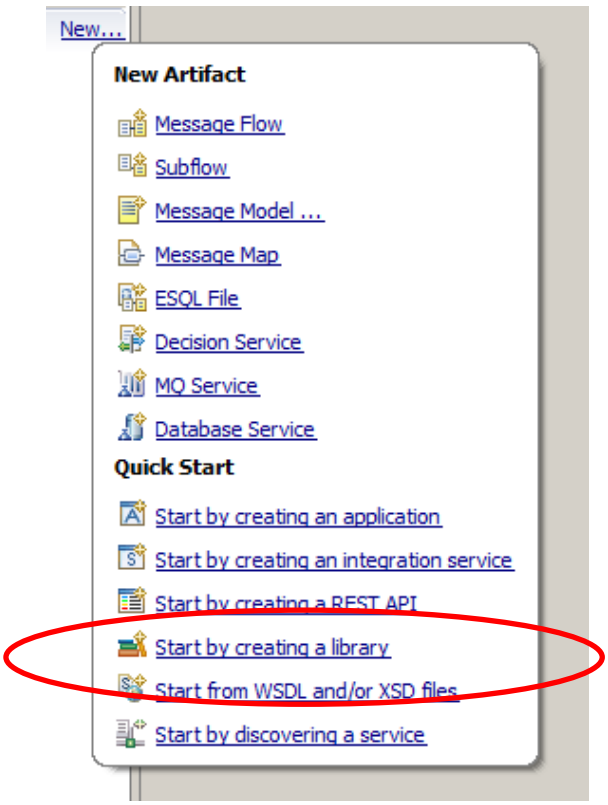
   Note that this HRDB.DBM file can be used by any other developer requiring access to the database. It contains a definition of the database, and the selected tables and schemas, and can be shared as required, to create the related IIB schema definitions.



2.2 Create the IIB Message Model (XML schema)

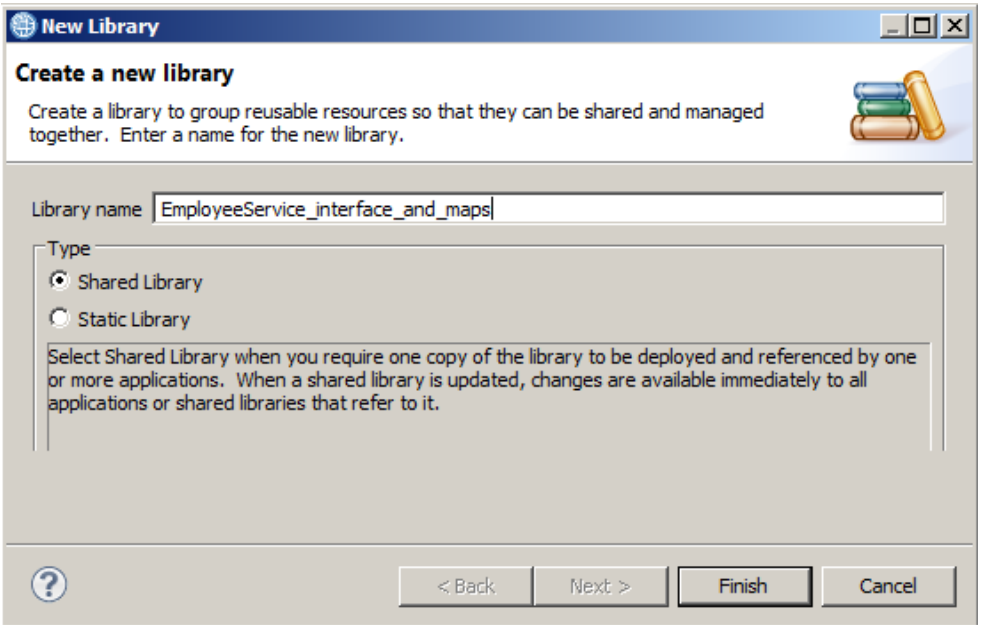
The message model schema will be derived from the HRDB database definition that you have just created. This lab will use the EMPLOYEE table, but you will generate schemas for both the EMPLOYEE and DEPARTMENT tables, for future use in later labs.

1. A Message Model is defined within a container called a Library, so you must first create a Library. In the Navigator view, click New, and select "Start by creating a library".

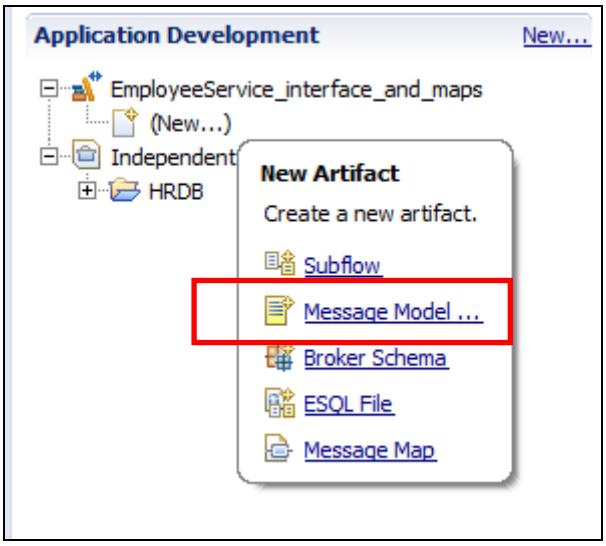


2. You can choose whether to create a Shared or Static library. Choose Shared (the default), and name the library **EmployeeService_interface_and_maps**.

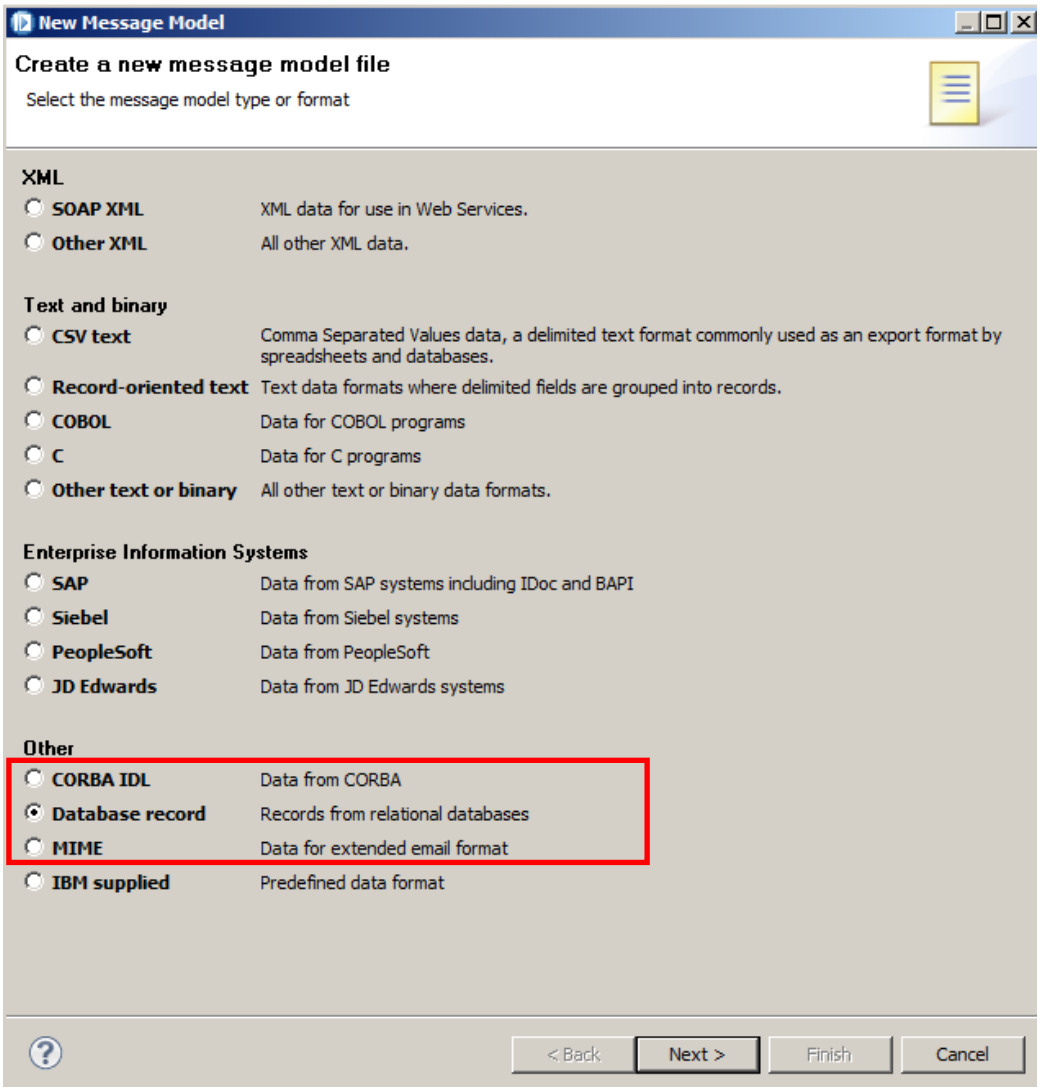
   Click Finish.



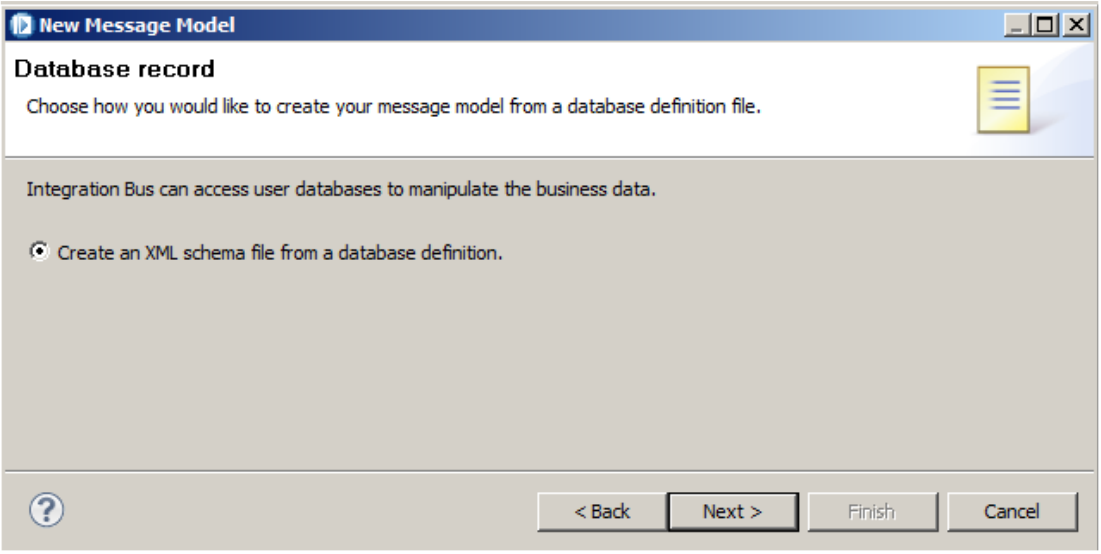
3. In this library create a new Message Model (click "New" under the library name), then select "Message Model".



4. Select the Database Record type, and click Next.



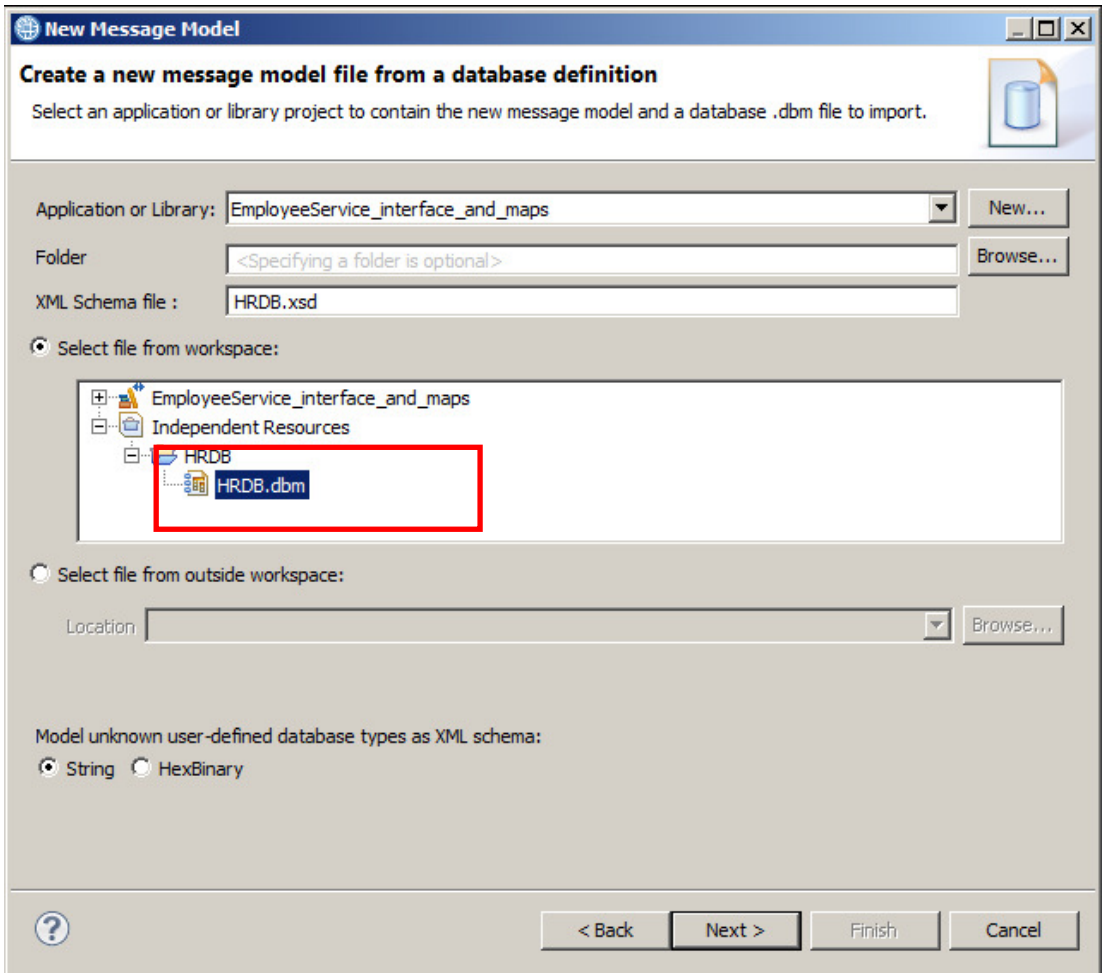
5. Create an XML schema - click Next.



6. Select the HRDB.dbm file from the database definition project in the workspace (under Independent Resources).

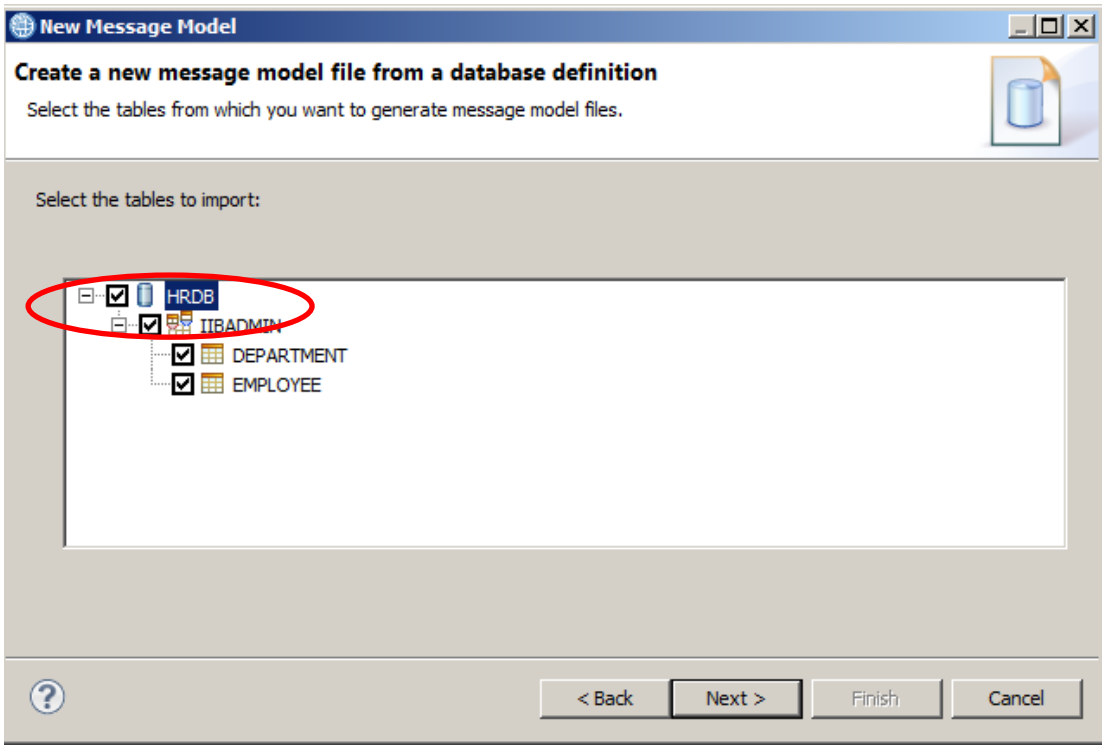
   This will automatically create the XML Schema file name, HRDB.xsd.

   Click Next.

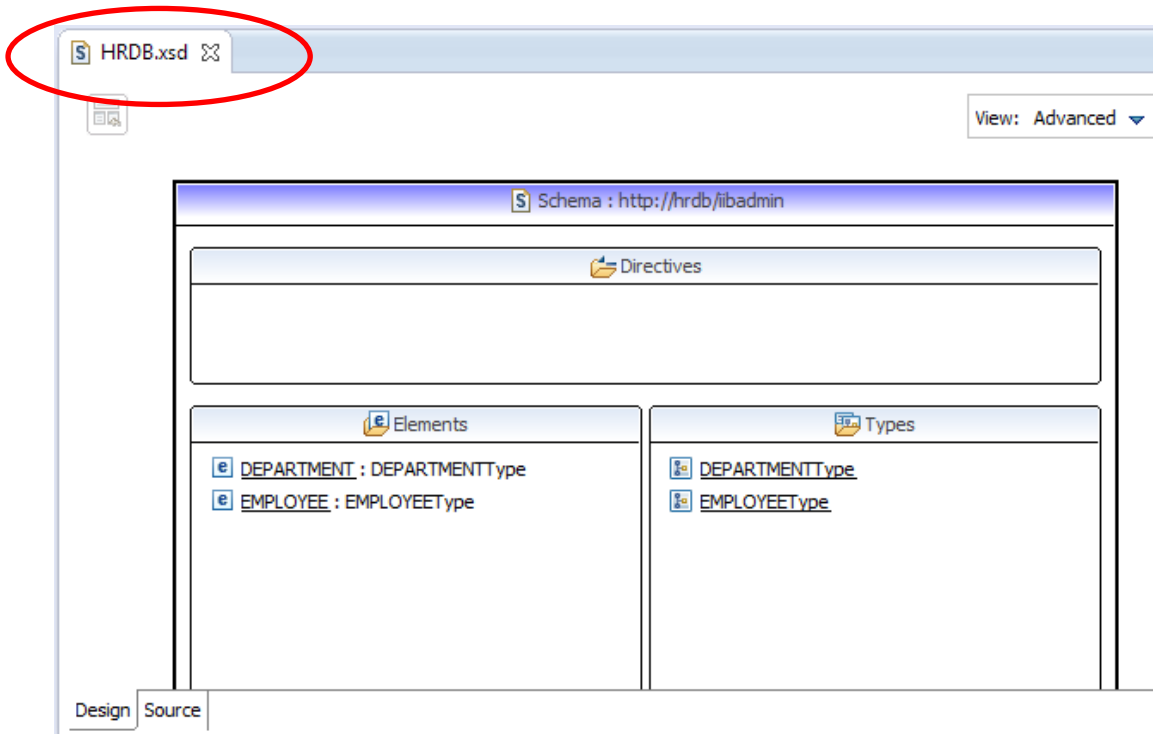


7. Select the HRDB database, which will automatically select the EMPLOYEE and DEPARTMENT tables.

   Click Next, then Finish.



8. The schema will be created, and the Toolkit will open it with the XML schema editor. Note that the name of the schema is HRDB.xsd.



2.3 Add additional references to the schema

This lab will use a database scenario to illustrate the use of the Graphical Mapping Node within an Integration Service.

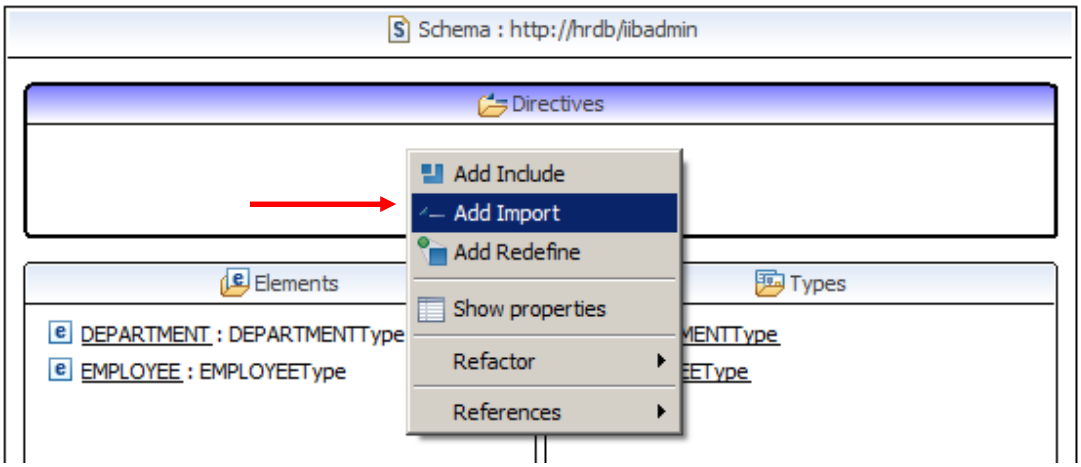
To do this, the Integration Service will pass a return code, plus details of any SQL messages, to the requesting application. You will therefore import a second schema, DBResp, into the HRDB schema.

The DBResp schema contains several elements that can be used to report SQL responses, as well as a user return code.

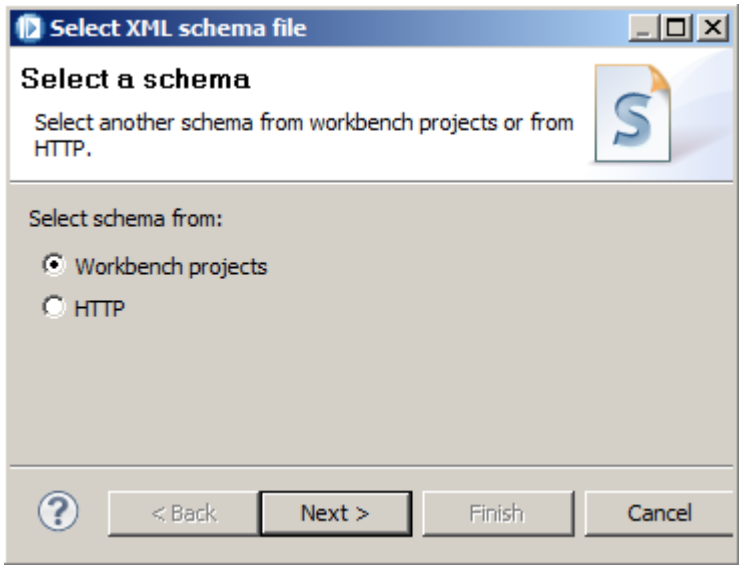
The HRDB schema will be augmented with a new complex type, EmployeeResponseType, which contains the EMPLOYEE and DBResp types.

1. In the HRDB schema, right-click in the Directives section.

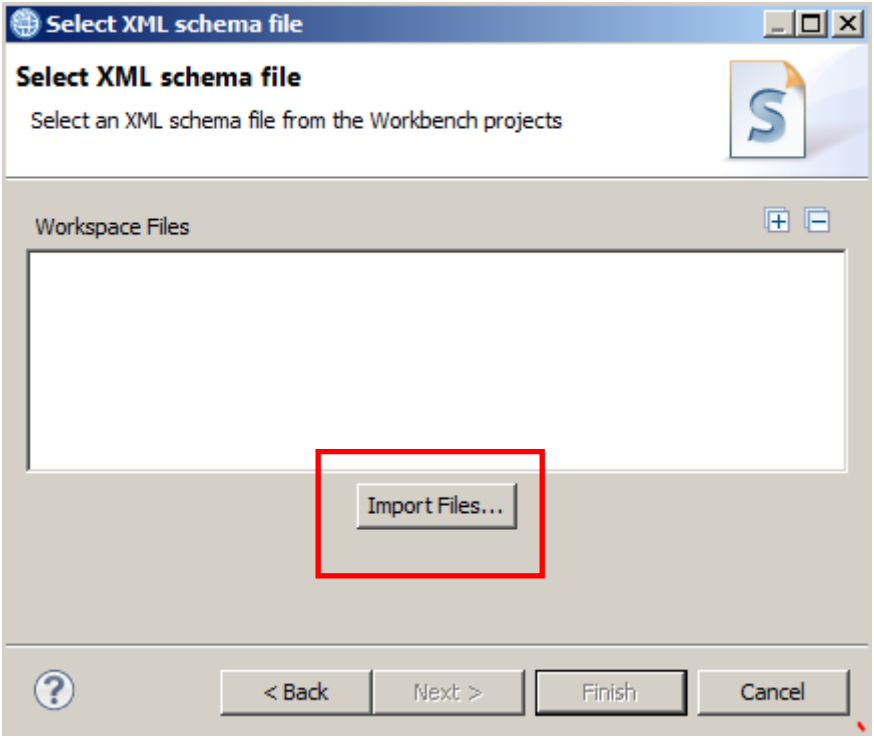
   Click Add Import.



2. Select Workbench projects, and click Next.



3. Click Import Files.



4. In the "From directory", use the browse button to navigate to

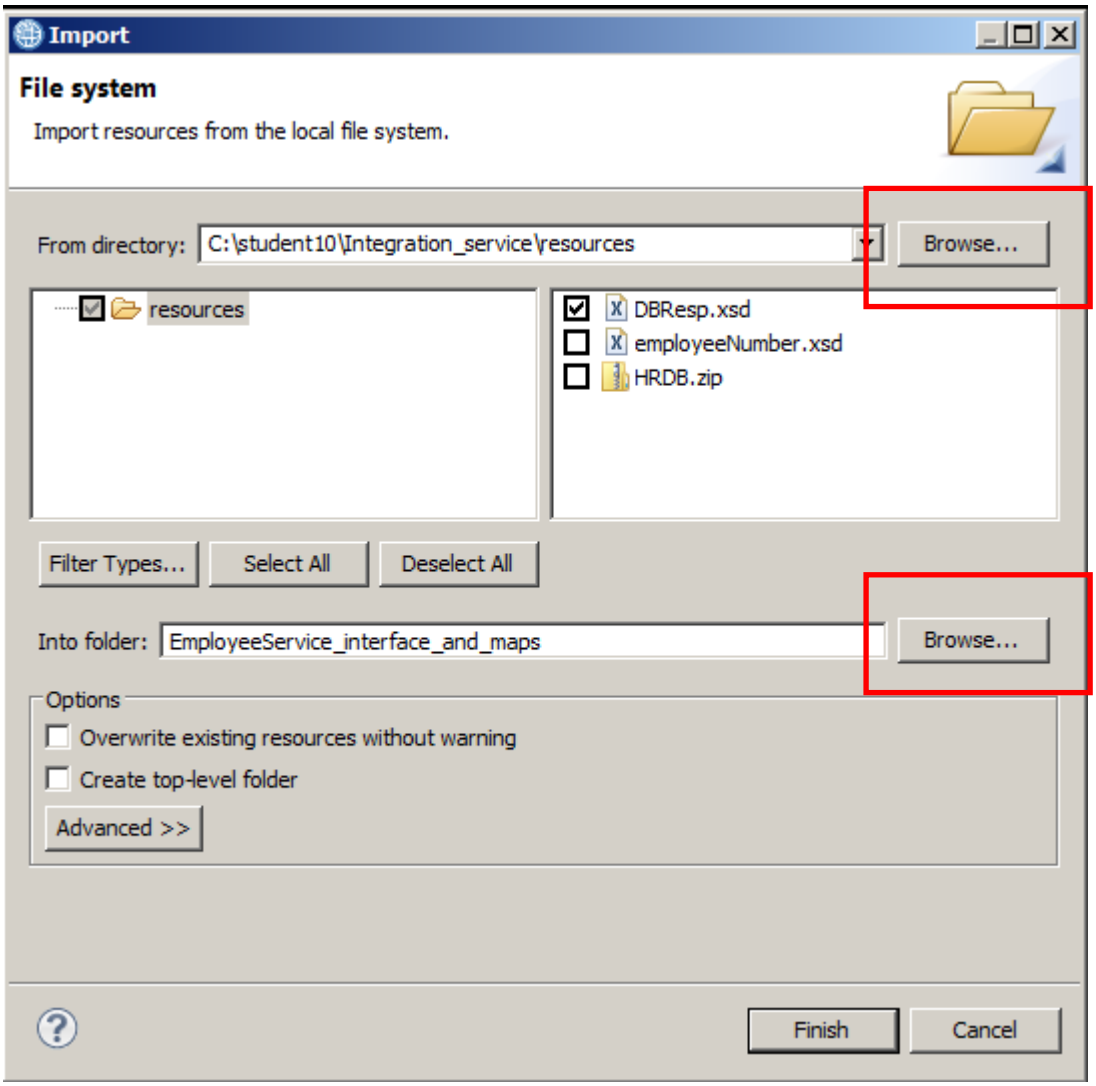
   `c:\student10\integration_service\resources`

   Select the **DBResp.xsd** schema.

   Use the second Browse button to select the target folder:

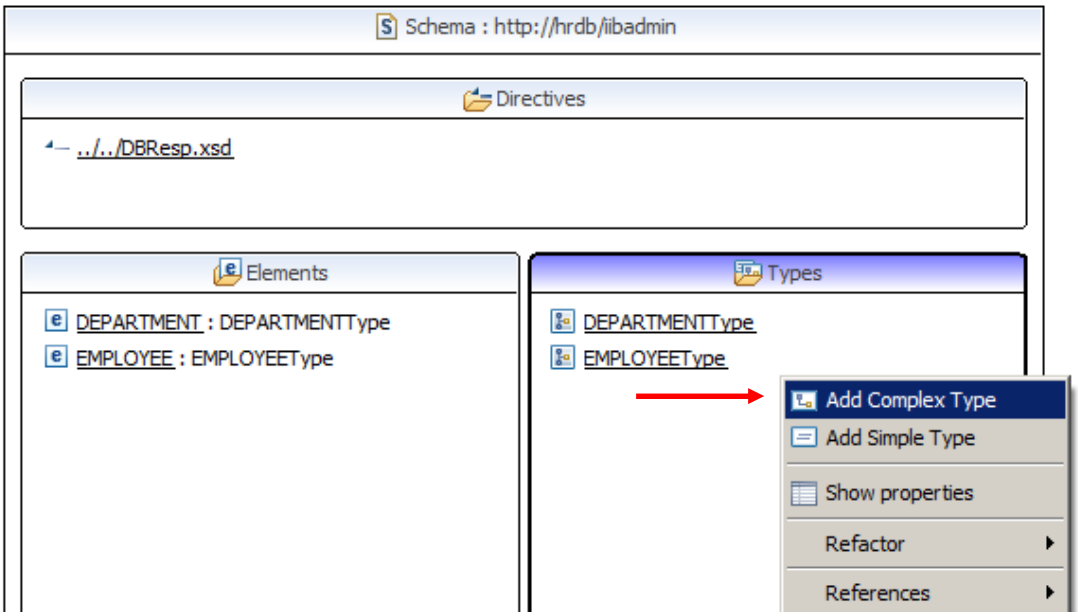
   `EmployeeService_interface_and_maps`

   Click Finish, and then Finish again on the next window.

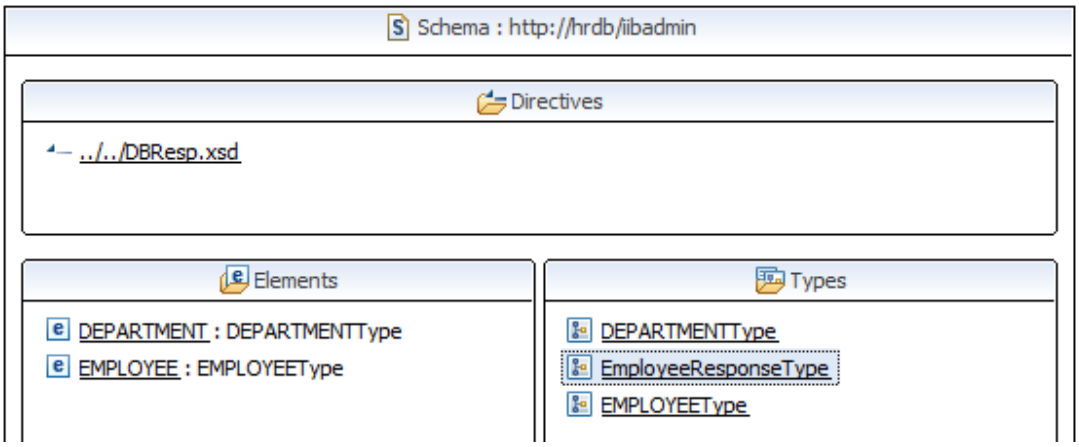


5. Note that the DBResp schema has been imported.

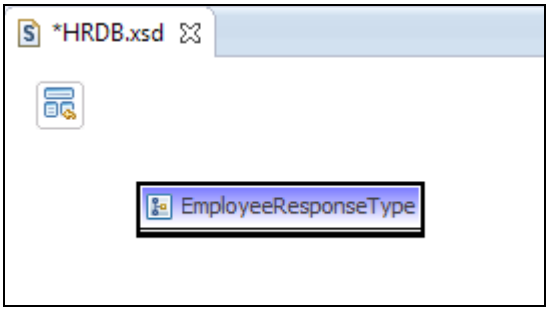
   In the Types pane, right-click in an open area, and select Add Complex Type.



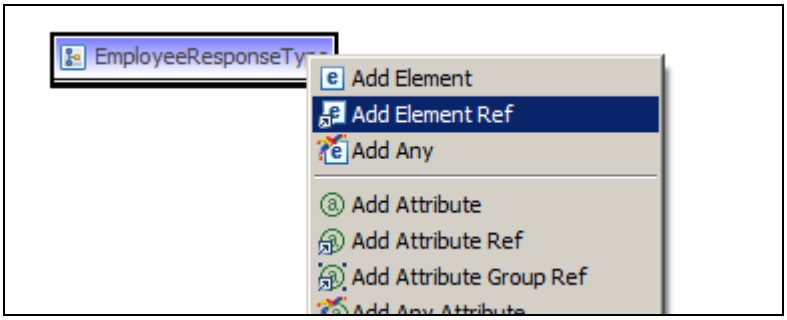
6. Set the name of the new Complex Type to EmployeeResponseType.



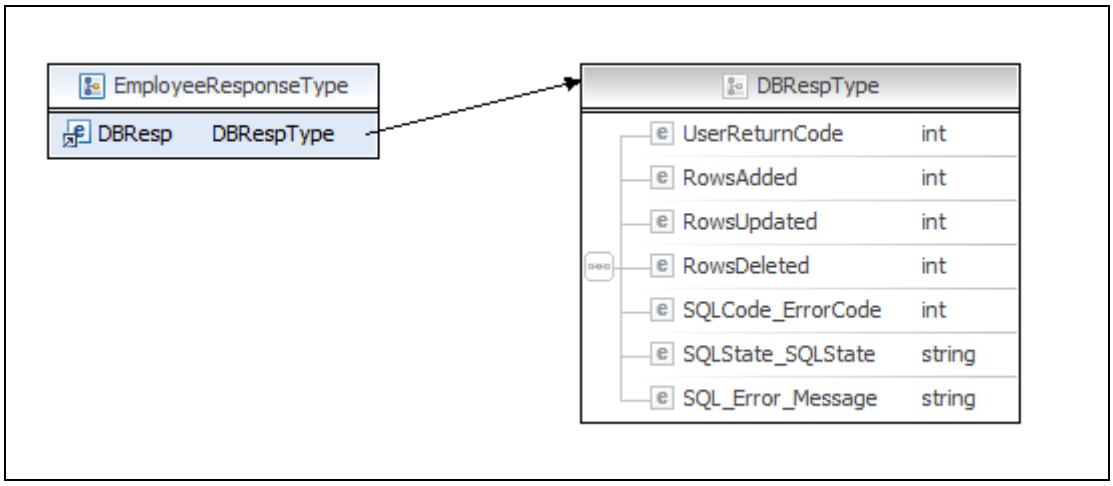
7. Double-click EmployeeResponseType which will open the schema editor for this type.



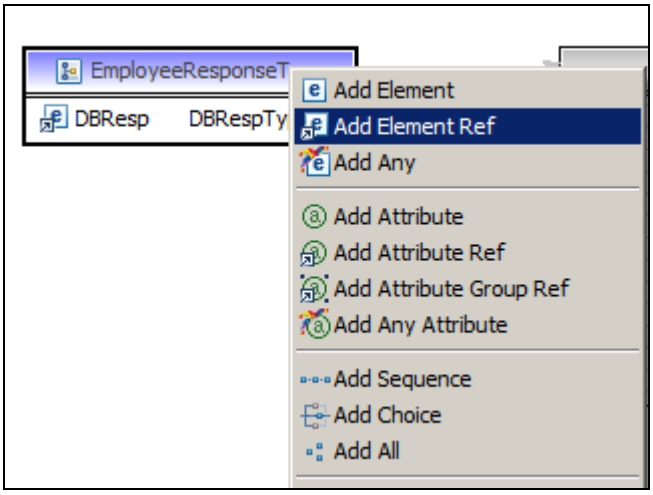
8. Right-click EmployeeResponseType and select Add Element Ref.



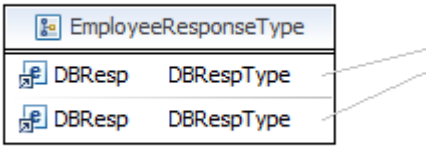
9. The editor will add a default element using the first alphabetically available. Luckily, since DBResp is the first available, this is the one that we want.



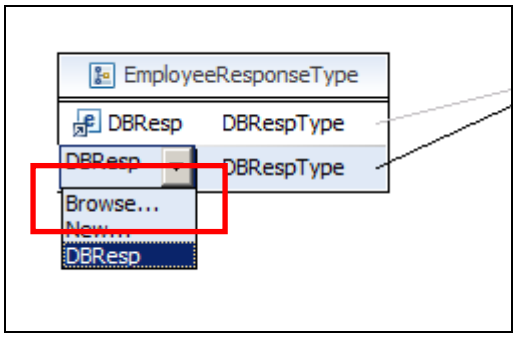
10. Now, add a second element reference, by right-clicking the EmployeeResponseType again.



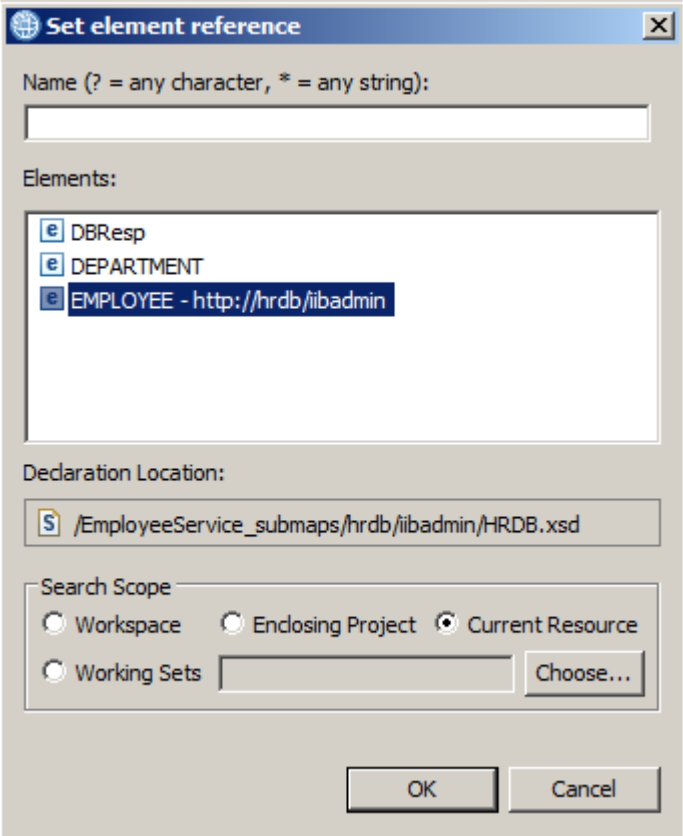
This will add a second reference to DBResp.



11. To change this second reference, click the second instance of DBResp (click DBResp, not DBRespType). A drop-down menu will be shown. Click Browse.

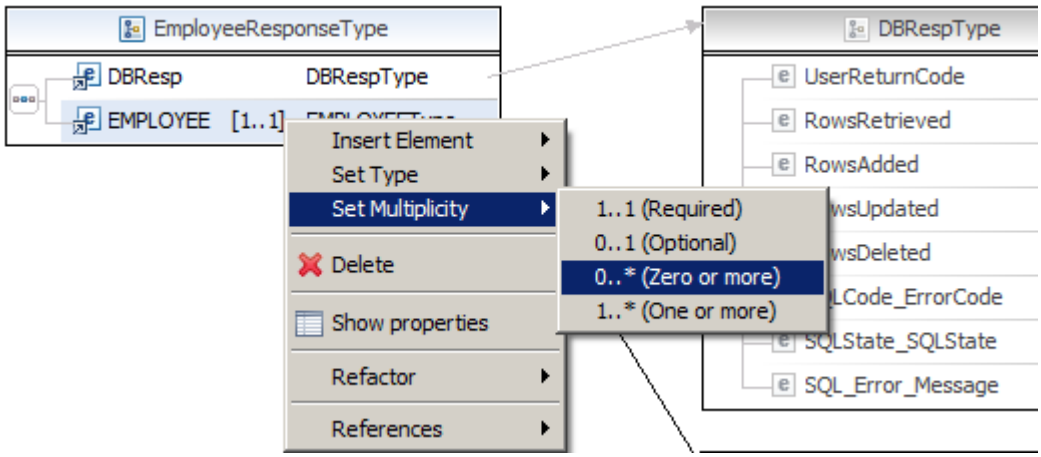


12. Highlight the EMPLOYEE element, and click OK.

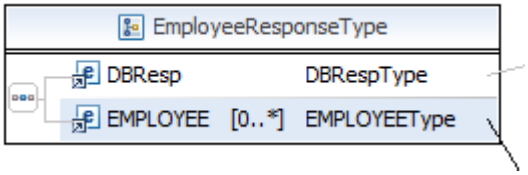


13. Since multiple rows might be returned from the EMPLOYEE table, it will be necessary to be able to handle multiple instances of the EMPLOYEE element within EmployeeResponse, so set the Multiplicity as follows:

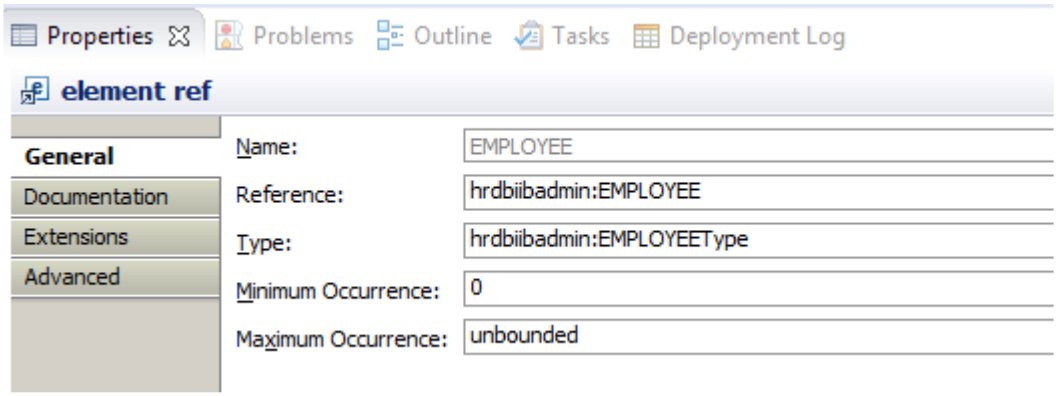
    Right-click EMPLOYEE, click "Set Multiplicity", then "0...* (Zero or more)".



When you've done this, the complex element will look like this (click the EMPLOYEE element) (You may need to click in an open area of the editor to update the display of the multiplicity).



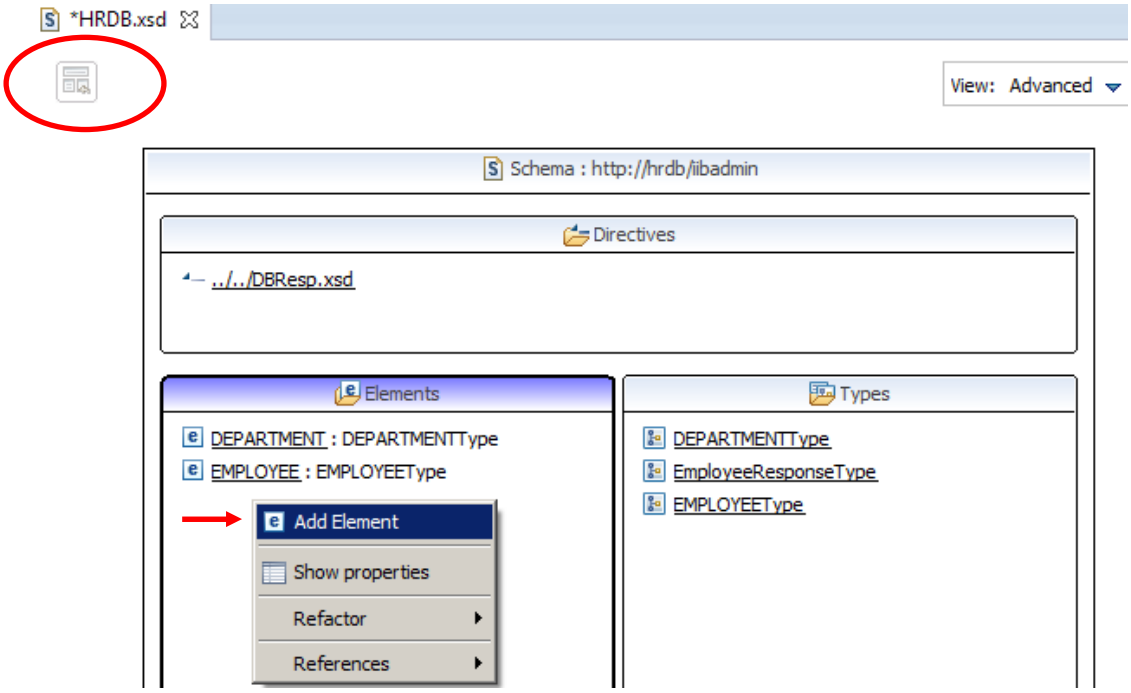
The properties will look like this (note the minimum and maximum occurrences when the EMPLOYEE element is selected):



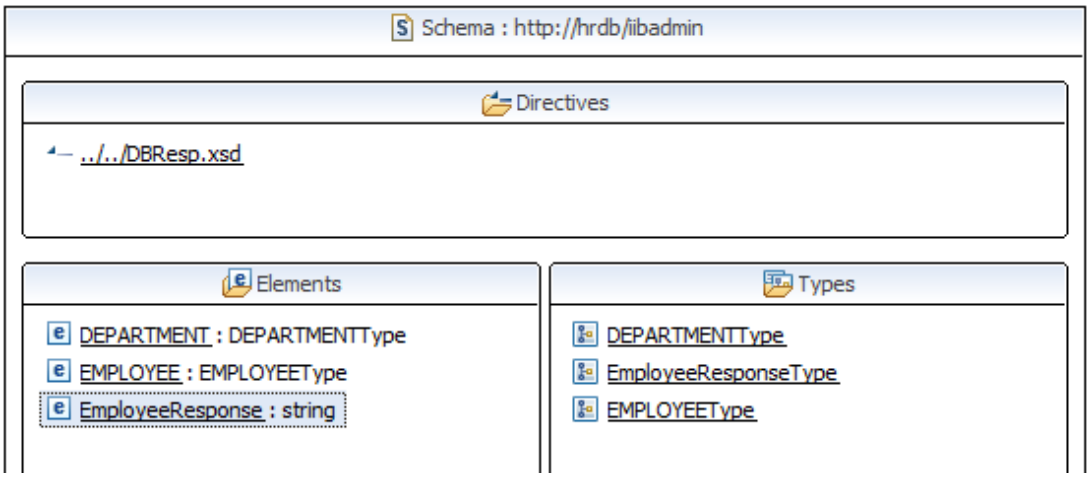
14. Return to the high-level definition of the schema (click the highlighted icon shown below).

    The Mapping Node bases its message inputs and outputs on element definitions, rather than element types, so it is necessary to define a new element named EmployeeResponse.

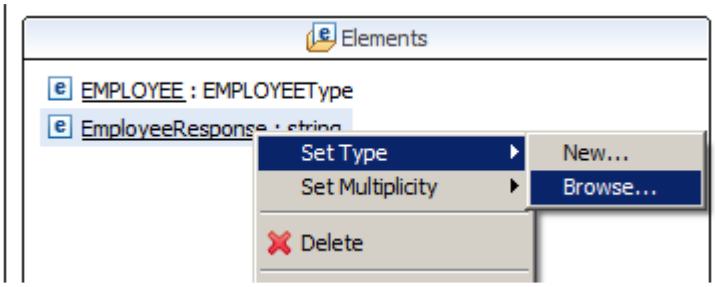
    In the Elements pane, right-click, and select "Add Element".



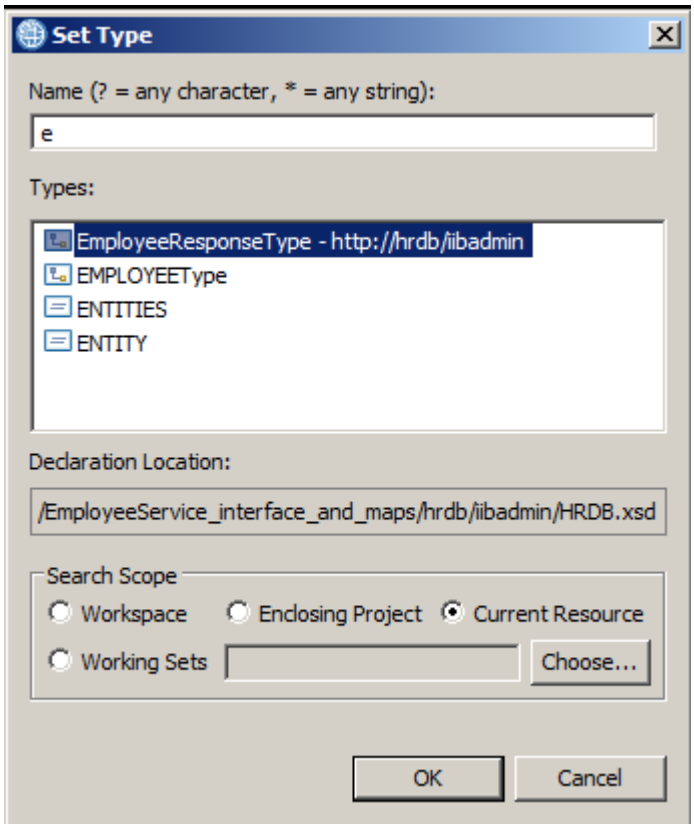
15. Name the new element EmployeeResponse.



16. Set the Type of the new element to EmployeeResponseType (right-click, Set Type, Browse).



Type "e" into the filter to reduce the list. Select EmployeeResponseType, and click OK.



Save and close the HRDB.xsd schema file.

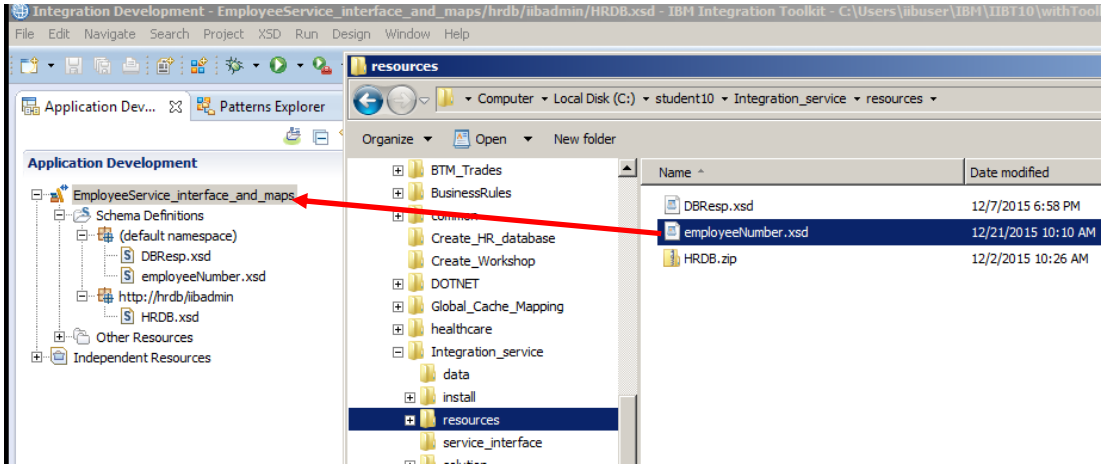
2.4 Import employeeNumber Schema

Finally, we need an element (and element type) that represents the employee number as a single element.

1. For the purposes of this lab, a suitable schema has been provided, so copy/paste (or drag/drop) this into the shared library.

   Use the file
   `c:\student10\integration_service\resources\employeeNumber.xsd`

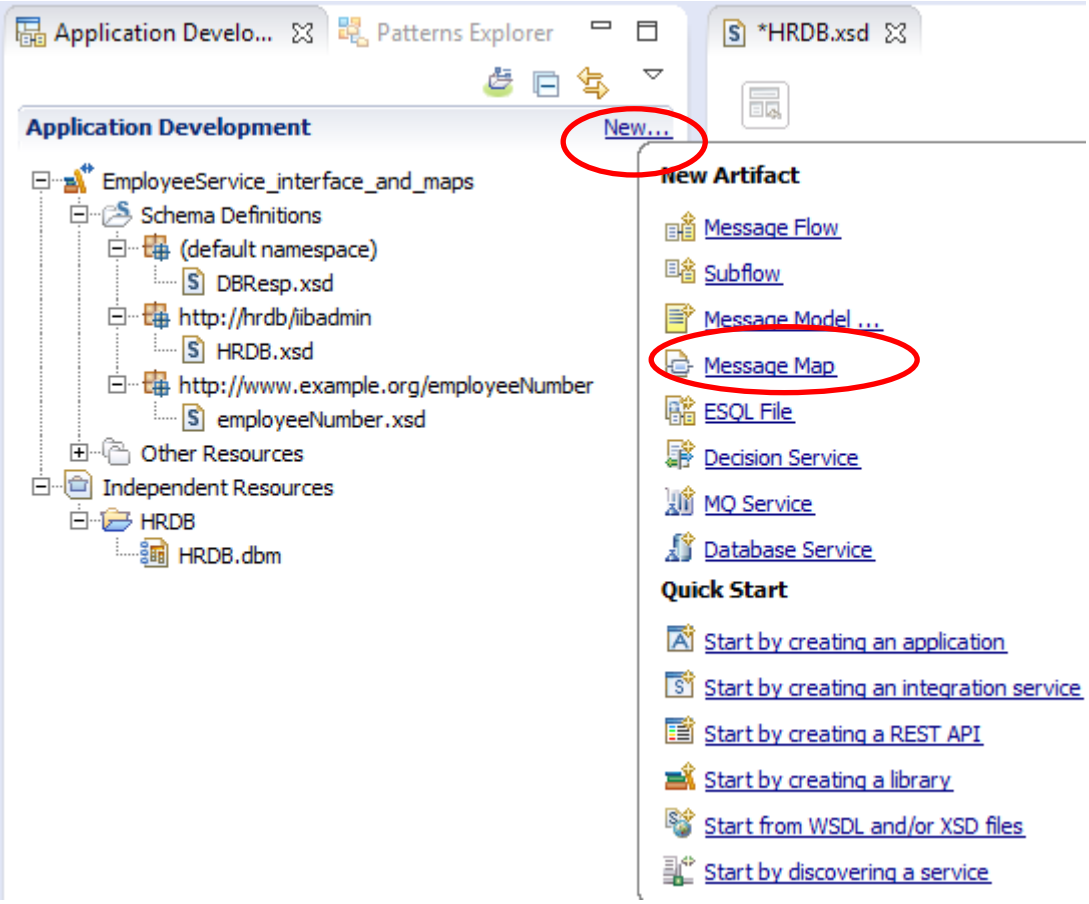
   This schema has no namespace, so will be imported into the (default namespace).



2.5 Create the submap

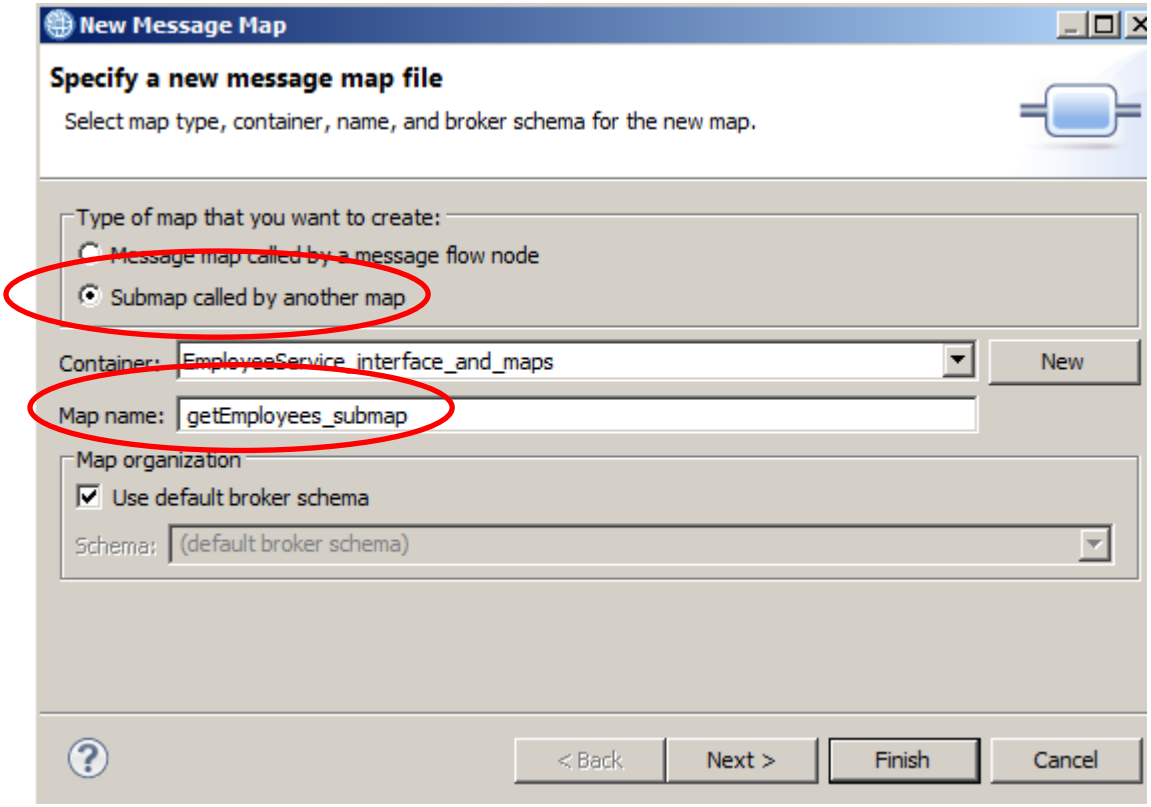
The new service will contain just one operation. This operation will use a Mapping Node to retrieve employee details from the EMPLOYEE table in the HRDB database. The function to access the database will be isolated in a submap, so that it can be invoked from a variety of applications. The submap will be created and stored in the Shared Library.

1. Highlight the EmployeeService_interface_and_maps library, and click New. Select Message Map.



2.
   - Set Type of map = "Submap called by another map".
   - Make sure the Container is EmployeeService_interface_and_maps
   - Set the Map name to "getEmployees_submap"
   - Double-check that you have selected Type = "**submap**".

   Click Next.

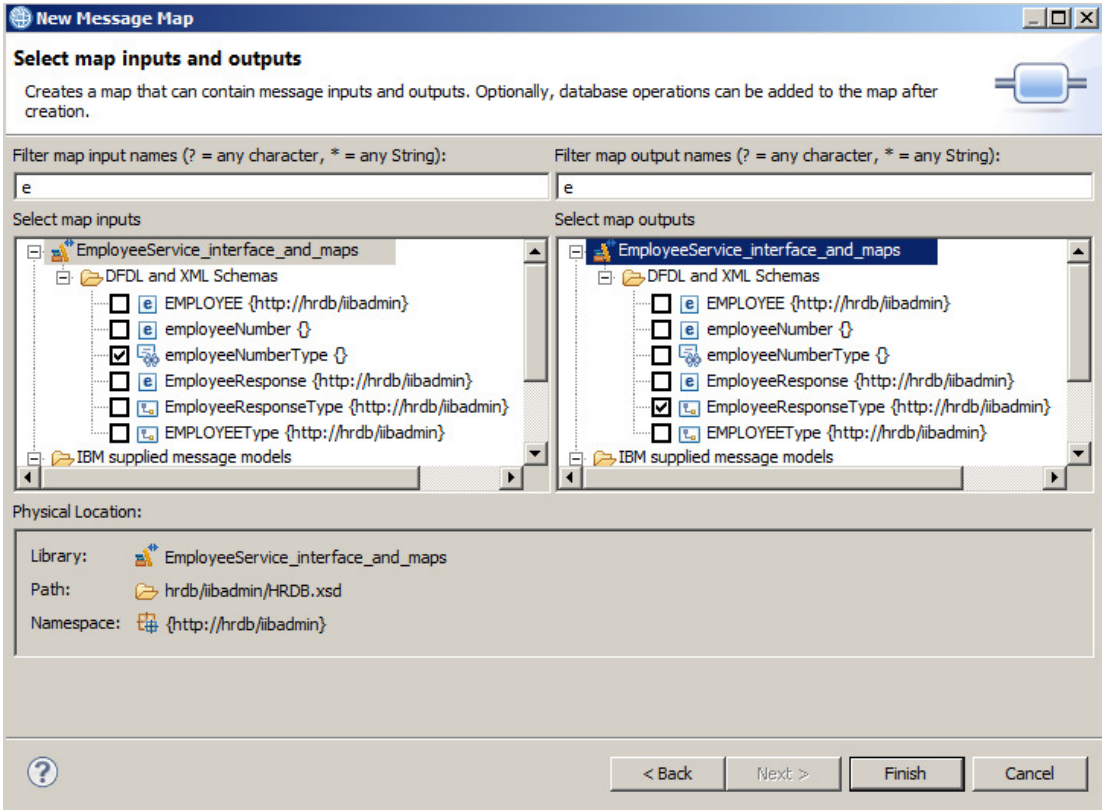


3. Expand the EmployeeService_interface_and_maps library for both input and output to the map. Expand DFDL and XML Schemas.

   Since this is a submap, you should use only "Type" definitions, not elements.

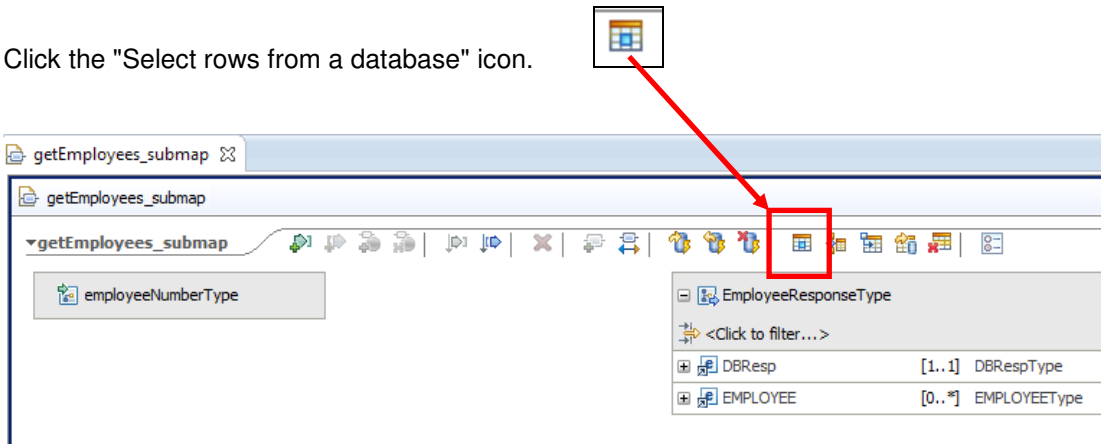
   - For the input, select the input message "**employeeNumberType**".
   - For the output, select the output message "**EmployeeResponseType**".

   Click Finish.

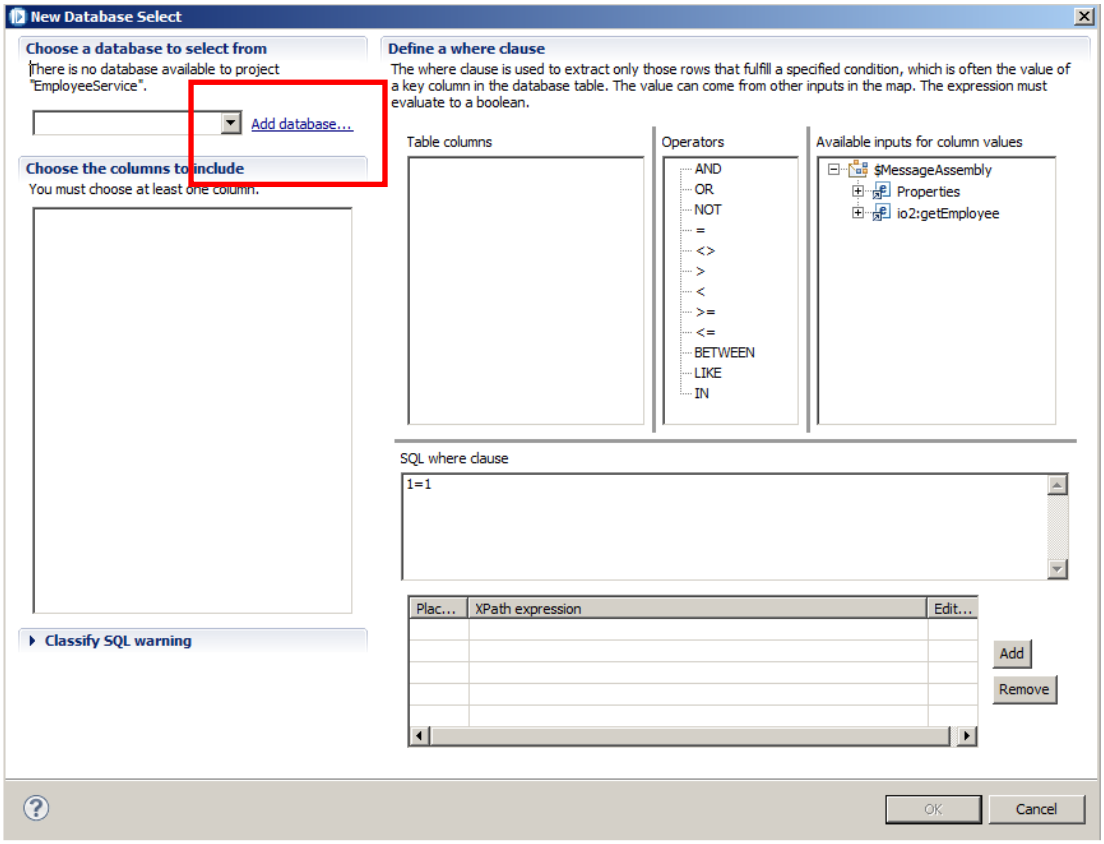


4. The map editor will be shown, but no transforms have been made so far.

   Click the "Select rows from a database" icon.

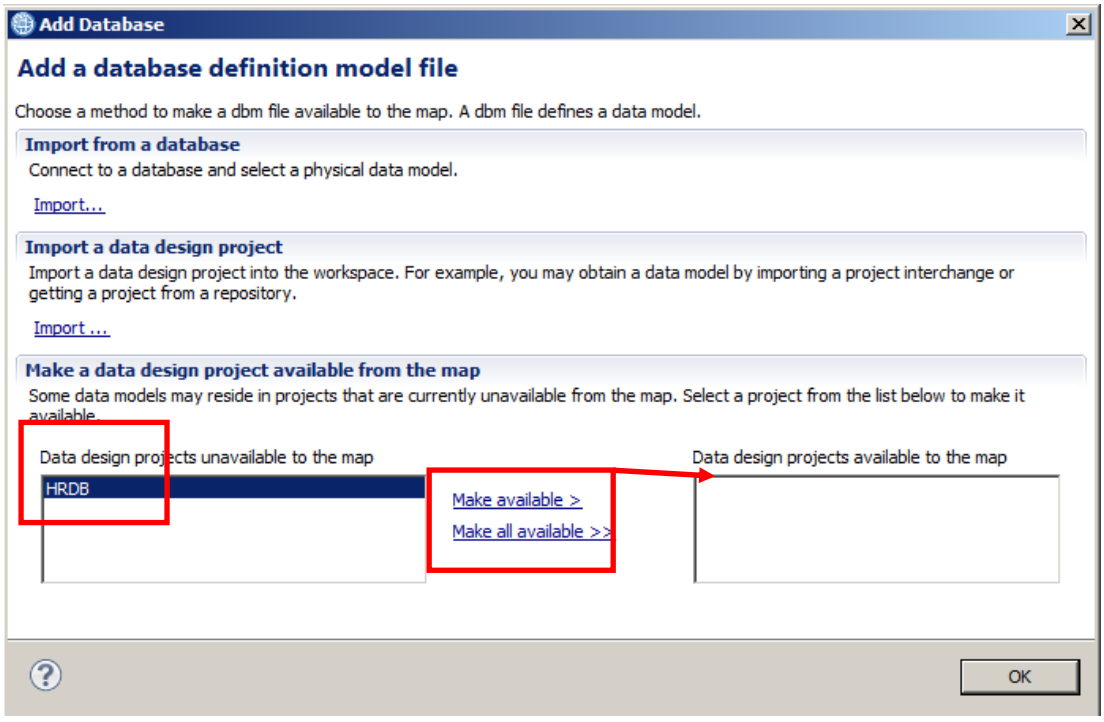


5. Since the shared library does not have a project reference to the HRDB.dbm project, click the "Add database" link.



6. Select the HRDB database, and "Make available".

   When HRDB appears in the pane on the lower right, click OK.



7. In the Database Select wizard:

    1. Select (tick) the EMPLOYEE table (note this will select all columns - you can select individual columns if you want).
       Note - only select the EMPLOYEE table, not the DEPARTMENT table. This function is the generating the SQL SELECT clause.

    2. Remove the "1=1" phrase from the "SQL where clause" pane.

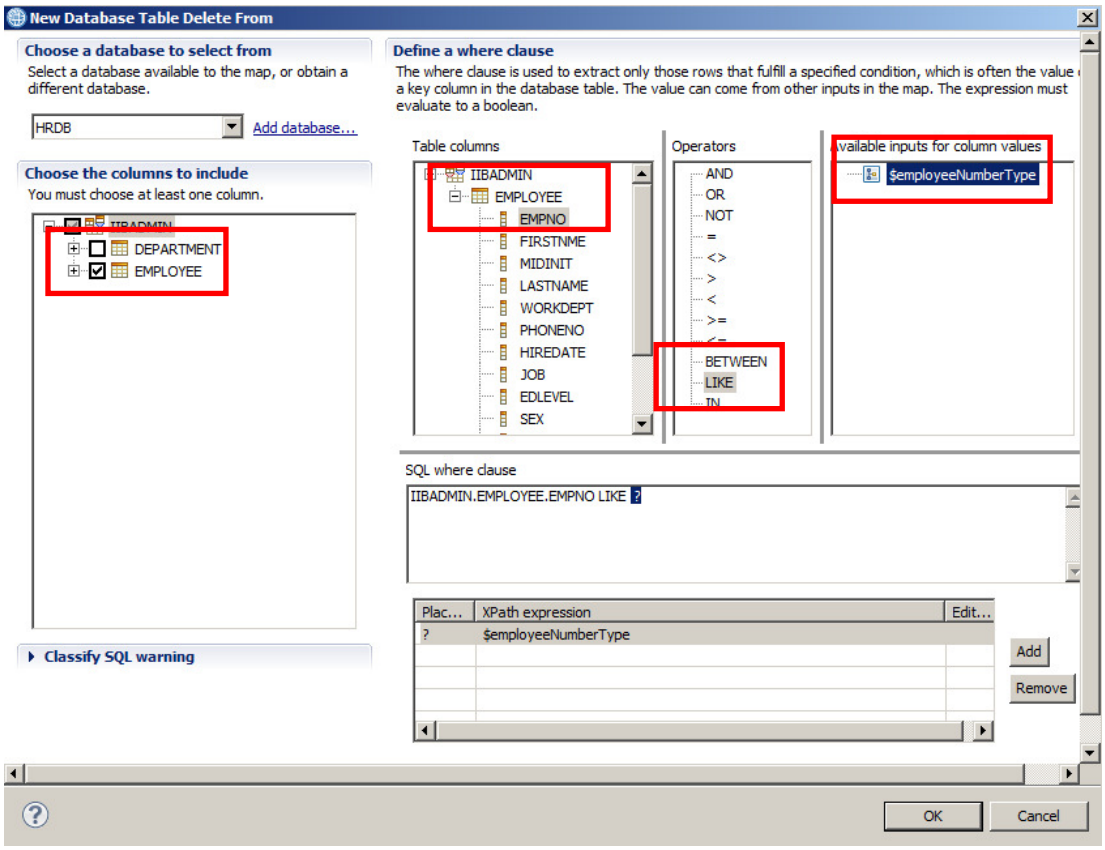
    3. Construct the SQL where clause:
       a. Double-click the EMPNO column
       b. Double-click the "LIKE" operator
       c. Double-click the $employeeNumberType element from the available inputs

As a check, the SQL clause that should be generated should be:

```
IIBADMIN.EMPLOYEE.EMPNO LIKE ?
```

And the generated XPath expression should be:

```
$employeeNumberType
```



8. Refine the XPath expression.

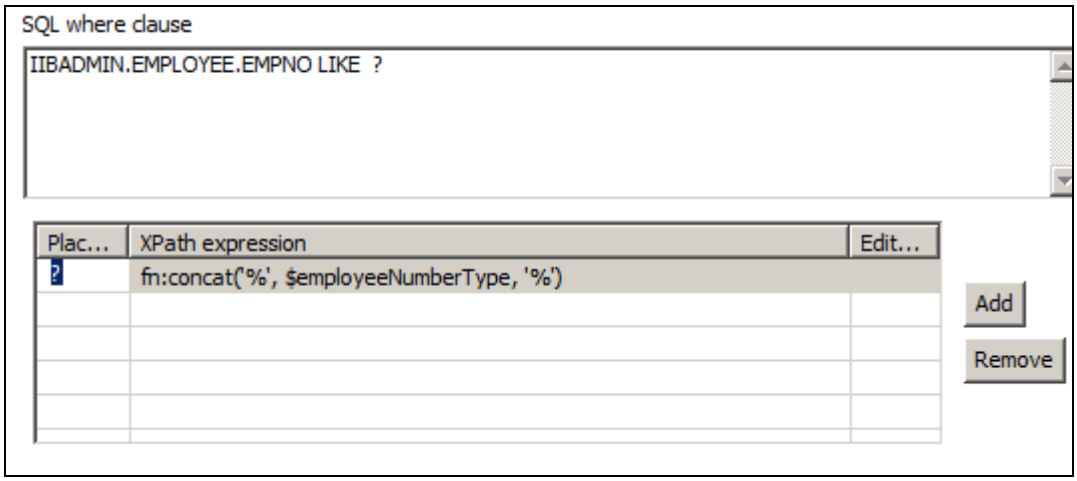
   The map will be built to retrieve all employee records that partially match the provided key. For example, if the provided key is "0020", we want the map to retrieve rows with a value of "000020", "00201", "00204", etc.

   This is done by extending the SQL LIKE statement, in conjunction with the "%" character, appended both as a prefix and a suffix. Hence the statement "SELECT xxxx LIKE %0020%" will achieve the result described above.

   To do this, edit the XPath expression as follows:

```
fn:concat('%', $employeeNumberType, '%')
```

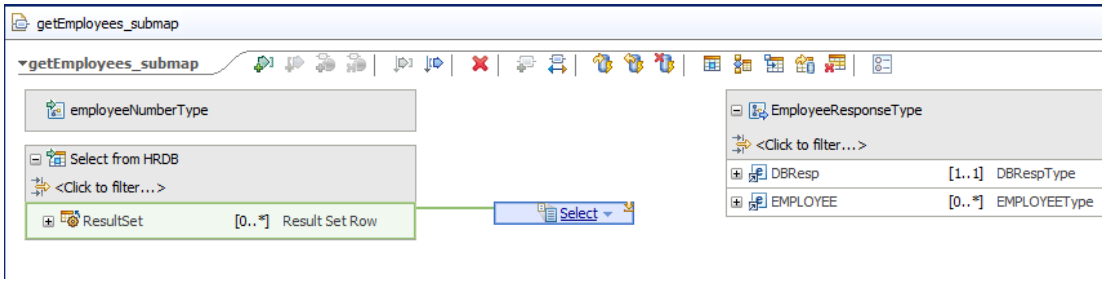
   The result should look like this:



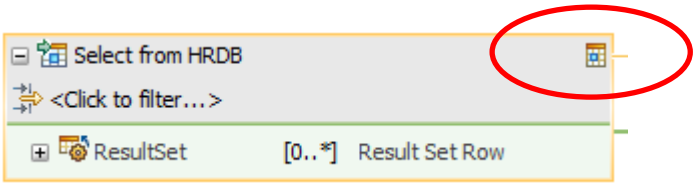
   Now double-check that you typed this XPath **exactly** as shown above !!

   Click OK to close the database definition window.

9. The map will be shown as follows. You will now see the Select input assembly showing in the map editor.

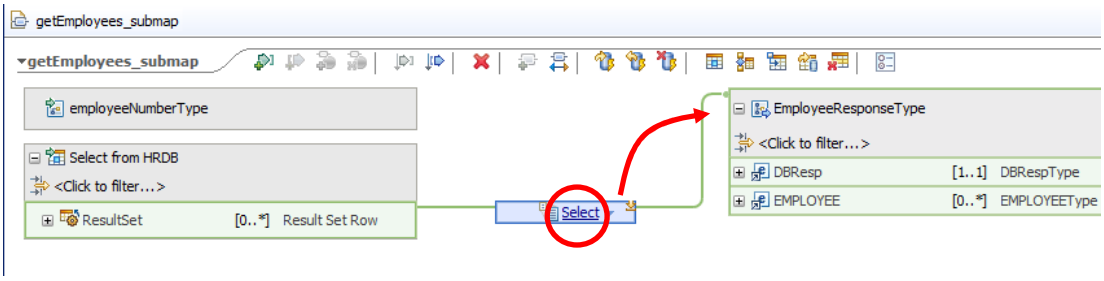


   (Note - you can return to refine the SQL statement by hovering over and selecting the database icon, as below).

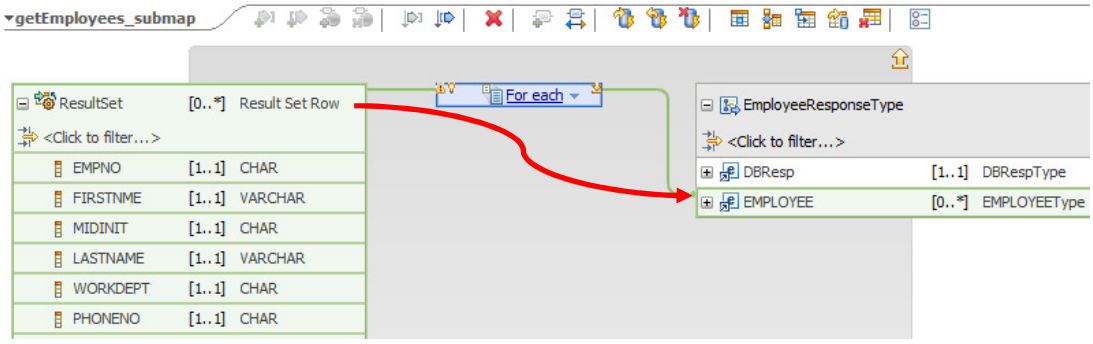


10. In the output message assembly, connect the Select transform to EmployeeResponseType.

    Then, click "Select" to enter the next logical level of the map.

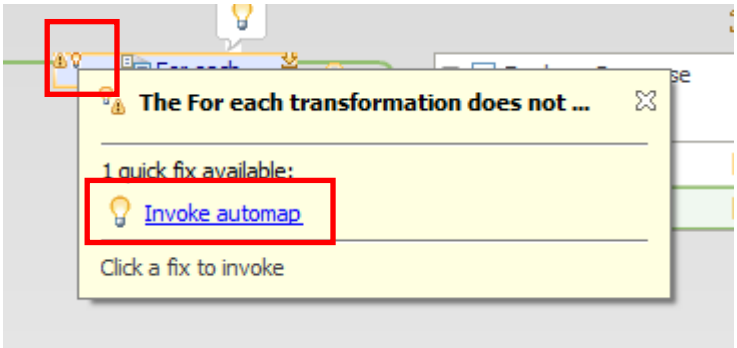


11. Connect ResultSet to EMPLOYEE in EmployeeResponseType. This will generate a "For each" transform.

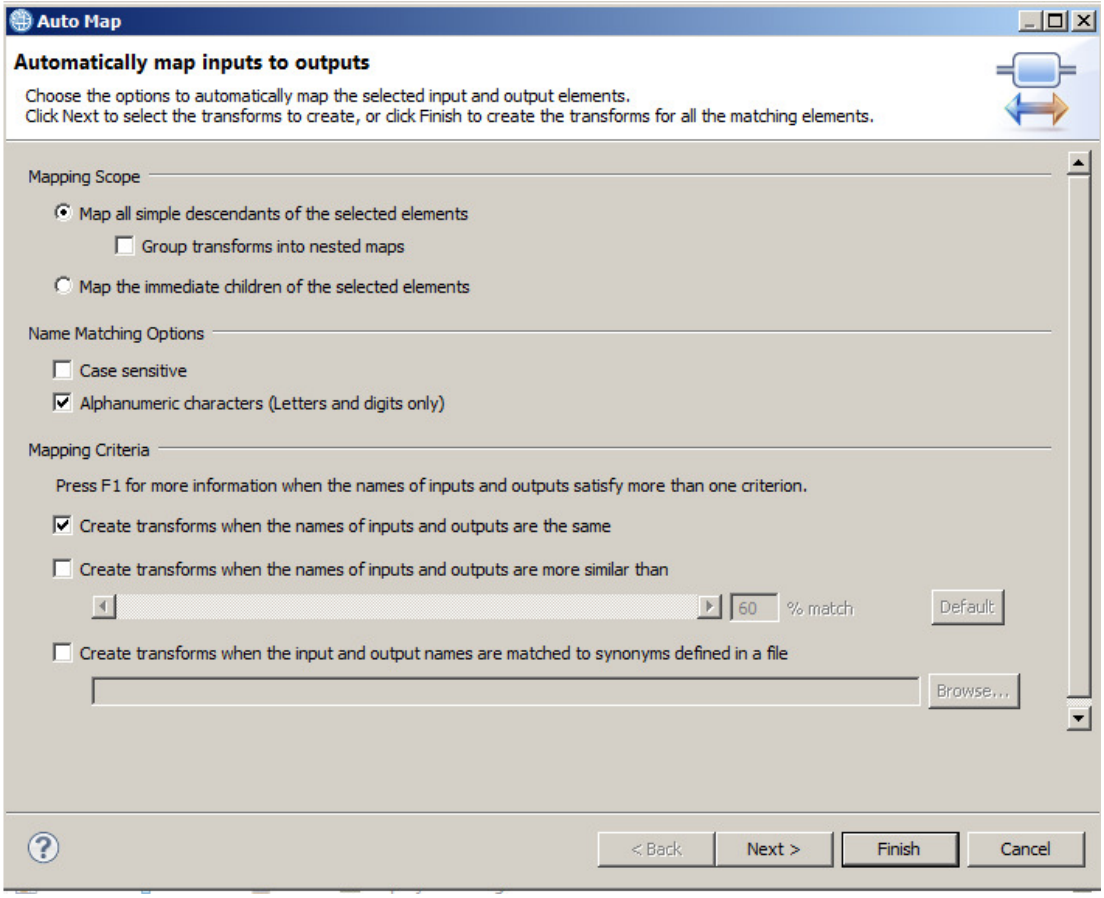


12. The "For each" transform will indicate a Warning, and suggest a "quick fix". Hover over the light-bulb on the transform, which will show the available "quick fix".

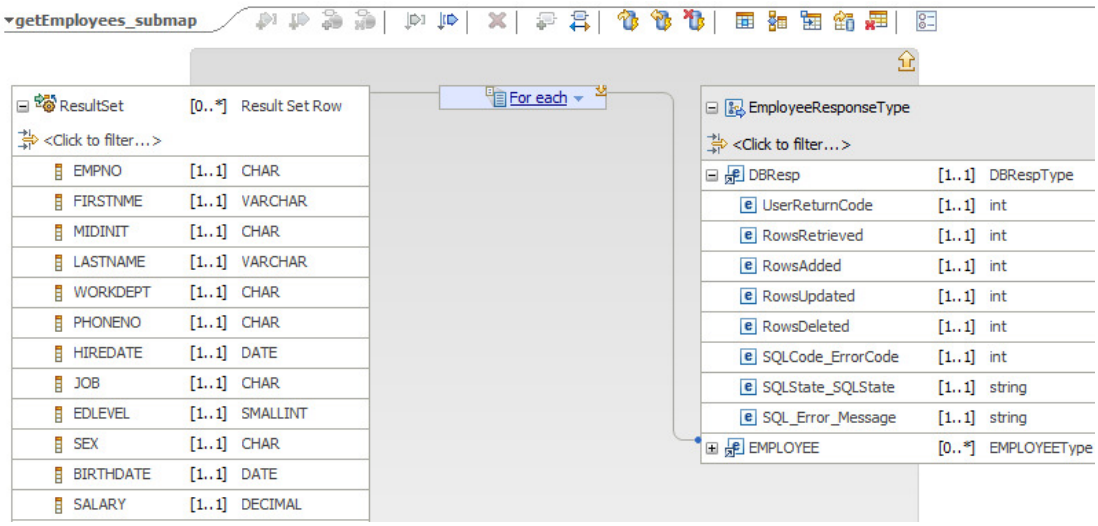
    .



13. Invoke the quick fix by clicking "Invoke automap". Since we simply want to map similarly-named elements, you can accept the defaults here, and click Finish.

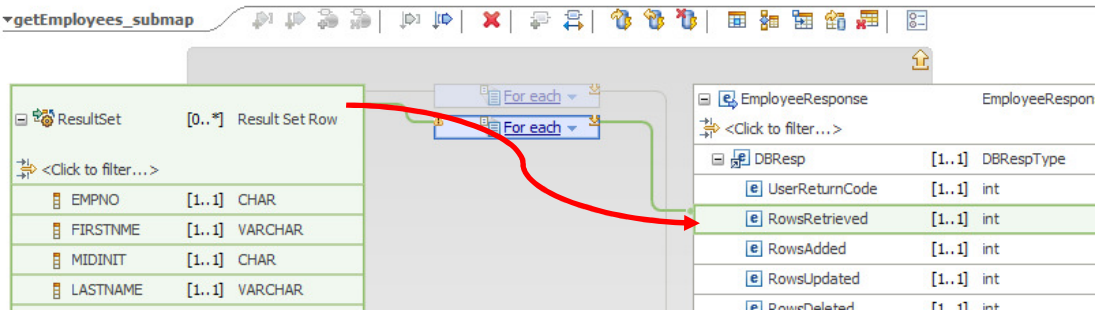


14. The Warning will have been resolved.

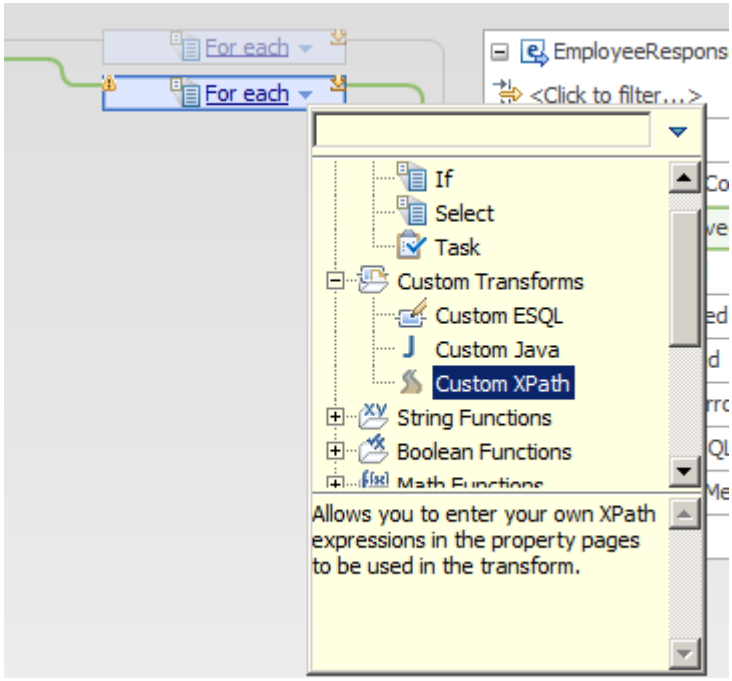


15. We want to set the value for RowsRetrieved.

    Expand DBResp, and connect the input ResultSet to the output DBResp/RowsRetrieved. This will initially create a "For each" transform; the transform will show a warning.



16. Click the drop-down arrow on the transform, and change the transform to "Custom XPath".



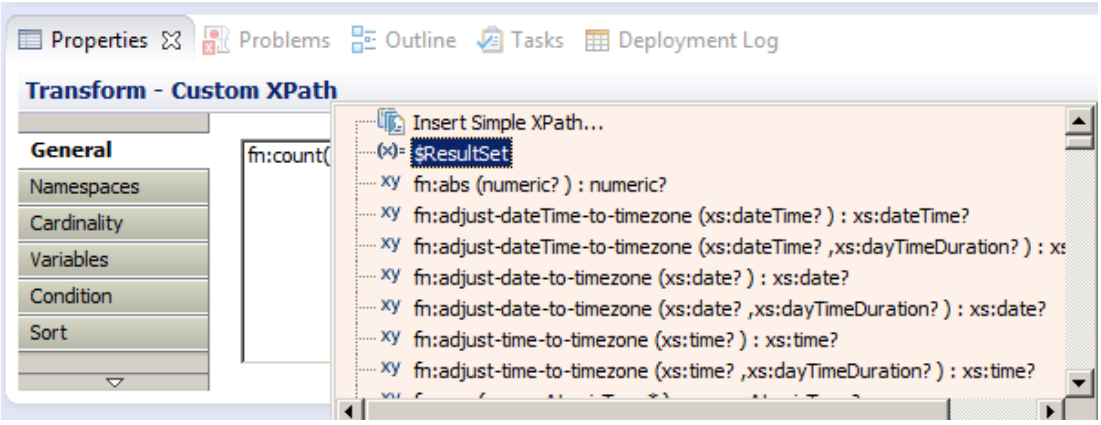
17. In the map editor, make sure the Custom XPath transform is selected.

    On the Properties tab for this transform, select "General" and type the following into the XPath editor:

    ```
    fn:count(
    ```

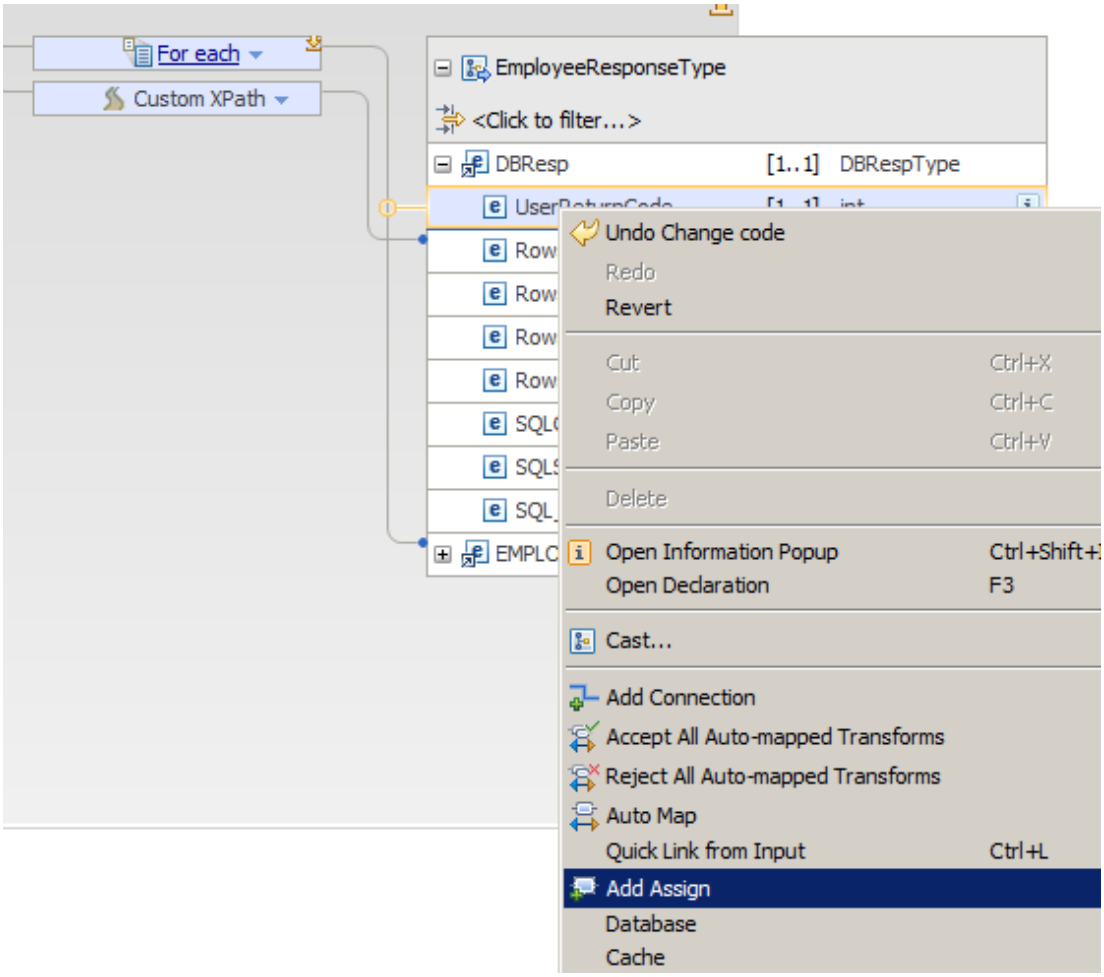
    Then, invoke the content assist function (Ctrl-space) which will show you the possible values for completion. Select the value that shows $ResultSet, or similar. Depending on whether you have made other editing changes, the value shown may be $ResultSet or have a suffixed number. The example shown below is $ResultSet.

    Complete the XPath expression with a ")". This expression will calculate the number of rows retrieved from the database.

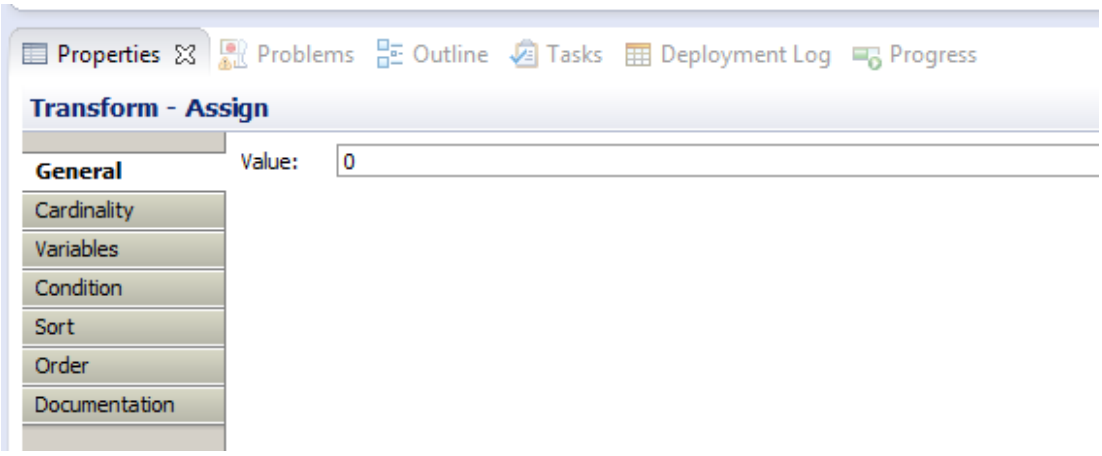


18. Set the value of UserReturnCode to 0. Do this by using an Assign transform.

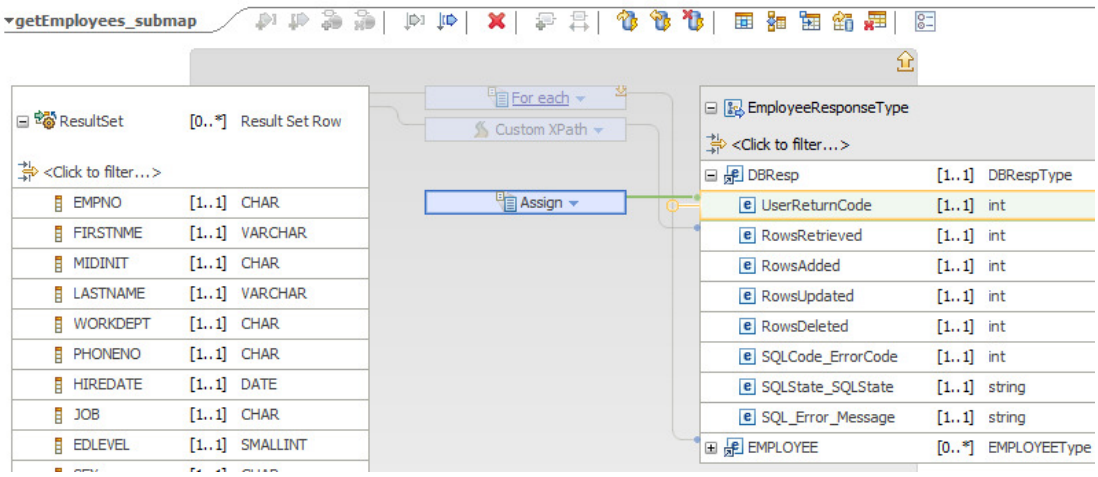
    Expand the output assembly DBResp, and right-click UserReturnCode. Select Add Assign.



19. The default value for an Assign is "0", so leave this unchanged.

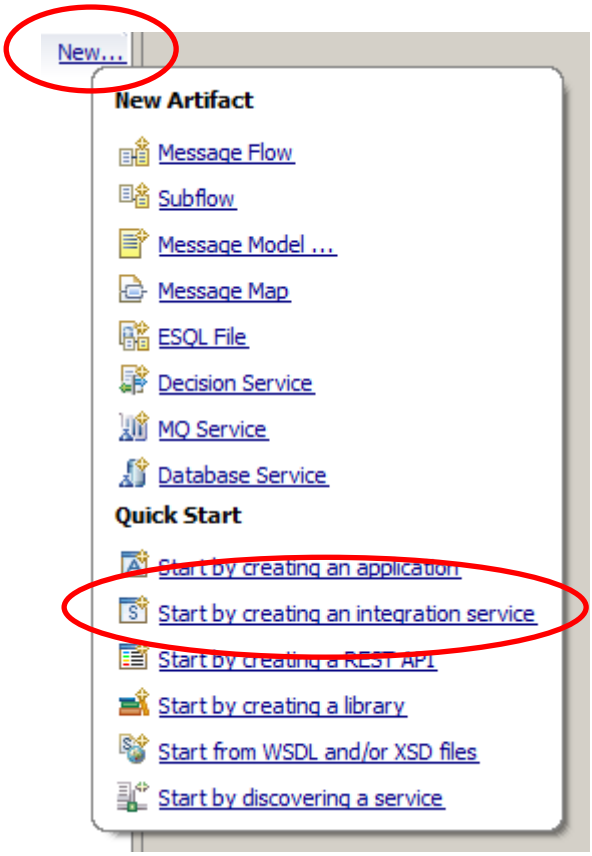


20. The submap will now look like this. The submap is complete, so save (Ctrl-S) and close the map.



3. Create the Integration Service

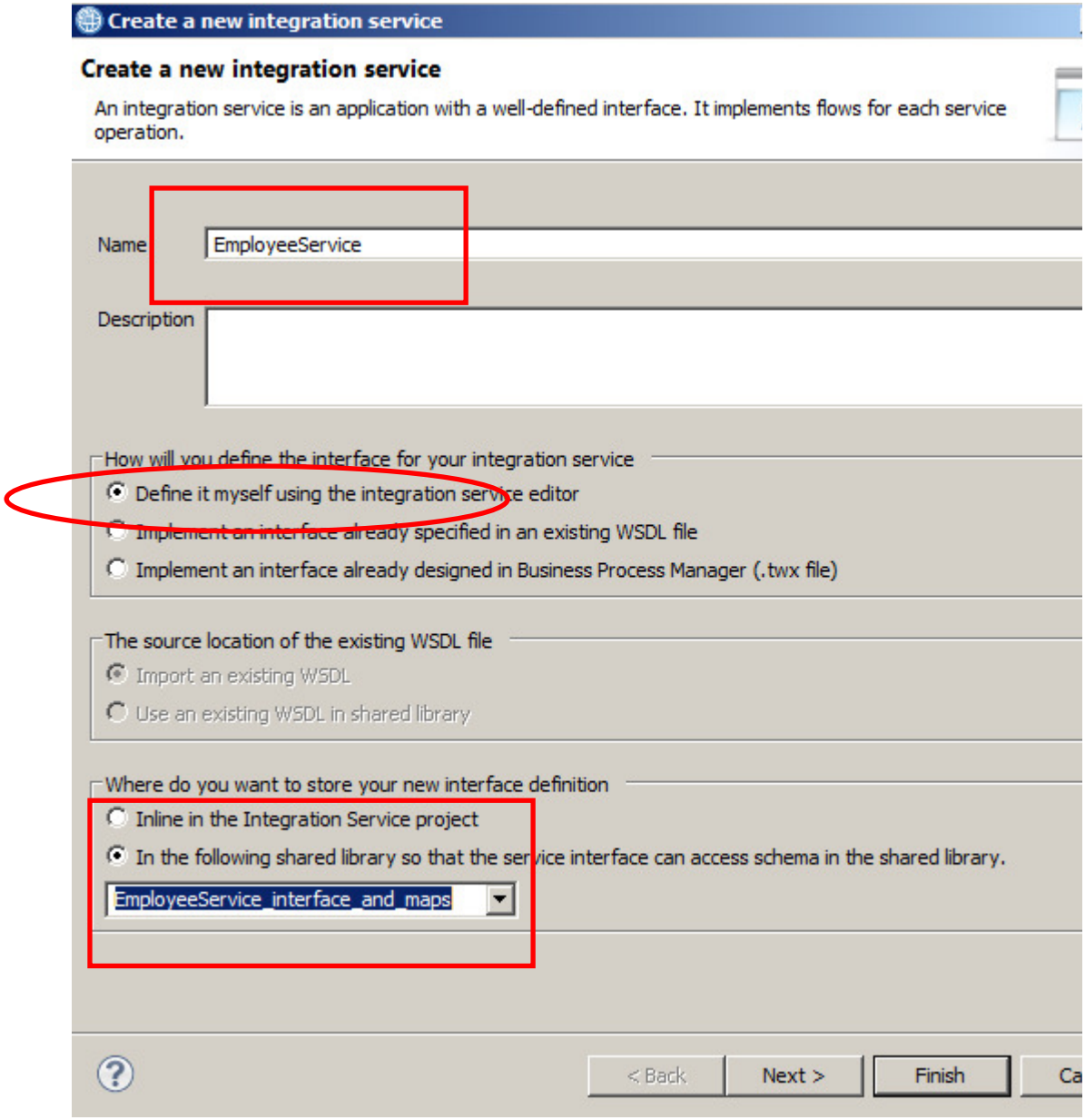
1. In the Application Development navigator, click New, and "Start by creating an integration service".



2. Name the service EmployeeService, and accept the default to "Define it myself ...".

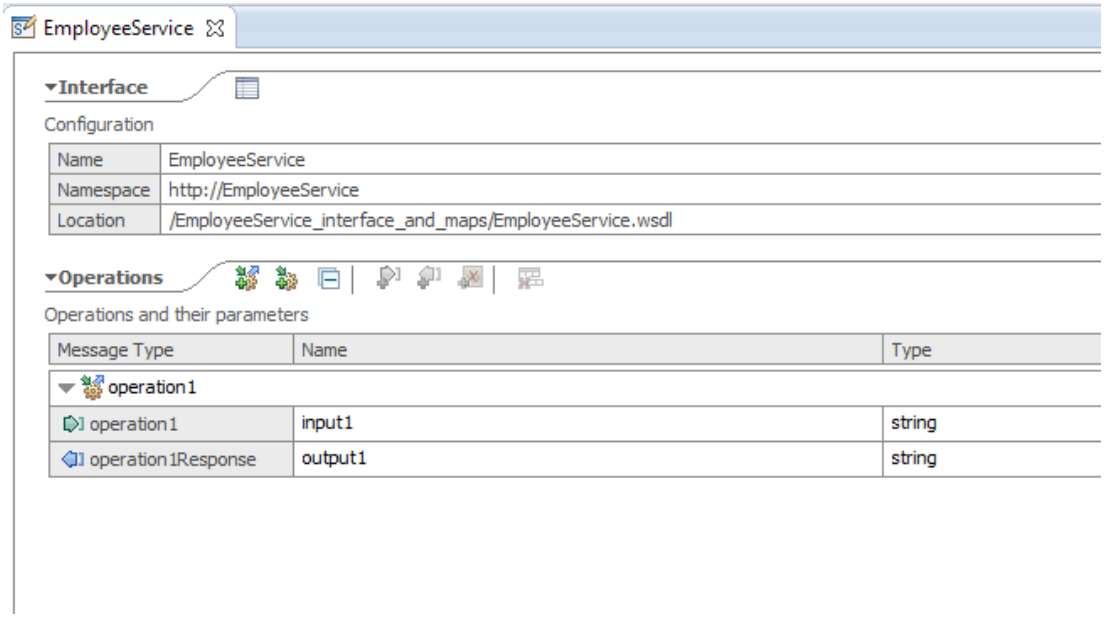
   Since the schemas will be defined in a Shared Library, you should choose the option to store the interface definition in a shared library. Select this option, and specify the name of the EmployeeService_interface_and_maps.

   Click Finish.



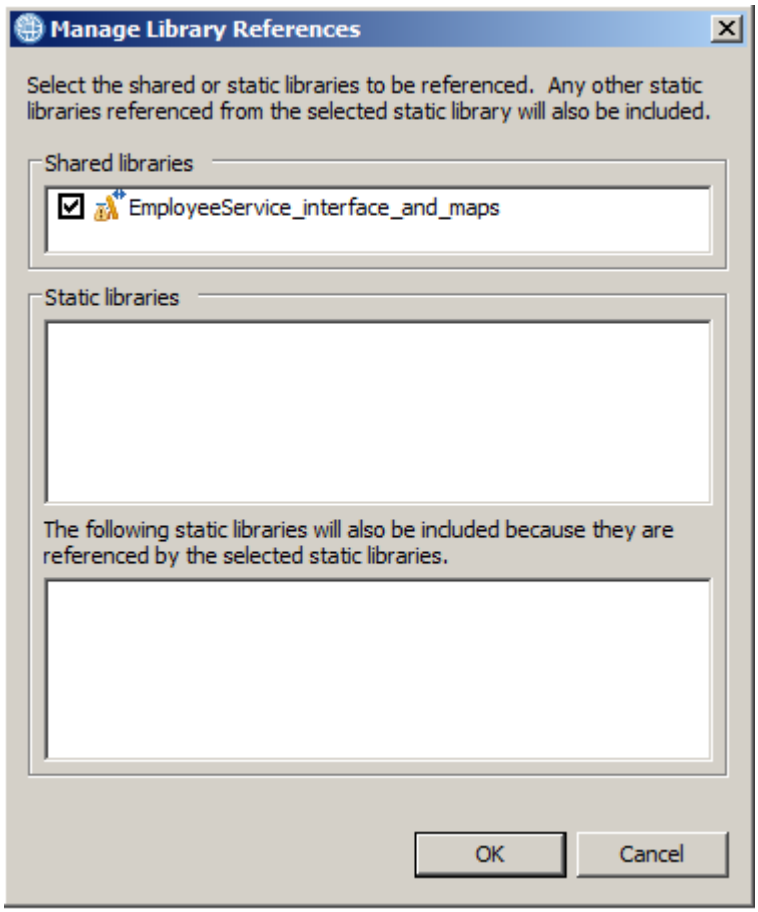
3. The following service editor will open and the following service interface will be generated. The service will contain a request/response operation named operation1.

   Note that the location of the service interface is EmployeeService_interface_and_maps, with the name of the wsdl = EmployeeService.wsdl.



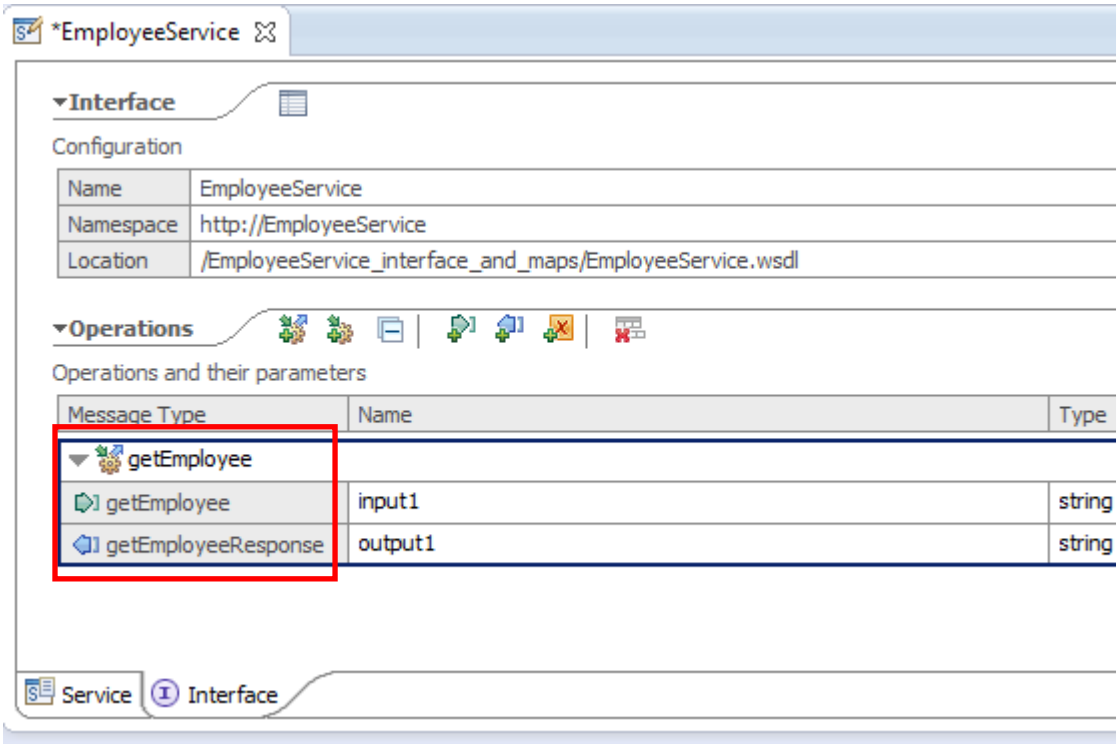
4. Note that the Library Reference of the new service has automatically been set to reference the Library where the service schemas and will be defined (right-click the new Integration Service and select Manage Library References).

   After reviewing, cancel this window.

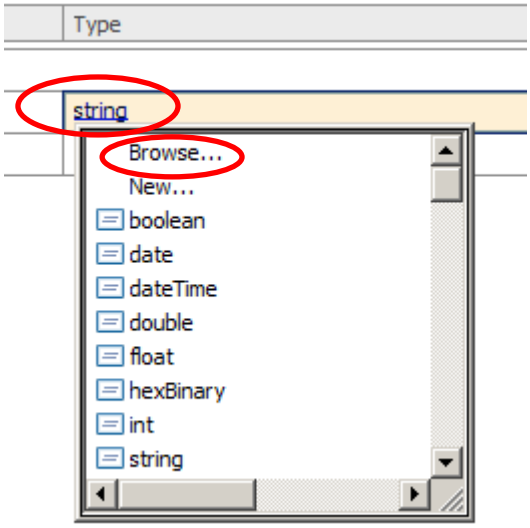


5. Change the name of operation1 to getEmployee:

   - Highlight operation1, and overtype with getEmployee
   - Press the Return key after renaming, which will automatically allocate the names of the message types for the operation

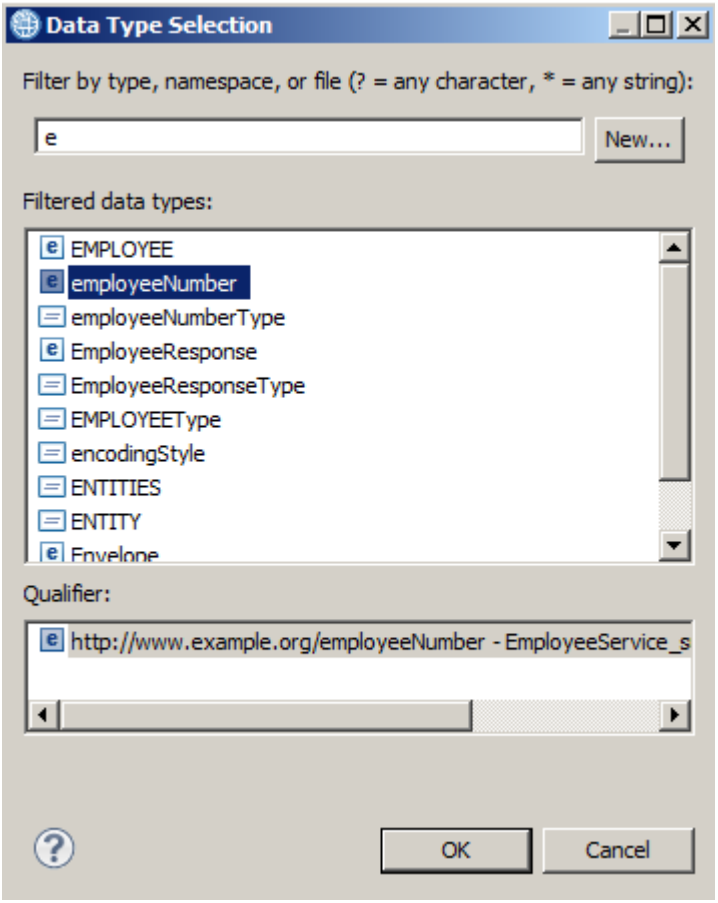


6. Change the Type of the input message to **employeeNumber**. (Click the current value "string"), then click Browse).

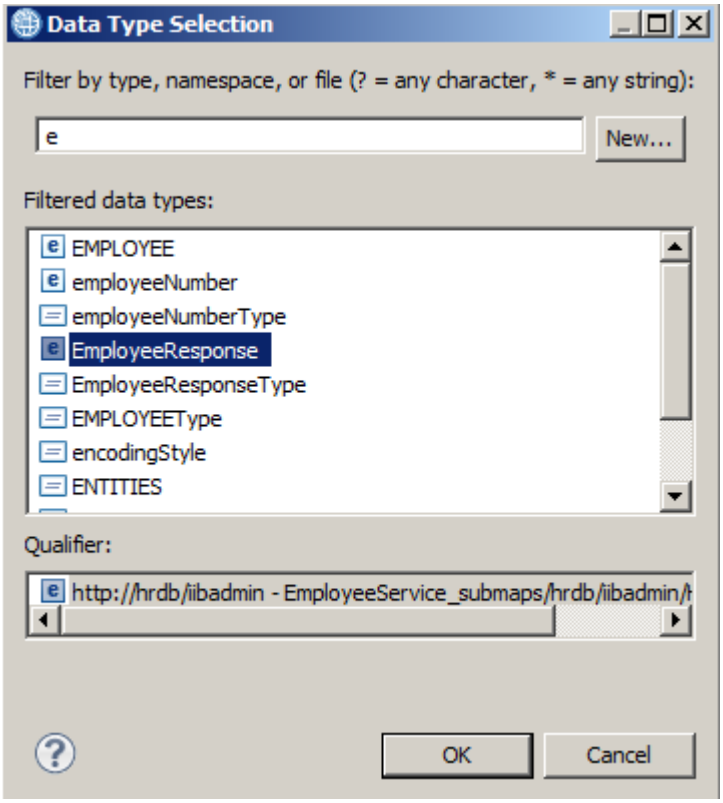


7. Type "e" into the filter, and select employeeNumber.

   Click OK.



8. For the output message, use the same technique and select **EmployeeResponse**.

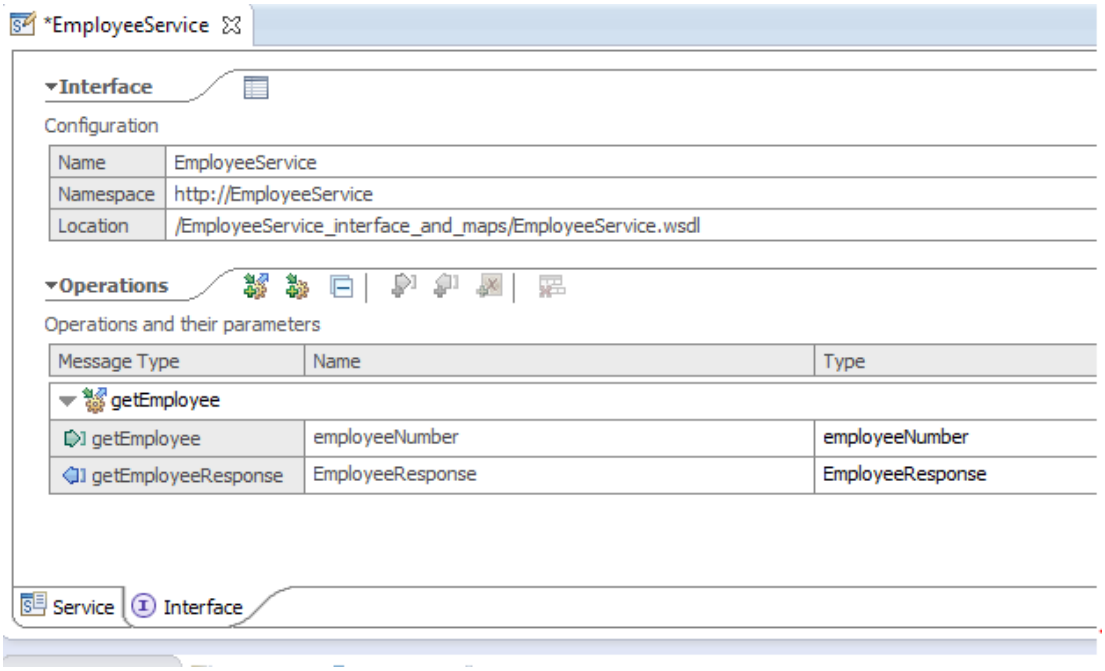


9. The getEmployee operation interface is now fully defined, and should look like this.

   Note that the name of the input message will have been changed to employeeNumber. This is because employeeNumber is an XSD element, and the "name" attribute is not permitted when using an XSD element.

   Similarly, for the output message, you have used an element (EmployeeResponse). You could have used the element type (EmployeeResponseType), which would not have changed the name of the operation types.

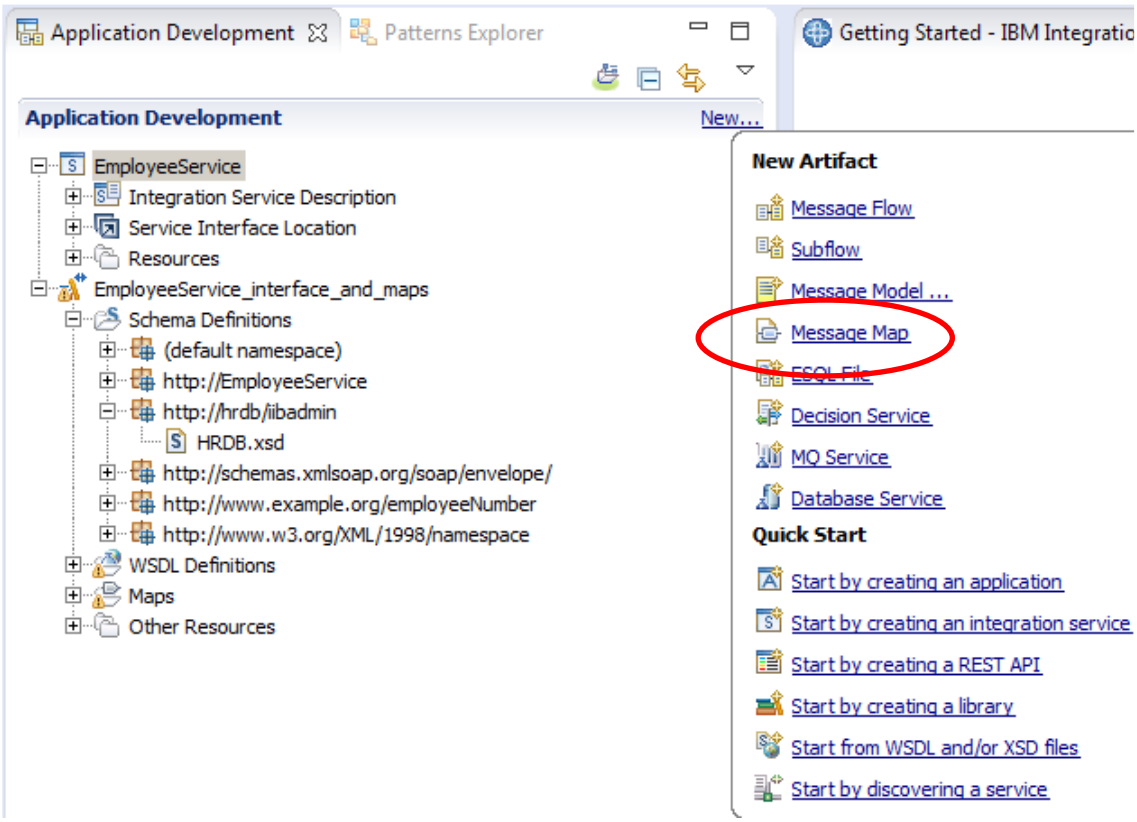
   Save the service at this point (Ctrl-S).



3.1 Implement the Main Map

You now need to create the main map that will invoke the submap. The main map will be aware of the type of protocol that invokes it (in this case an integration service). It will extract the payload of the message (employeeNumber), and pass this to the submap to access the database. The main map will be located in the shared library EmployeeService_interface_and_maps.

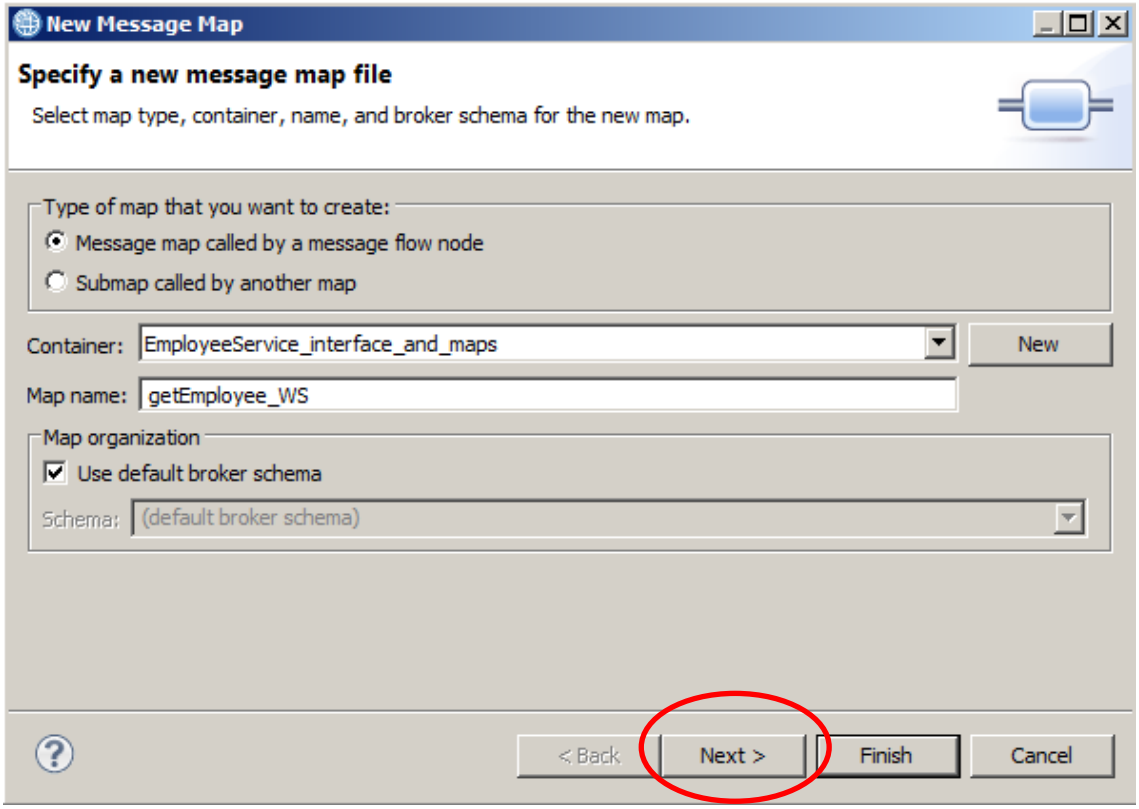
1. Highlight the EmployeeService_interface_and_maps library, and click New, Message Map.



2. Ensure the container is correctly identified.

   Name the new map getEmployee_WS

   Click Next.

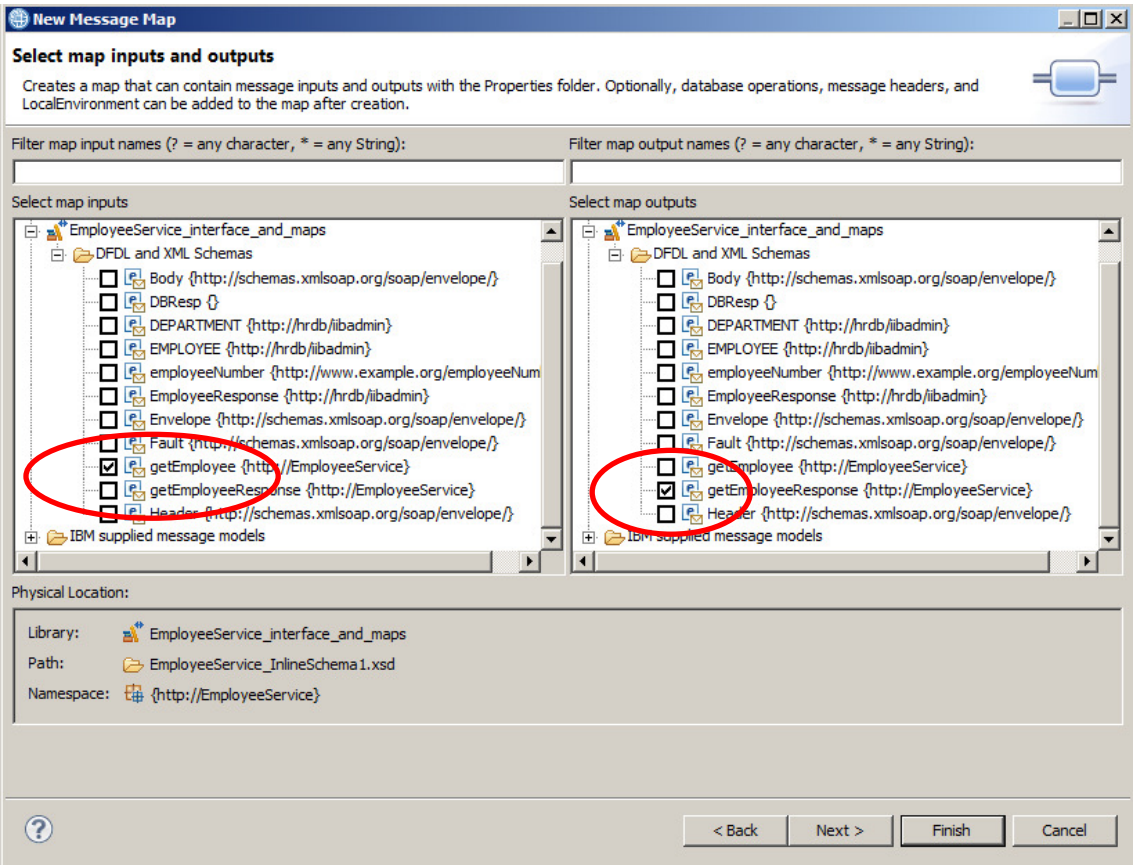


3. For both the map input and output, expand EmployeeService_interface_and_maps, and expand DFDL and XML Schemas.

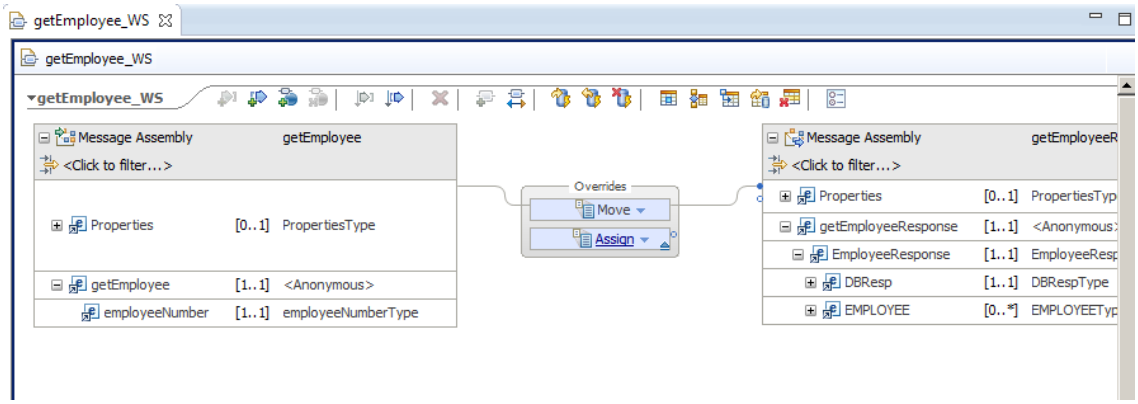
   - For the input, select **getEmployee** (remember this originates from the input message of the integration service, and resolves to element employeeNumber).

   - For the output, select **getEmployeeResponse** (resolves to element EmployeeResponse).

   Click Finish.

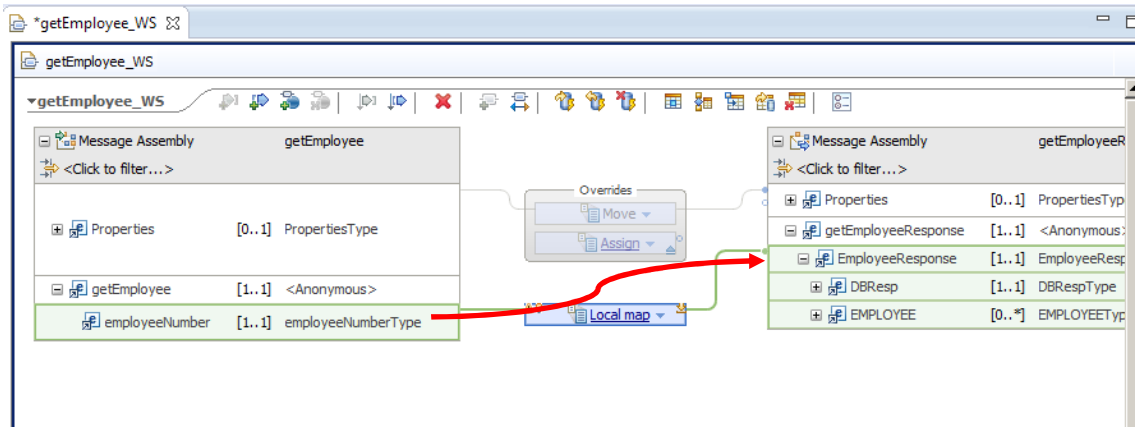


4. The map editor will open. Expand both the input and output assemblies.

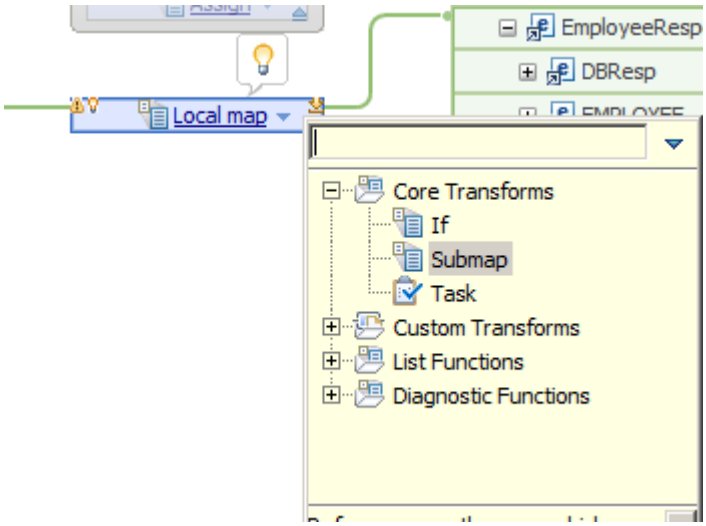


5. Connect the input **employeeNumber** to the output **EmployeeResponse**. This will produce a Local Map.

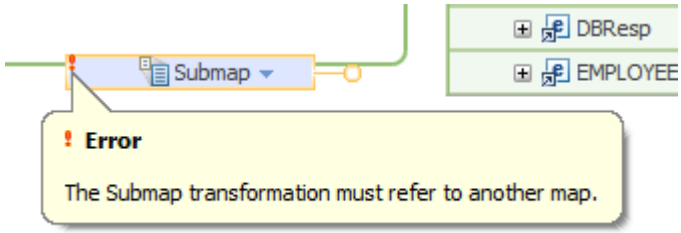
   A warning will indicate that there are no valid element mappings. We will resolve this by changing the Local Map to a Submap.



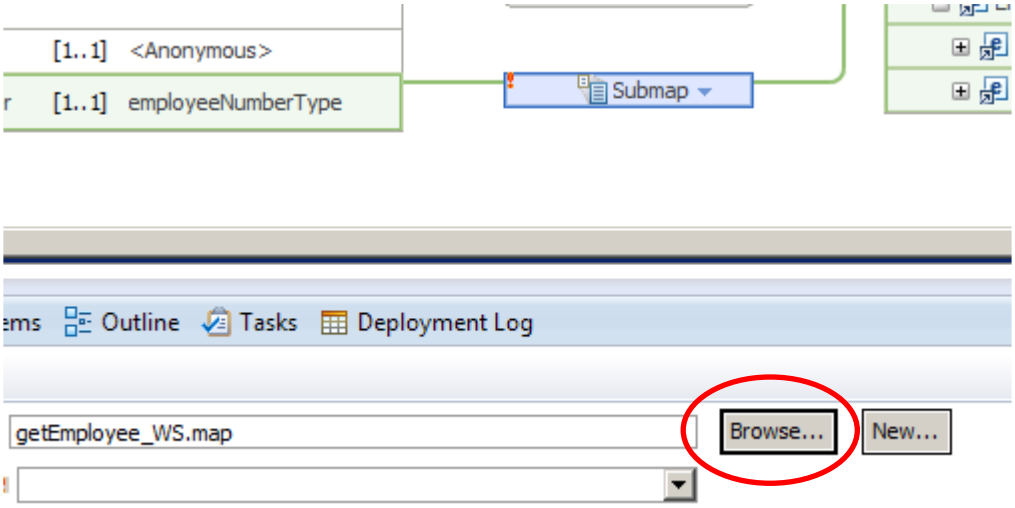
6. Change the Local Map to a Submap (click on the blue arrow).



The submap will show an error, since you haven't yet provided the name of the submap.

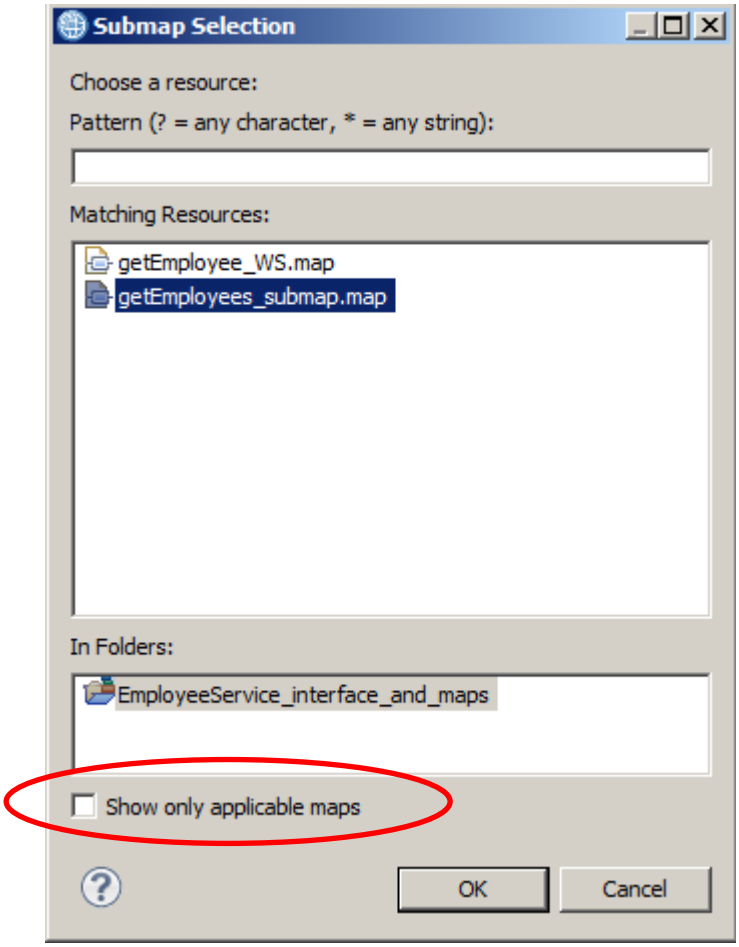


7. Highlight the submap transform, and locate the Properties of the transform.

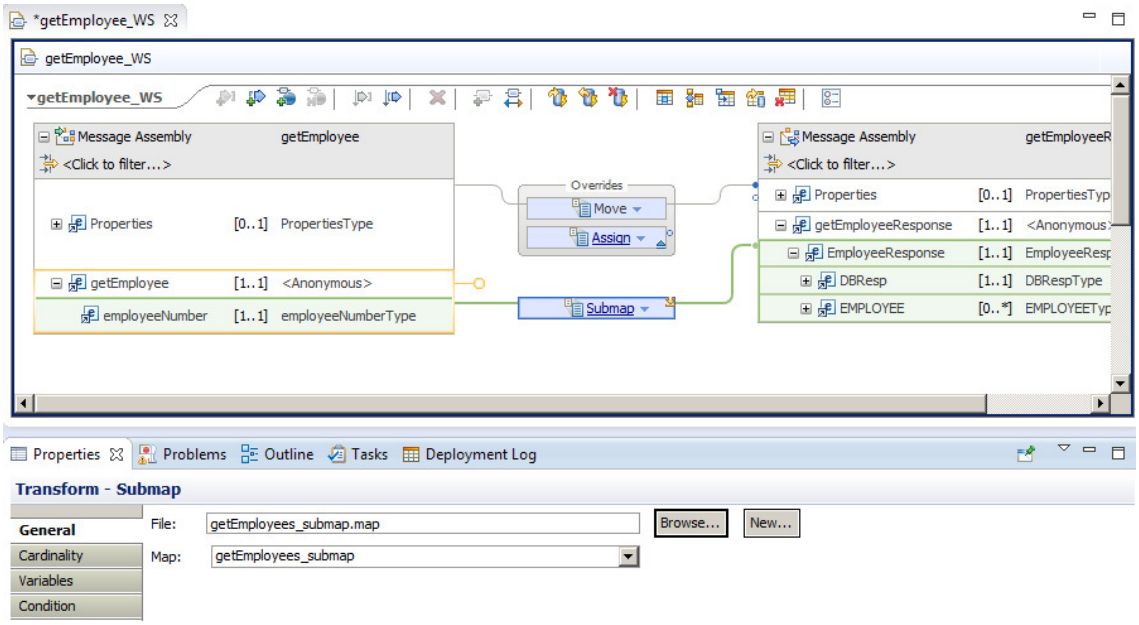


8. Click Browse, and select the **getEmployees**.submap located in the EmployeeService_interface_and_maps library. Click OK.

   Note - the map selection wizard will initially only show maps or submaps that are available for use in the current context. The screen capture below has unchecked this option, enabling you to see all maps and submaps, even though not all of these choices would make sense in the current context.



9. The main map is now complete, so save (Ctrl-S) and close.

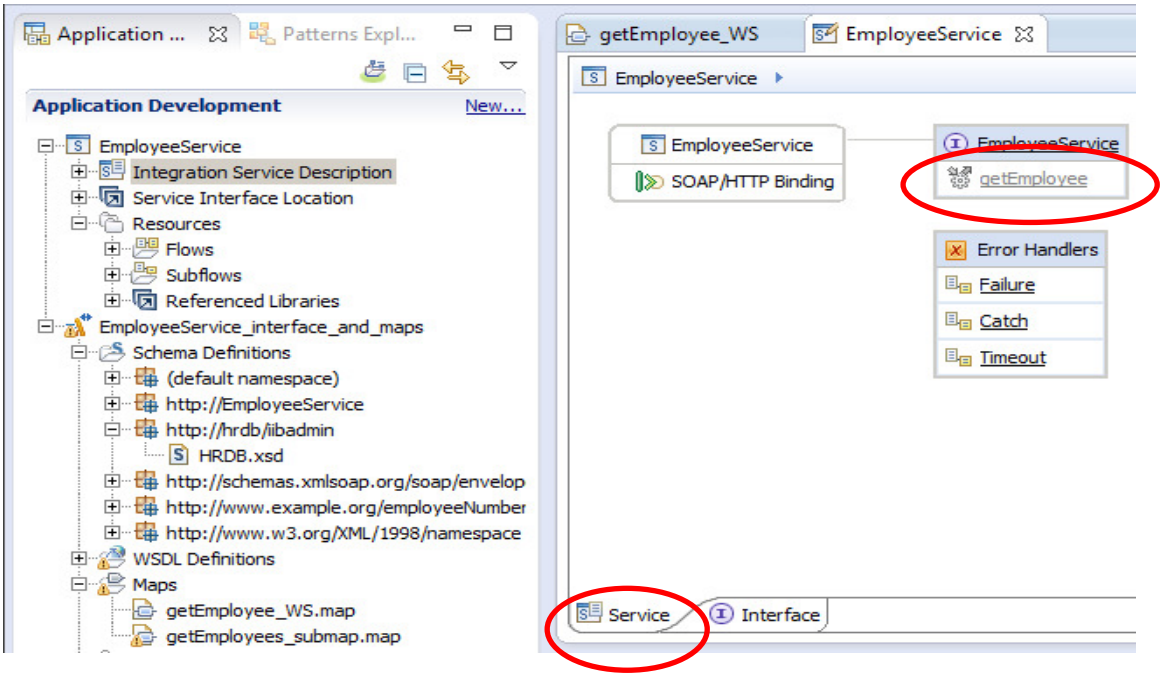


3.2 Implement the getEmployee Operation

You now need to complete the implementation of EmployeeService, so return to the service editor.

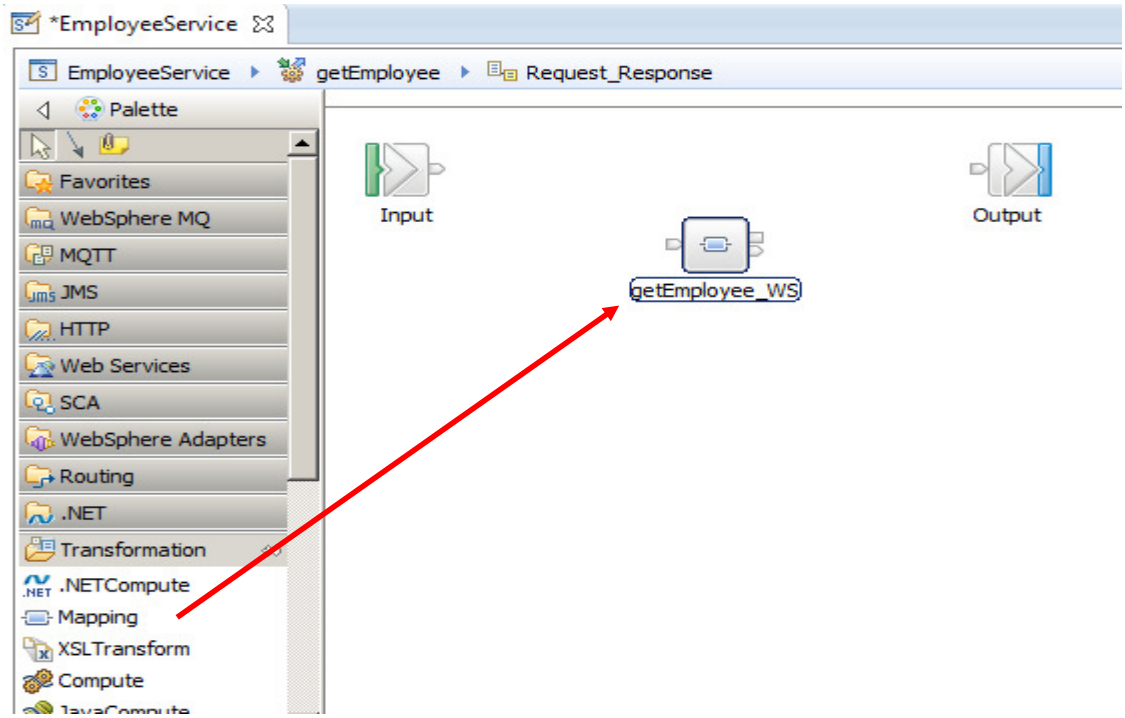
1. In the EmployeeService, click the Service tab.

   You will see that the getEmployee operation is not implemented, so is greyed out. Click getEmployee to implement it.

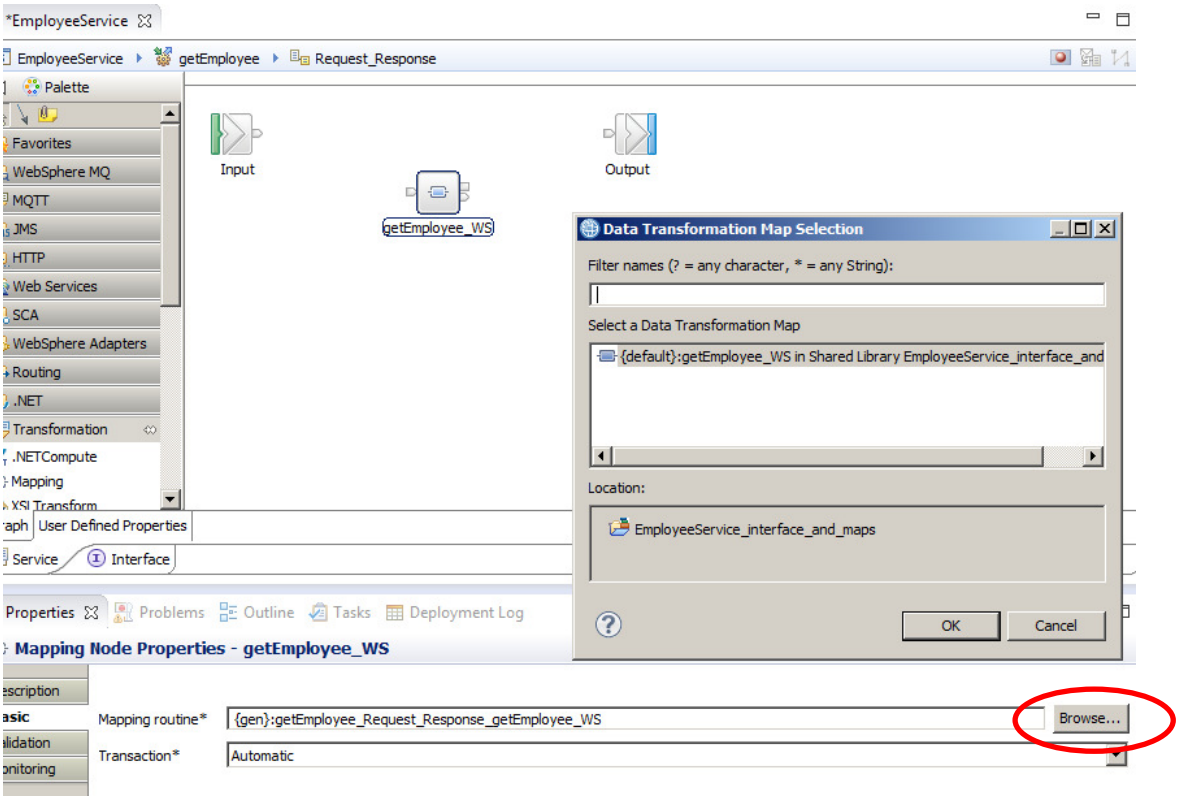


2. From the Transformation folder, drop a Mapping Node onto the flow editor.

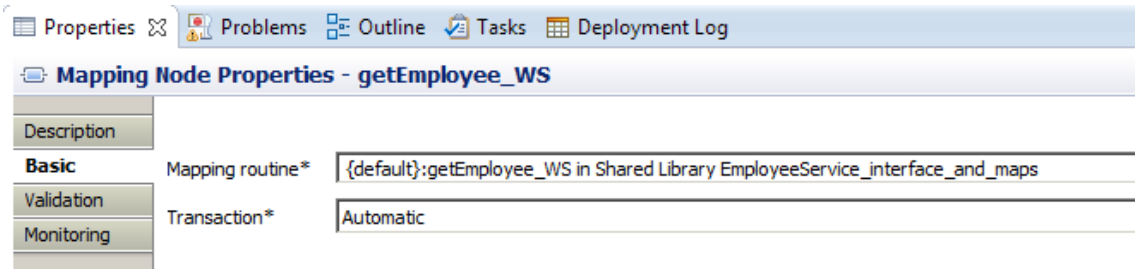
   Name it getEmployee_WS.



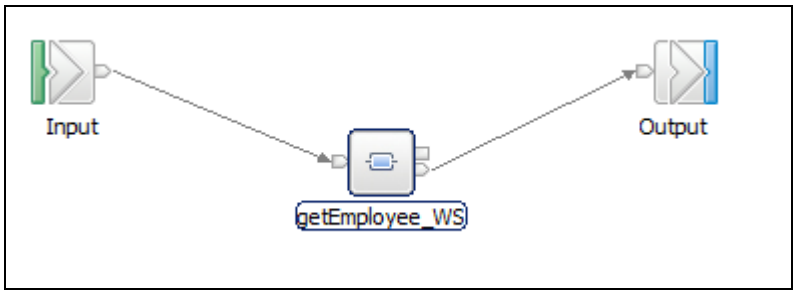
3. In the Properties (Basic) of the new Mapping Node, click Browse to select the map you created earlier. Select the getEmployee_WS map in the Shared Library, and click OK.



4. The new map node properties will show the map in the Shared Library.



5. Connect the nodes as shown.



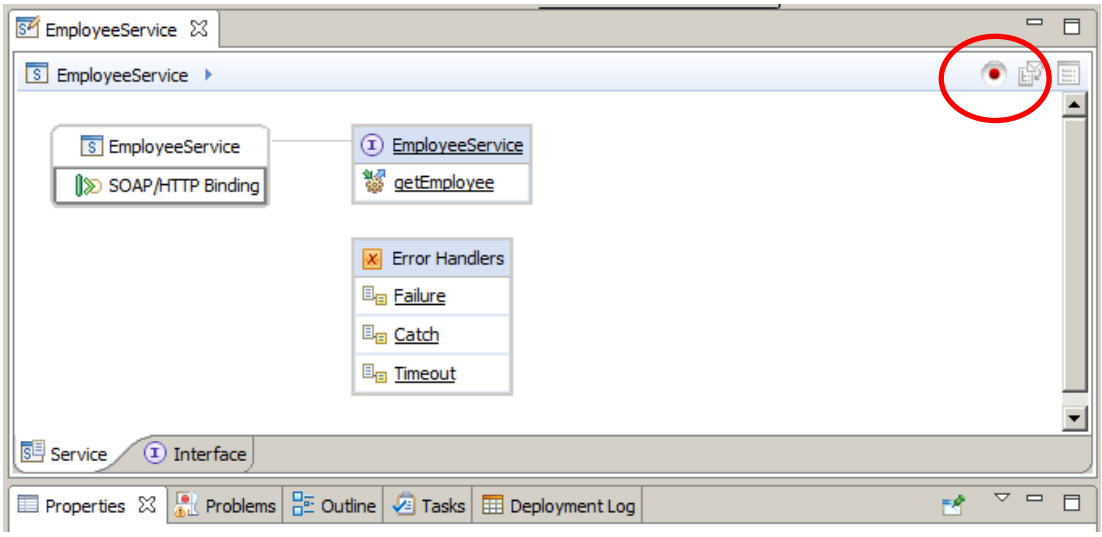
Save the flow.

4. Test the Integration Service with the Flow Exerciser

The Flow Exerciser is a new function in IIB v10. You will use this to perform a simple unit test of the service you have just developed. You will see additional Flow Exerciser functions in later labs.

First, make sure that the node TESTNODE_iibuser is started. You can check this in the Toolkit Integration Nodes pane. If it isn't started, right-click and select Start.

1. In the service editor, make sure the Service tab is selected. You will see a red button in the top right of the editor pane.

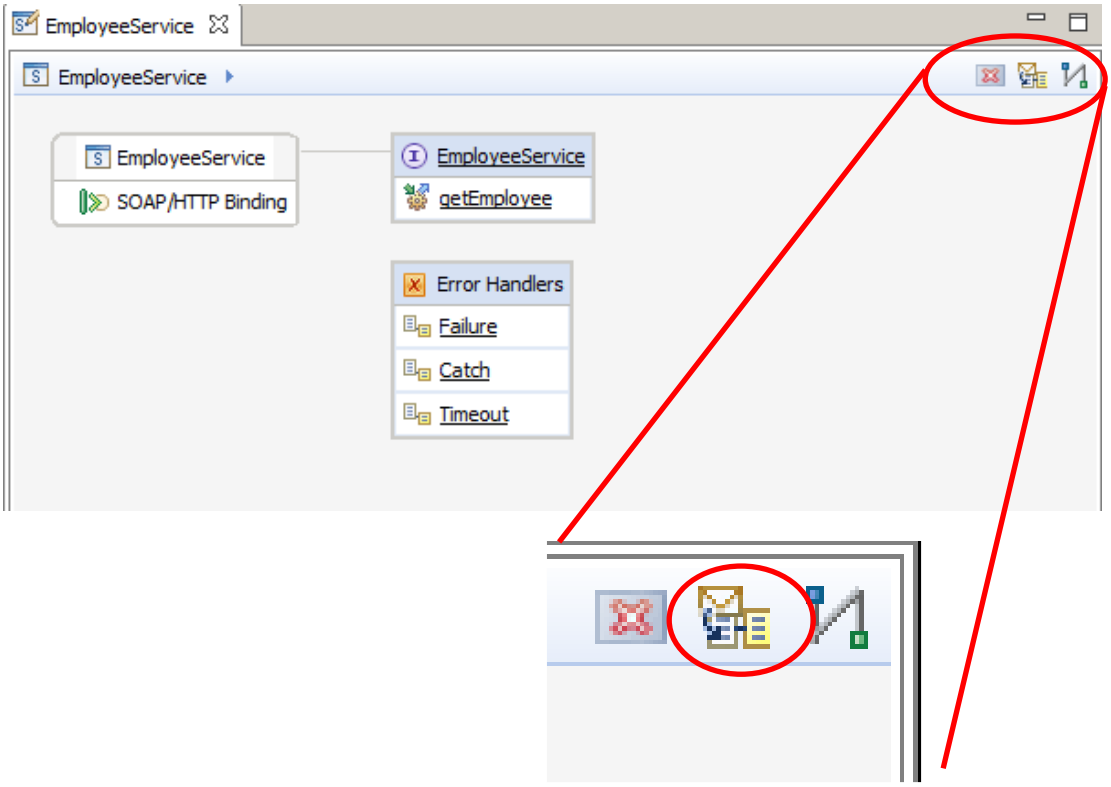


2. Click the red button. The integration service will be deployed to the TESTNODE/default. Note that if you have other nodes or servers running, you will be presented with a selection dialogue.

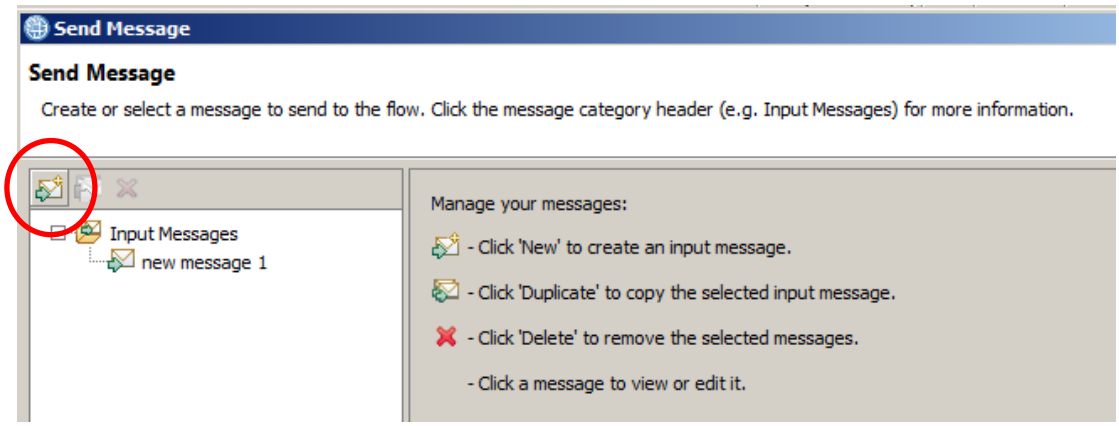
   When the service is deployed, you will see an information message. After reading this, select the box to not see in the future, and click Close.

3. The flow editor will turn grey, indicating that the Flow Exerciser is ready for use.

   In the top right, the available icons will have changed. Select the "Send message" icon (the middle icon).



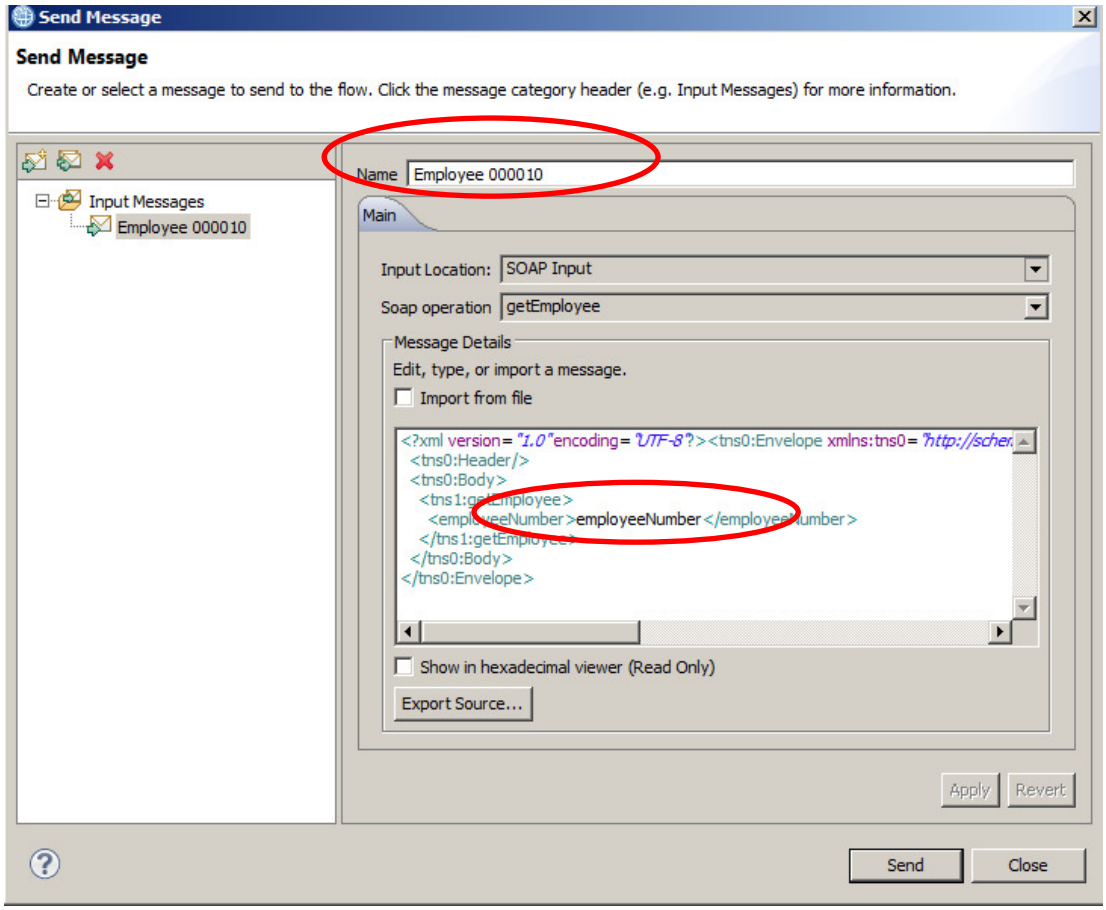
4. In the Send Message window, click the button to create a new message.



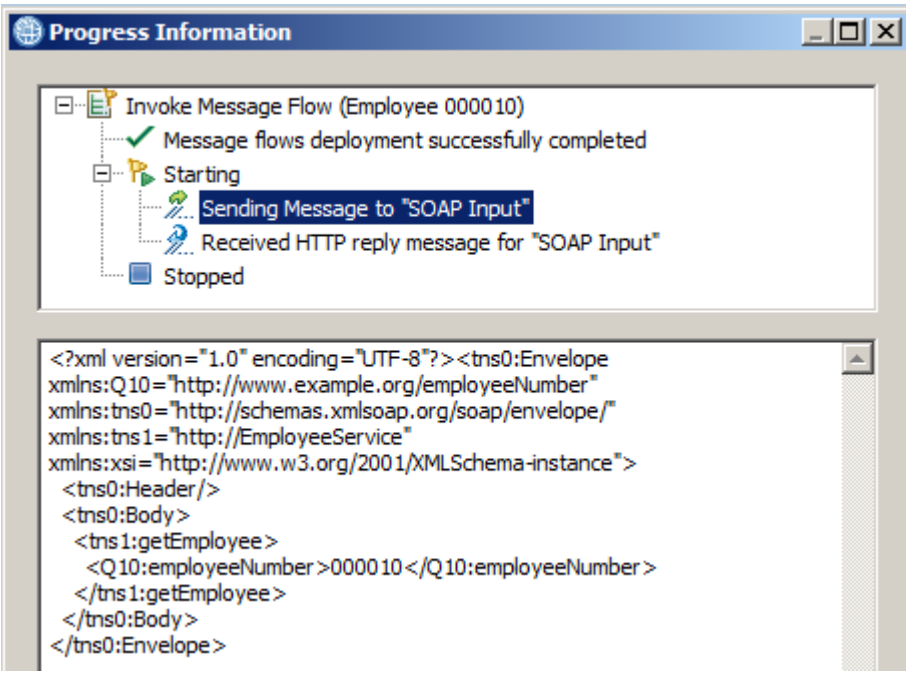
5. Name the new message "Employee 000010".

   The Flow Exerciser will have automatically populated a template input message, based on the WSDL and associated service operation input message.

   In the employeeNumber element, change the value to 000010, and click Send.

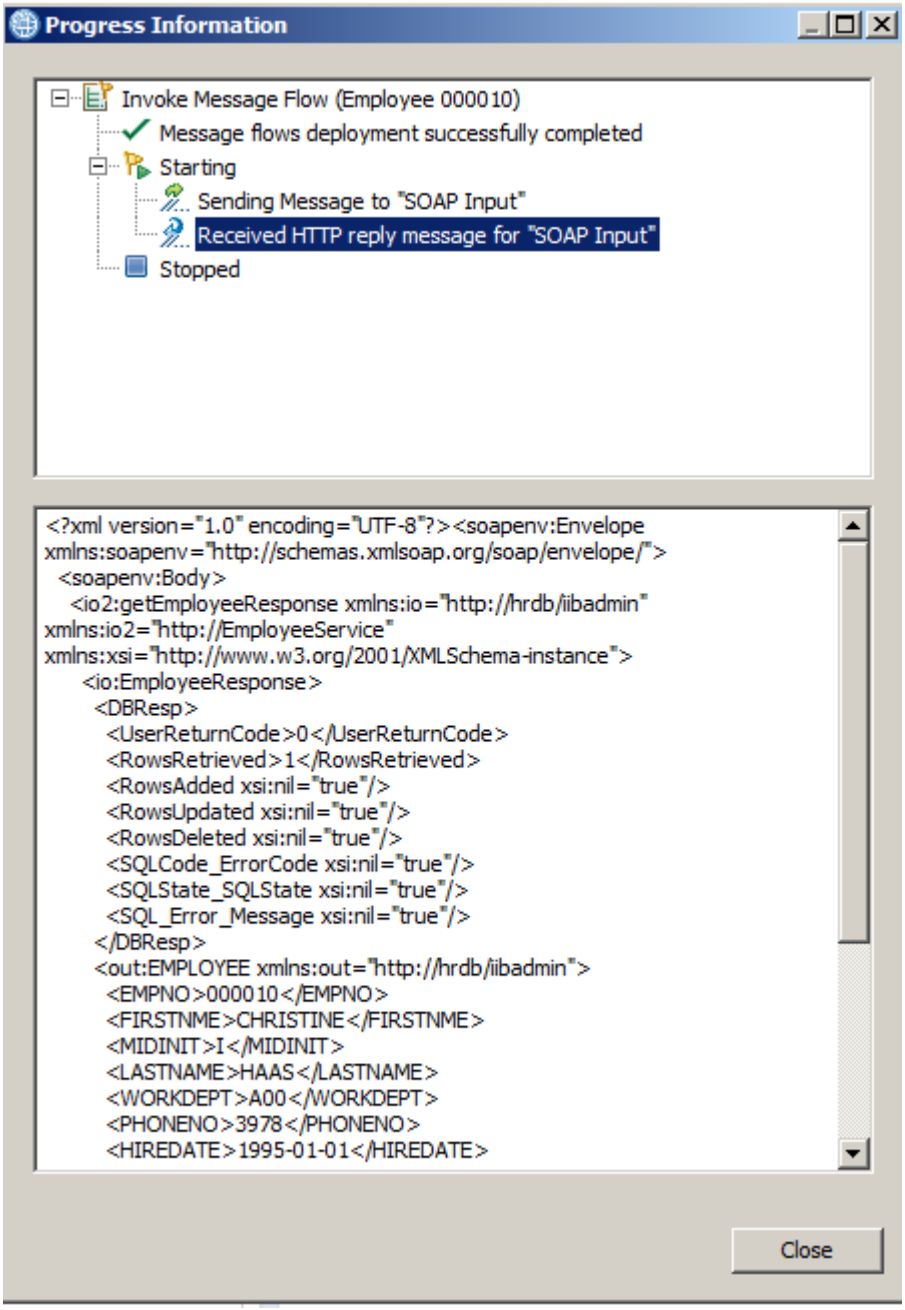


6. The service operation will be executed. Highlighting the "Sending Message" line will show the message that was sent to the service.



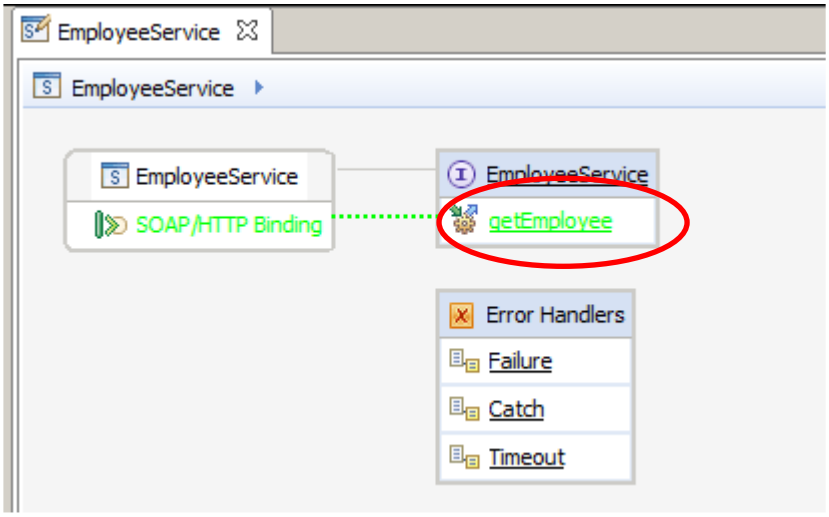
7. Highlighting the "Received HTTP" line will show the message returned from the service. Note that the employee record for the selected employee number has been retrieved from the EMPLOYEE table.

   Also note the UserReturnCode = 0, and RowsRetrieved = 1.

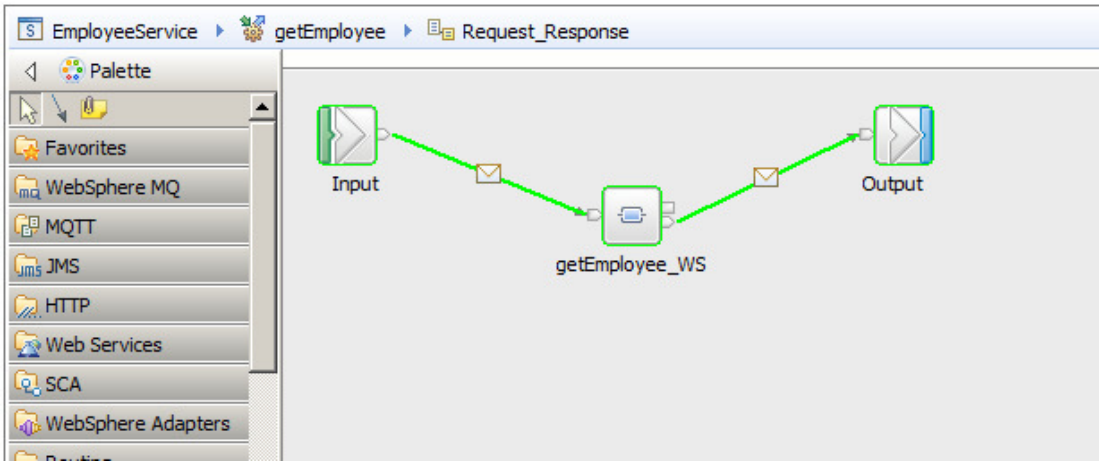


Close the Progress Information window.

8. In the flow editor, you will now see that some parts of the service definition have turned green. Click the "getEmployee" operation name.



9. In the flow editor, you will see the subflow that implements this operation. You will see that the connectors are green, indicating that the message flowed down this path.

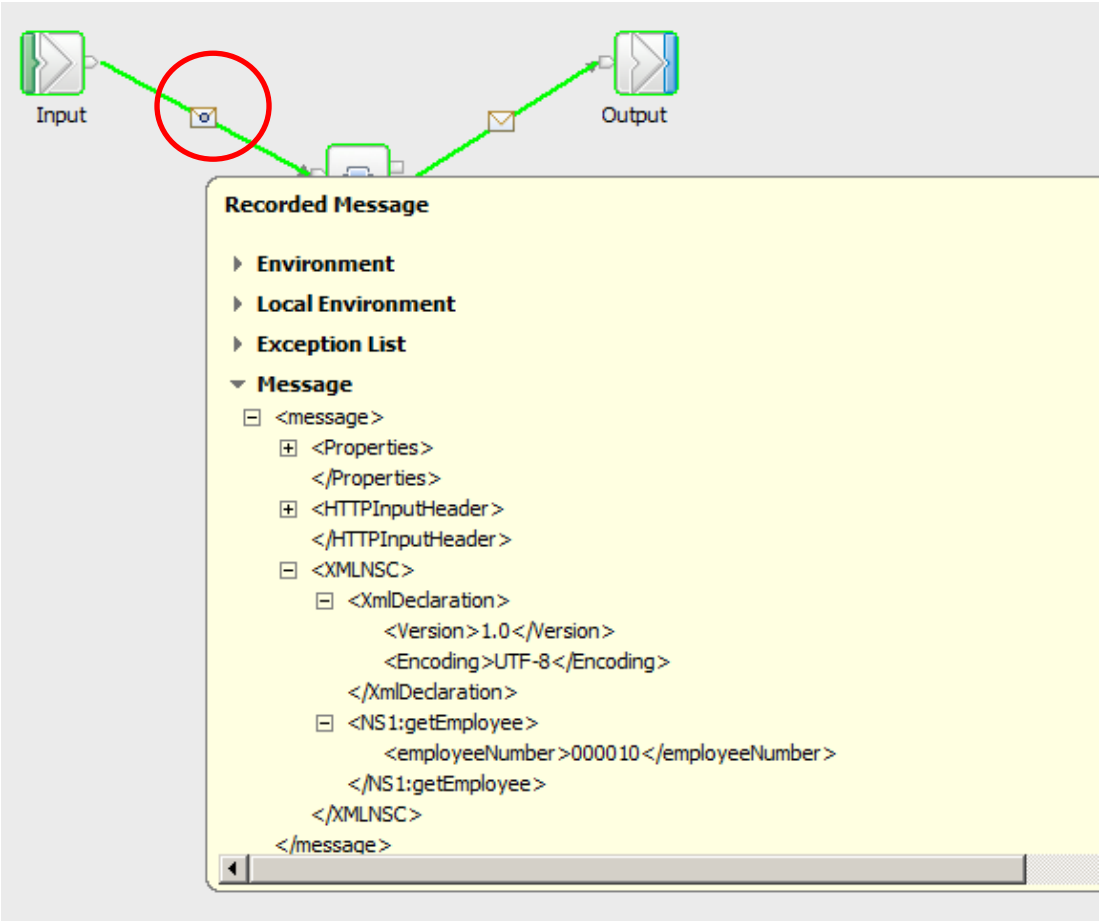


10. Click the icon on the first connector.

    The message tree that was current at the time it passed through this connector will be retrieved and displayed. Note that this is the full message tree, not just the user data.

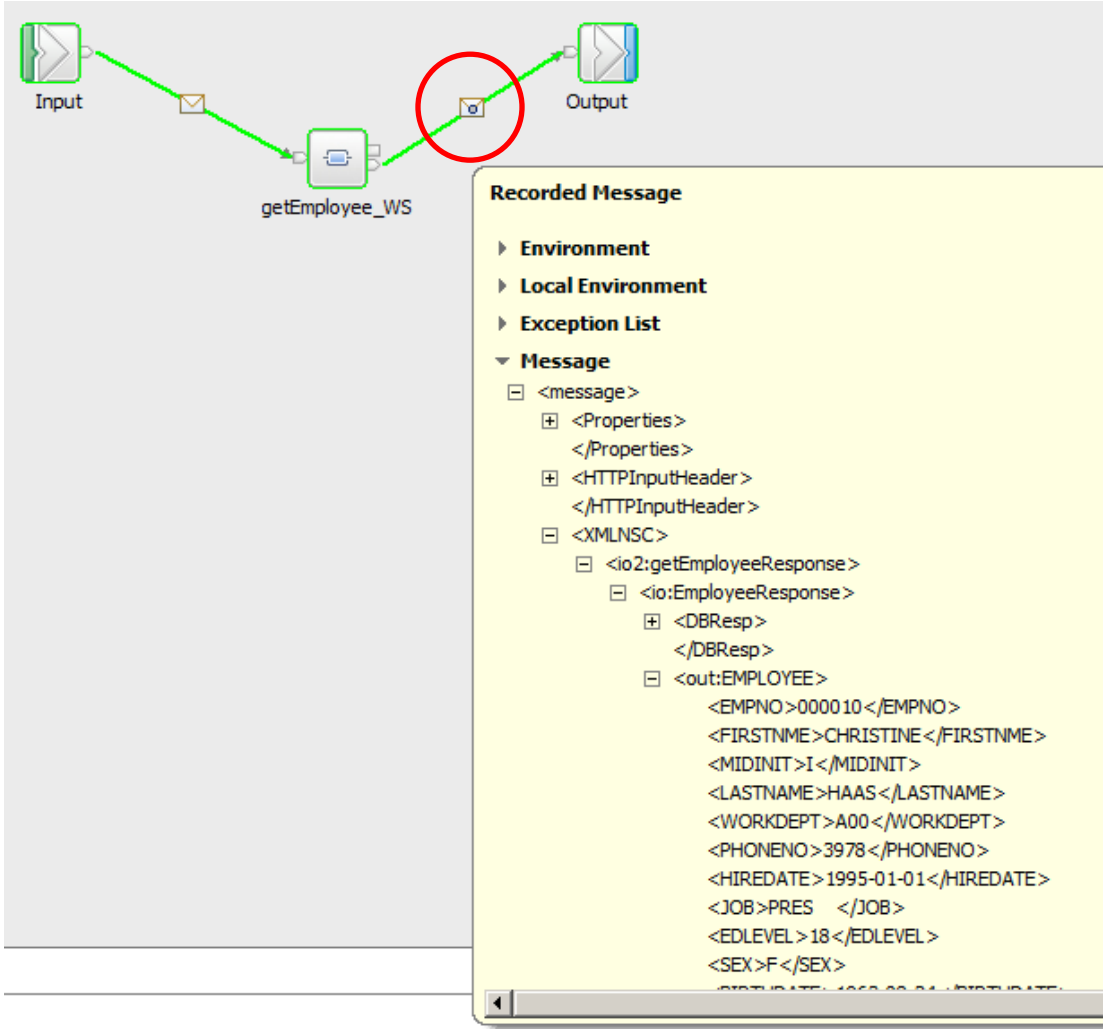
    You can expand various parts of the tree, such as LocalEnvironment, Environment, ExceptionList, as well as the Message.

    Note that the domain of the message tree is shown. In this case, it is XMLNSC.

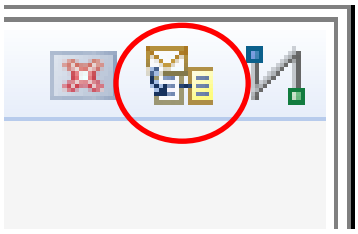


11. Click the icon on the final connector. This time, you will see the message tree after the employee record has been retrieved from the database.

    Note that you can return to the first connector icon to review the message tree again, at that point. The data is not discarded.

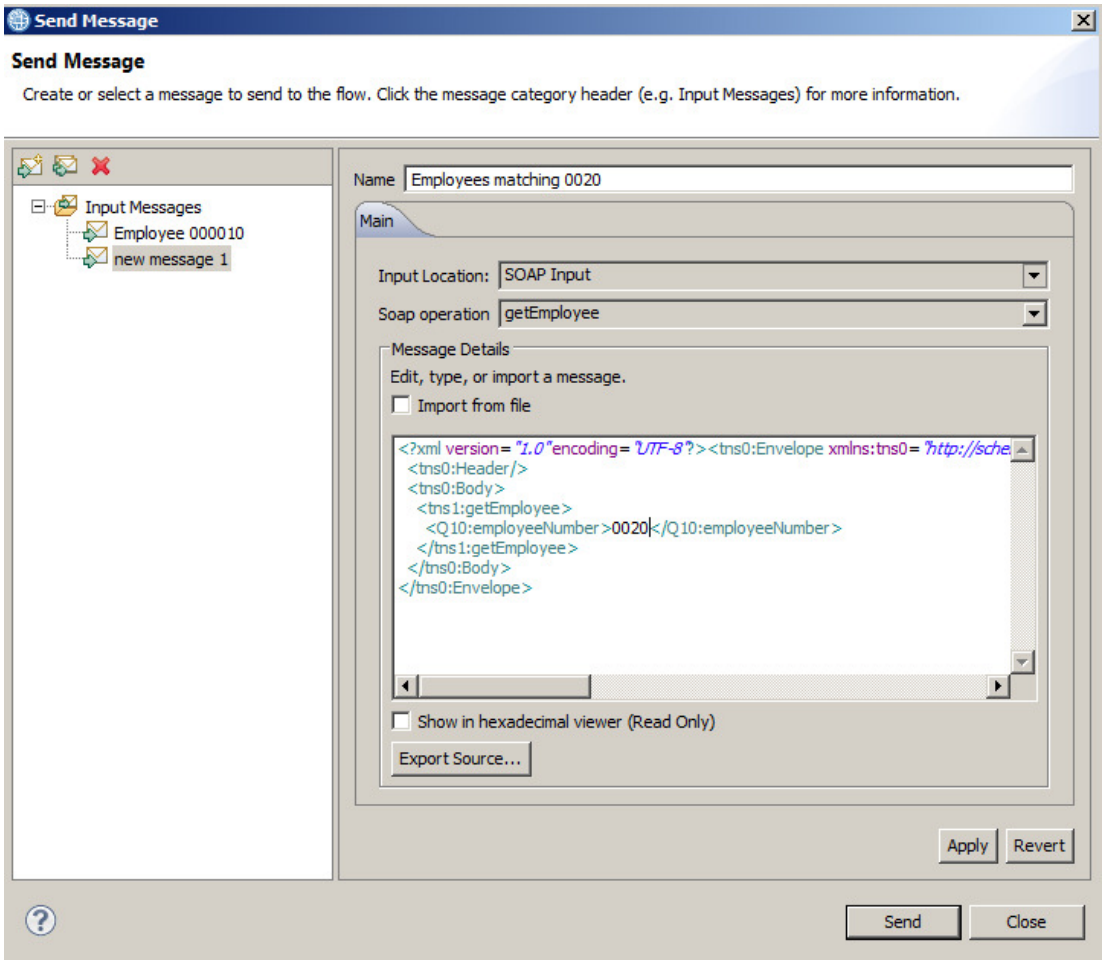


12. Click the Send Message icon again.



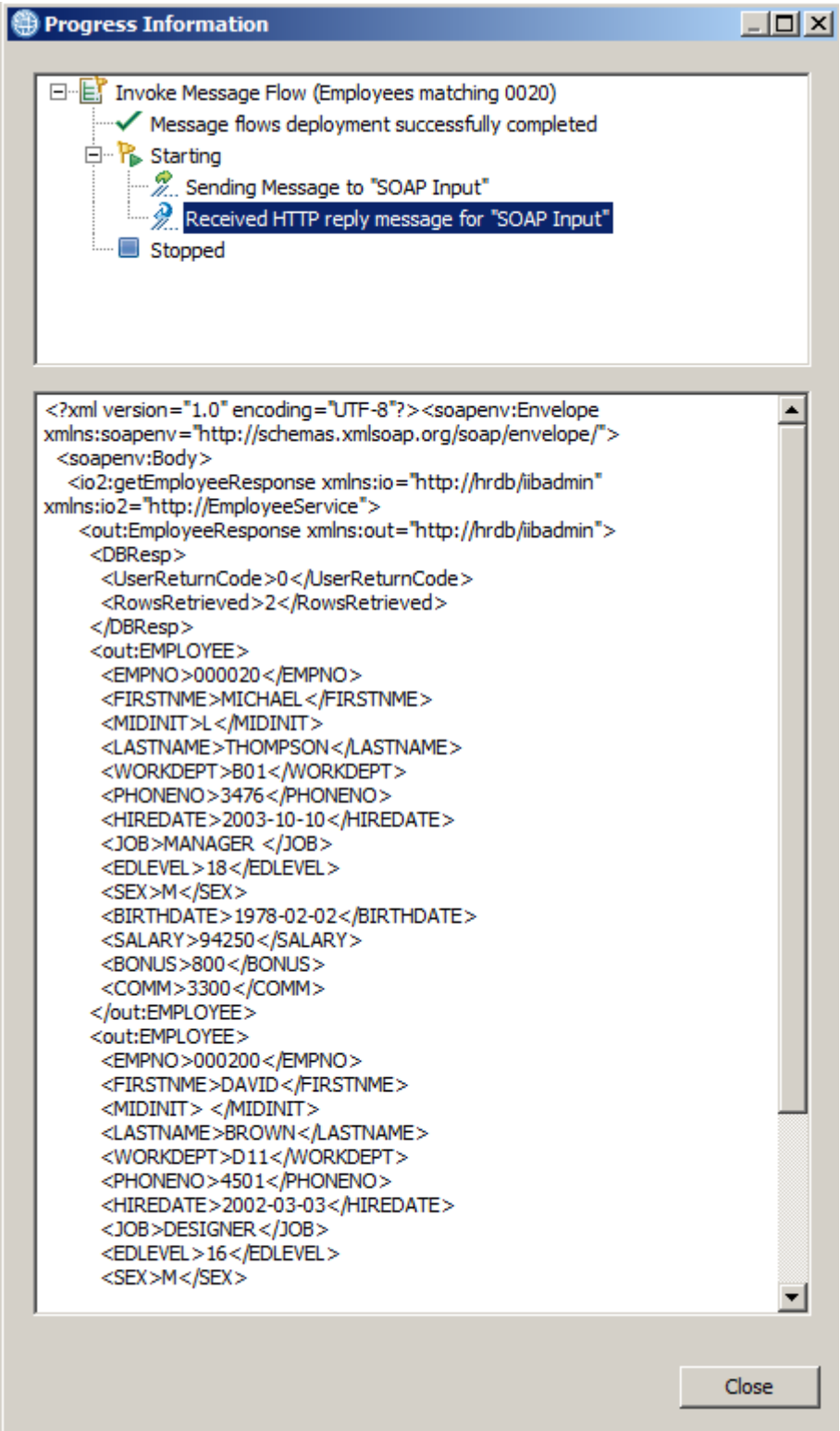
13. Create a new test as before. Name it "Employees matching 0020".

    Set the employeeNumber to 0020, and click Send.



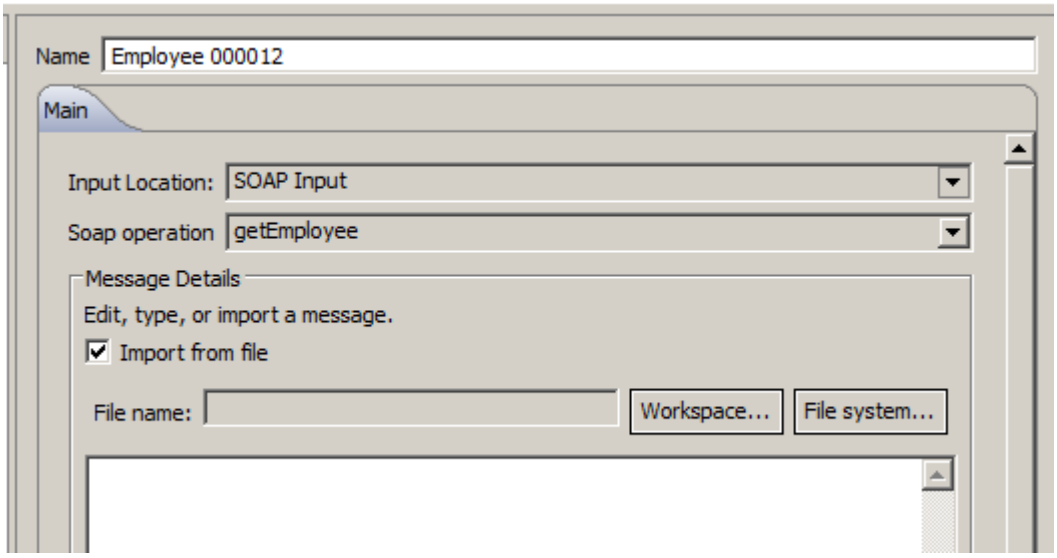
14. The returned message will contain two employees, both containing "0020" in the employee number field.

    RowsRetrieved = 2.

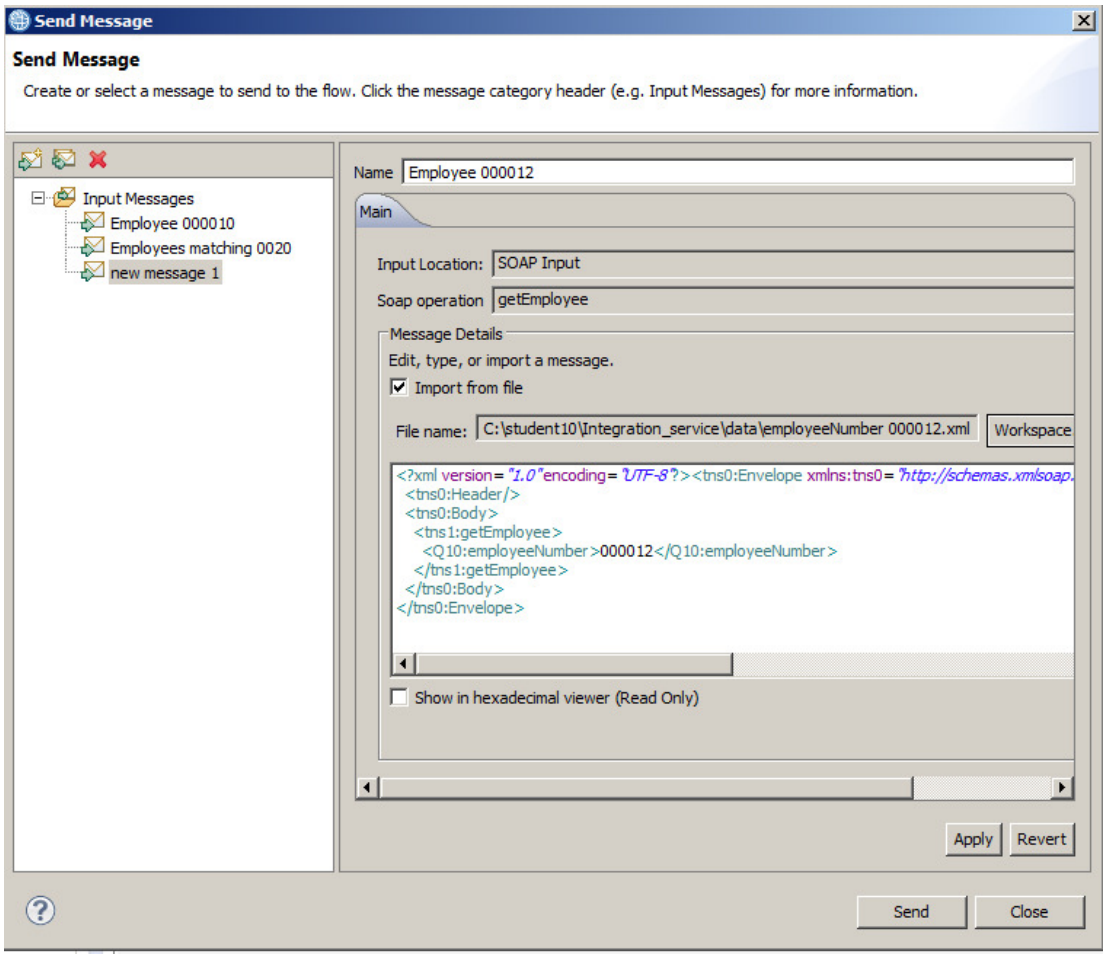


15. Create a new test as before. Name it "Employee 000012".

    This time, select "Import from file". Using the File system button, navigate to c:\student10\integration_service\data\employeeNumber 000012.xml.

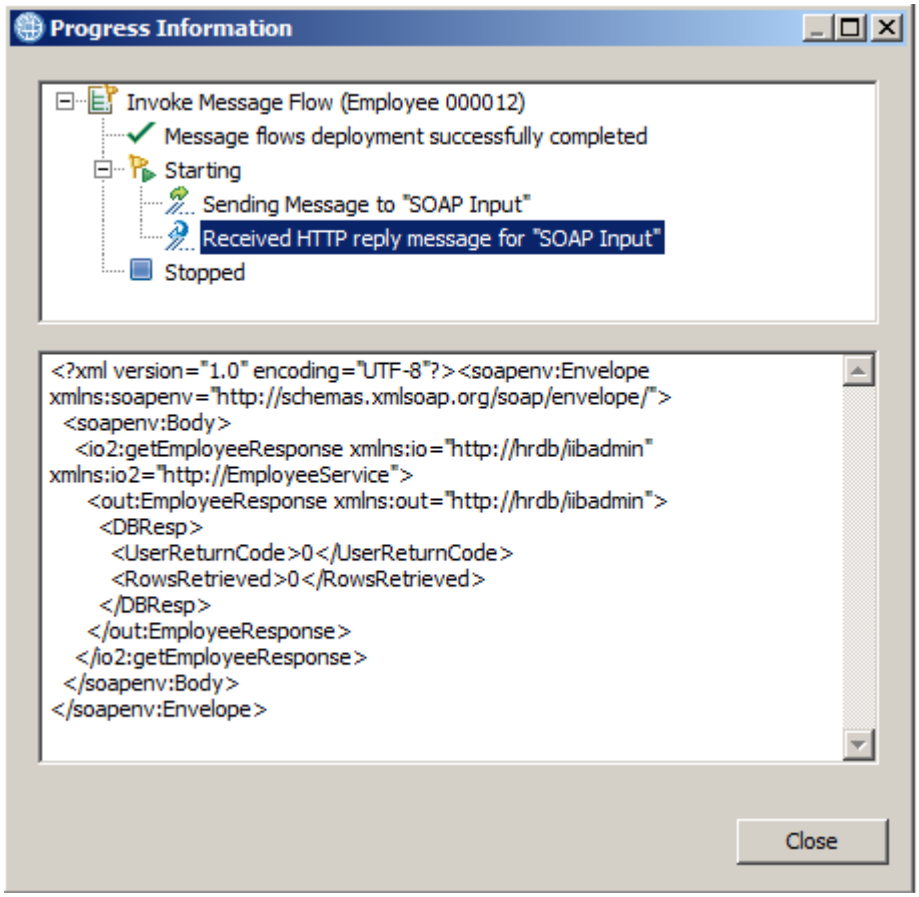


16. Click Send, and use the same tools to review the test outcome.



17. You will see no rows have been returned, and the EMPLOYEE part of the message is not present.

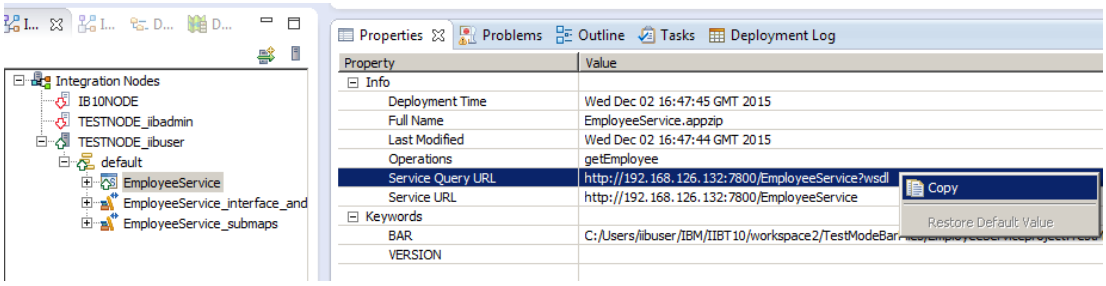
    RowsRetrieved = 0.



5. Test the service with SOAPUI

1. In the IIB Toolkit, select the EmployeeService in the node TESTNODE_iibuser, and select the Properties.

   Highlight the Service Query URL, and right-click and select Copy.



2. Open SOAPUI (on the Windows Start menu). Ensure you are using the SOAPUI workspace **EmployeeService_PrebuiltWorkspace**.

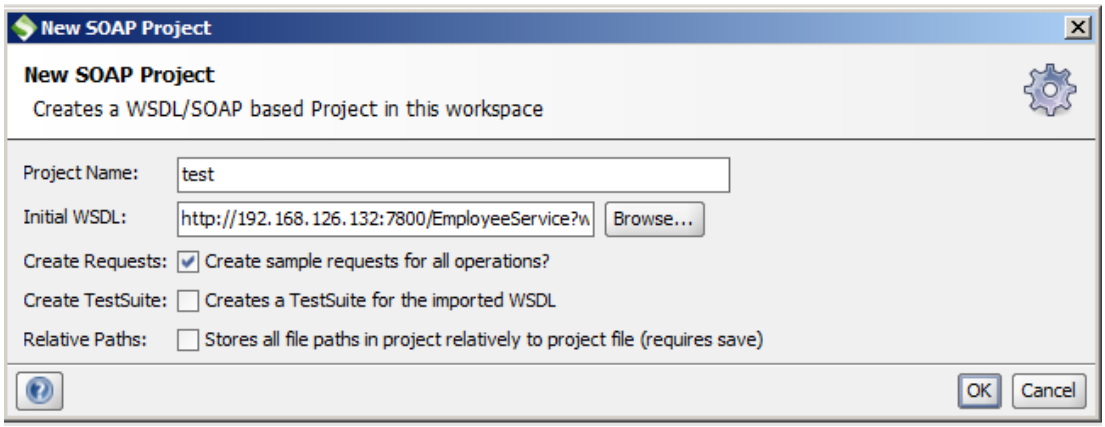
   Create a new SOAPUI project (File, New SOAP Project).

   Provide a suitable name of your choosing for the project (not one of the existing projects).

   In the "Initial WSDL" field, paste the contents of the Windows clipboard.

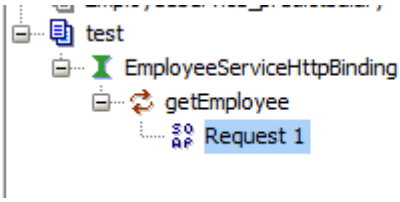
   Important - the Copy function included the text "Service Query URL" at the start of the string, so this must be manually deleted.

   Click OK.

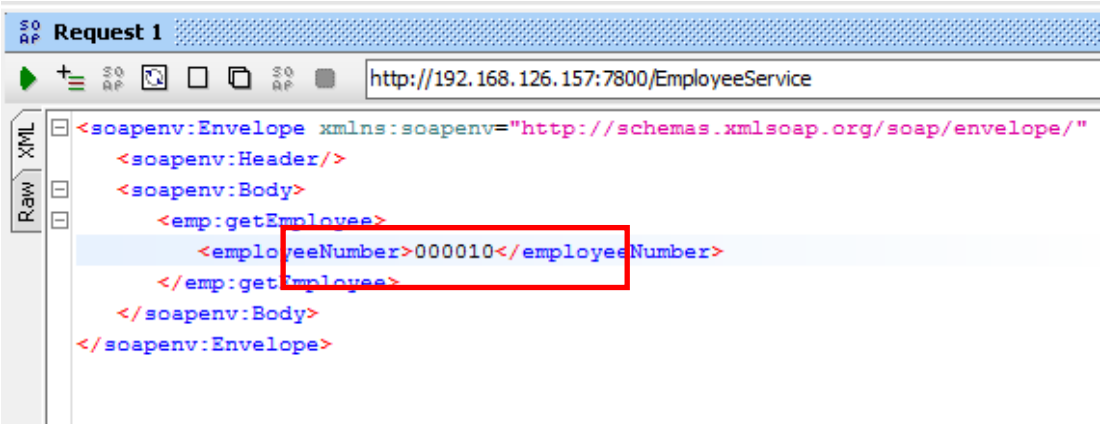


3. Expand the new project, and open Request1. You will see that SOAPUI has pre-populated the web service input with the required schema elements. In this case, these are derived from the EMPLOYEE table.

   Also note that the service port number has been set to 7800. This is the port number that has been allocated for the TESTNODE default server.



4. Provide a value for the EMPNO element. We suggest using value 0020 (replace the ? with this value, as shown).



5. Clicking the green arrow (top left), and the Integration Service should run and return the data to the client, similar to this, depending on the number of rows retrieved.


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope
   <soapenv:Body>
      <io2:getEmployeeResponse xmlns:io2="http://EmployeeService" xmlns:i
         <io:EmployeeResponse>
            <DBResp>
               <UserReturnCode>0</UserReturnCode>
               <RowsRetrieved>1</RowsRetrieved>
               <RowsAdded xsi:nil="true"/>
               <RowsUpdated xsi:nil="true"/>
               <RowsDeleted xsi:nil="true"/>
               <SQLCode_ErrorCode xsi:nil="true"/>
               <SQLState_SQLState xsi:nil="true"/>
               <SQL_Error_Message xsi:nil="true"/>
            </DBResp>
            <out:EMPLOYEE xmlns:out="http://hrdb/iibadmin">
               <EMPNO>000010</EMPNO>
               <FIRSTNME>CHRISTINE</FIRSTNME>
               <MIDINIT>I</MIDINIT>
               <LASTNAME>HAAS</LASTNAME>
               <WORKDEPT>A00</WORKDEPT>
               <PHONENO>3978</PHONENO>
               <HIREDATE>1995-01-01</HIREDATE>
               <JOB>PRES</JOB>
               <EDLEVEL>18</EDLEVEL>
               <SEX>F</SEX>
               <BIRTHDATE>1963-08-24</BIRTHDATE>
               <SALARY>152750</SALARY>
               <BONUS>1000</BONUS>
               <COMM>4220</COMM>
            </out:EMPLOYEE>
         </io:EmployeeResponse>
      </io2:getEmployeeResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

6. Create a second SOAPUI request in the same project. This time, use the value 000012, which does not exist in the EMPLOYEE table.

   When you click the green arrow, you should see the response indicating that the flow successfully accessed the tables (user return code 0), and no rows were returned (RowsRetrieved = 0).


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <io2:getEmployeeResponse xmlns:io2="http://EmployeeService" xmlns:io="http
         <io:EmployeeResponse>
            <DBResp>
               <UserReturnCode>0</UserReturnCode>
               <RowsRetrieved>0</RowsRetrieved>
               <RowsAdded xsi:nil="true"/>
               <RowsUpdated xsi:nil="true"/>
               <RowsDeleted xsi:nil="true"/>
               <SQLCode_ErrorCode xsi:nil="true"/>
               <SQLState_SQLState xsi:nil="true"/>
               <SQL_Error_Message xsi:nil="true"/>
            </DBResp>
         </io:EmployeeResponse>
      </io2:getEmployeeResponse>
   </soapenv:Body>
</soapenv:Envelope>
```


END OF LAB GUIDE