16L02

IBM Integration Bus

Developing a REST API using a Swagger JSON document

(without reference to XML schemas)

Featuring:

The REST API editor for IIB
The Mapping Node tools for REST API
Testing with Swagger UI

September 2016

Hands-on lab built at product version 10.0.0.6

1. Introduction and Preparation

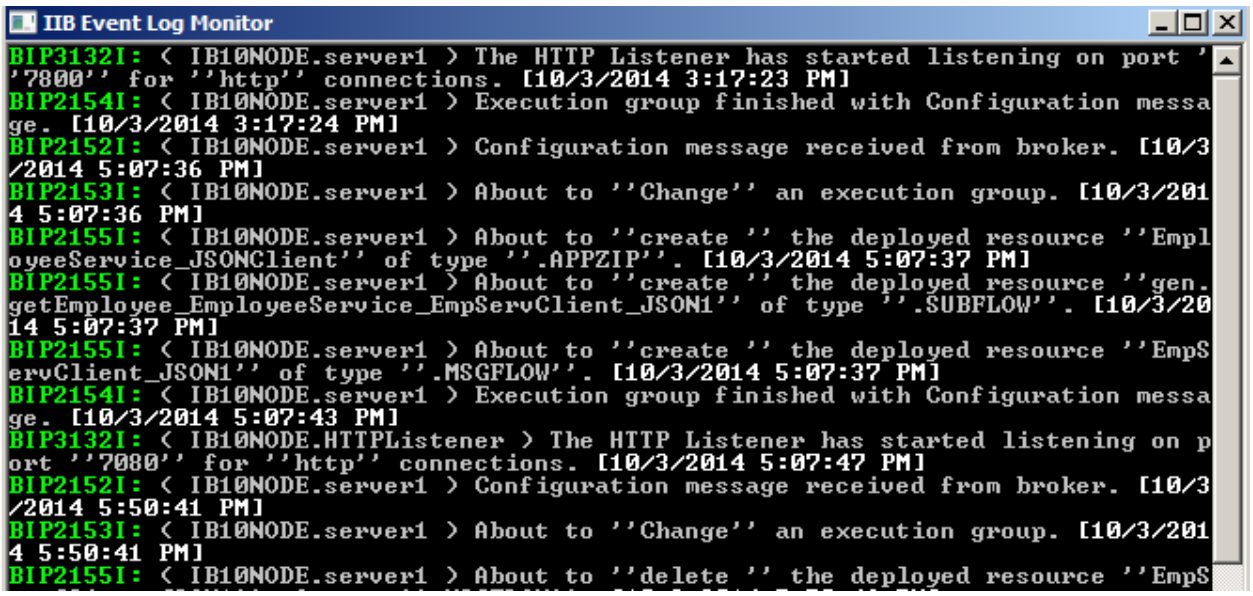
1.1 Introduction

In this lab you will create a new REST API, based on a provided Swagger document. The document is based on the EMPLOYEE scenario, and describes a number of REST operations. This guide will implement the Get Employee operation.

1.2 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.



This tool is not shipped as part of the IIB product; please contact us directly if you would like a copy.

Updates for IIB v10.0.0.3

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 and the REST 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.

Updates for IIB v10.0.0.5

IIB v10, fixpack 5 (10.0.0.5) has made significant changes in the Toolkit representation of REST APIs, the editor functions, and with the tools provided with the Mapping Node editor. This lab has been updated to reflect those changes.

Updates for IIB v10.0.0.6

This lab now uses JSON models throughout, and has removed the parts of the lab that were based on earlier XML schema components. The Mapping nodes use the Map Node wizard, which automatically creates input and output assemblies, based on the JSON model definitions.
Use of the pre-built submap (used in earlier versions) has been removed, and replaced with a single mapping node.

1.3 Configure TESTNODE_iibuser for REST APIs

The instructions in this lab guide are based on a Windows implementation, with a user named “iibuser”.
The Windows VMWare image on which this lab is based is not available outside IBM, so you will need to provide your own software product installations where necessary.

Login to Windows as the user "iibuser", password = "passw0rd". (You may already be logged in).

Start the IIB Toolkit from the Start menu.

The IIB support for the REST API requires some special configuration for the IIB node and server.

1. Ensure that TESTNODE_iibuser is started.

2. Enable Cross-Origin Resource Scripting for REST. This is required when testing with the SwaggerUI test tool. See http://www.w3.org/TR/cors/?cm_mc_uid=09173639950214518562833&cm_mc_sid_50200000=1452177651 for further information.

   (Helpful hint - the VM keyboard is set to UK English. If you cannot find the "\" with your keyboard, use "cd .." to move the a higher-level folder in a DOS window), or change the keyboard settings to reflect your locale.)

   In an IIB Integration Console (shortcut on the Start menu), run the following command.

   Note, the text should be typed on a single line - the parameters are shown on different lines here for readability; the same approach is taken throughout this and other lab documents.

```
mqsichangeproperties TESTNODE_iibuser
      -e default
      -o HTTPConnector
      -n corsEnabled -v true
```

1.4 Configure Integration Bus node to work with DB2

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

1. In an IIB Integration Console (from the Start menu), and navigate to

   c:\student10\Create_HR_database

2. Run the command

   3_Create_JDBC_for_HRDB

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

3. Run the command

   4_Create_HRDB_SecurityID

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

   mqsistop TESTNODE_iibuser

   mqsistart TESTNODE_iibuser

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

2. Create the HR_Service REST API

In this section you will create a new REST API. This scenario will be based on the EmployeeService example that you may have used in other labs in this series.

2.1 Examine the EmployeeService JSON document

1. In Windows Explorer, locate the file

   c:\student10\REST_API_HR_Service\resources\HR_Employee_and_Department_Services.json

   Open the file with the Notepad++ editor (right-click, select Edit with Notepad++).

   We have installed a JSON document plugin into Notepad++, so this JSON document will be formatted for easy reading.

   The JSON document has been constructed to define interfaces for the EMPLOYEE and DEPARTMENT tables. The document also has definitions for DBResp (for database response information, EmployeeResponse (includes DBResp and EMPLOYEE), and DepartmentResponse (includes DBResp and DEPARTMENT).

   The main section of the document is a series of operations (GET, POST, PUT, etc), associated with different types of operation (getEmployee, listEmployees, etc).

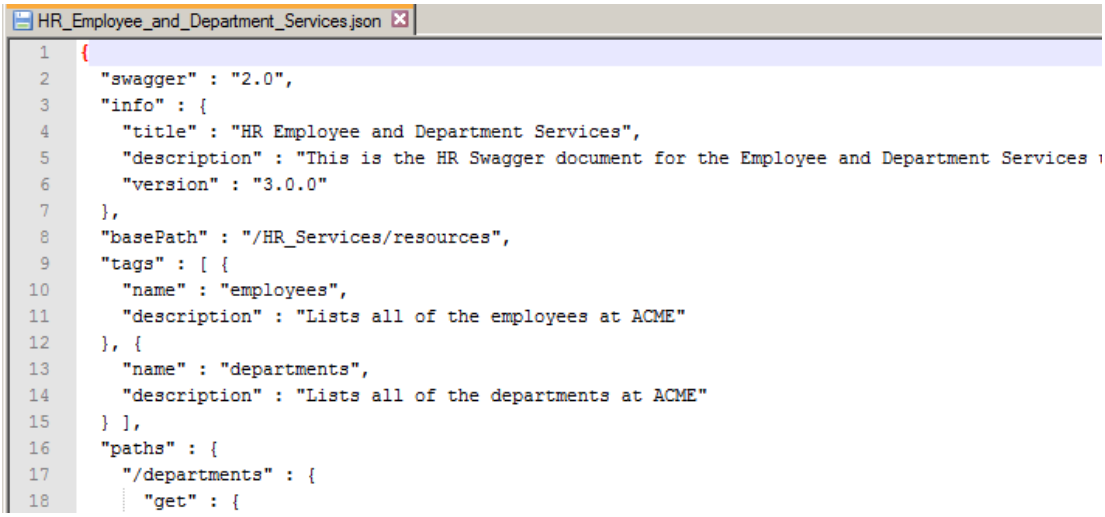
   Near the bottom of the document are two definitions for EMPLOYEE and DEPARTMENT, which defines the precise structure of these elements.

   This document will be used as the basis of the REST API that you will build in IIB.

   Look for the variable "EMPNO", which is referenced in several places in the document. This variable will be used by the IIB REST API.

   The input parameter to the operation that you will implement is “employeeNumber”; this is defined as a path parameter for the required operation. Note the “required” parameter is set to “true”.

   Close the editor without making any changes to the document.



2.2 Import the HRDB Shared Library

The REST API that you will develop will use the EMPLOYEE table from the HRDB database. This requires the HRDB Database Definition project, which represents the tables schemas. This is used by the Mapping nodes that access the EMPLOYEE table. The project is contained in a Shared Library called HRDB; creating it as a shared library means that it can be shared with more than one project.

The HRDB shared library and associated project will not be developed in this lab - see Lab 01 from the 2015 series on lab guides (Creating an Integration Service) for details of how to do this. In this lab, you will import a pre-built version of this project.

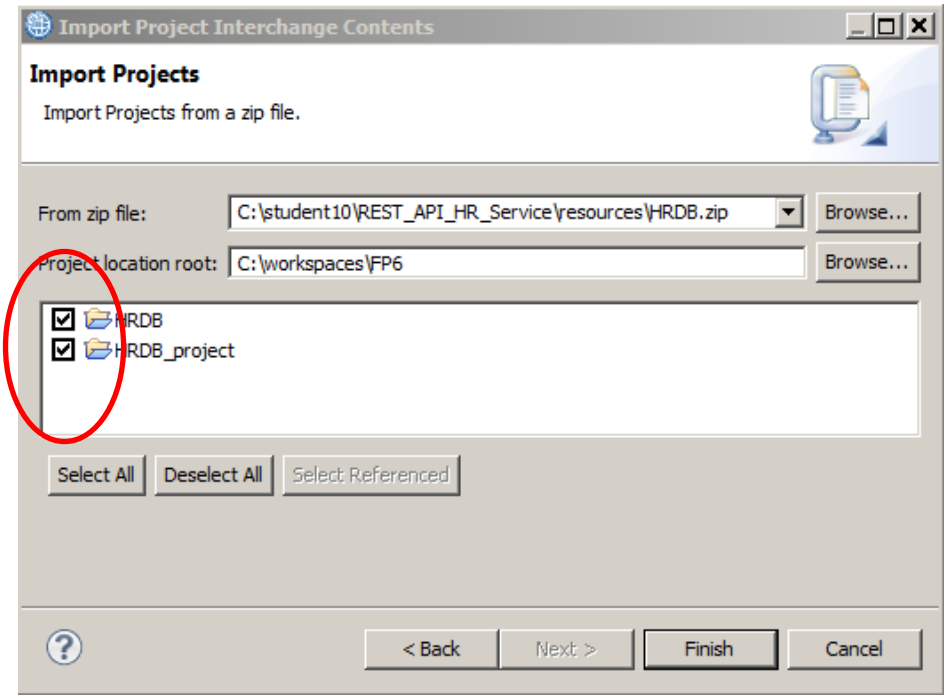
1. Start the IIB Integration Toolkit, and create a new workspace. If you already have a workspace open, click File, Switch Workspace. Give the new workspace the name

   `c:\users\iibuser\IBM\IIB 10\workspace_REST`

2. In the new workspace, import the Project Interchange file:

   `C:\student10\REST_API_HR_Service\resources\HRDB.zip`

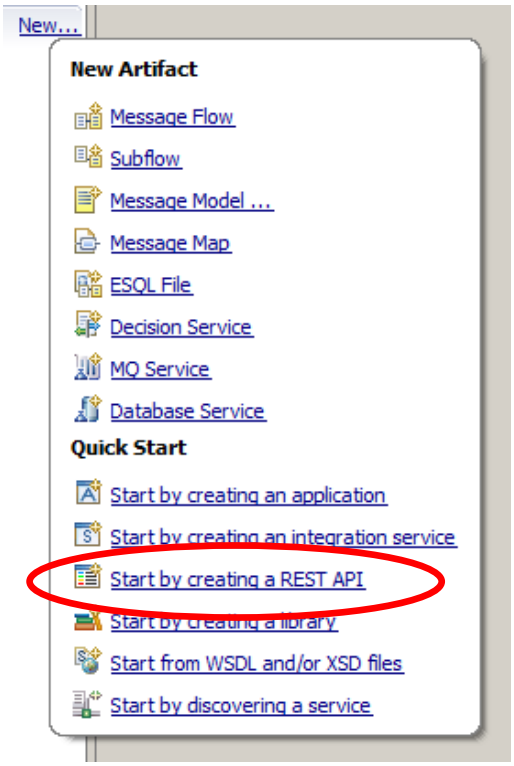
   Import both the **HRDB** shared library and **HRDB_project** from this PI file, and click Finish.



3. When imported, you will see the HRDB shared library as shown here. Note that HRDB_project is an Independent Resource (ie. it is a plain project, not a shared library).

2.3 Create the new REST API

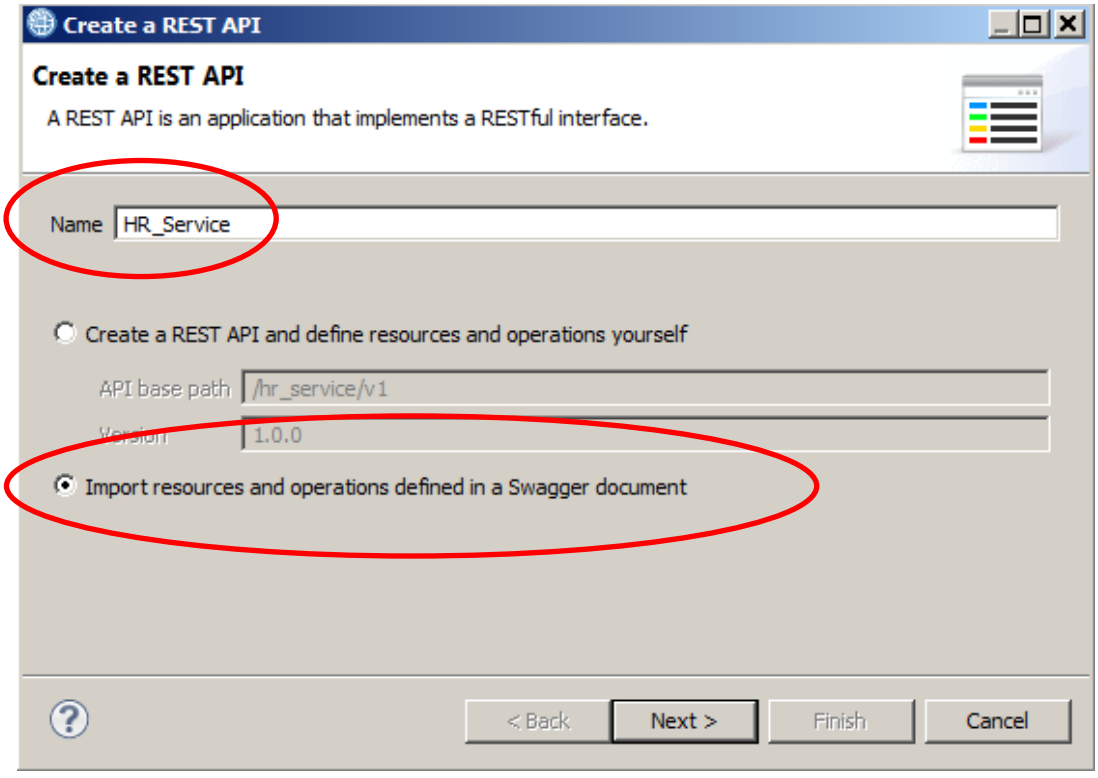
1. In the workspace, create a new REST API..



2. Name the new service HR_Service.

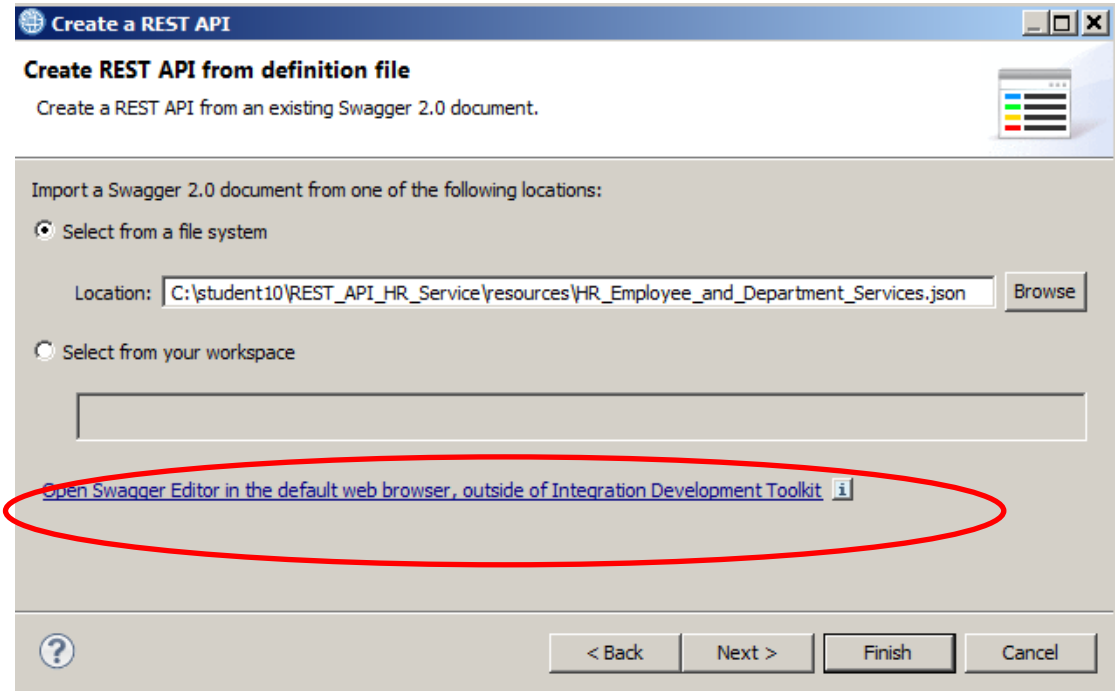
   Select "Import resources and operations defined in a Swagger document".

   Click Next.



3. Using the Browse button, import the JSON document

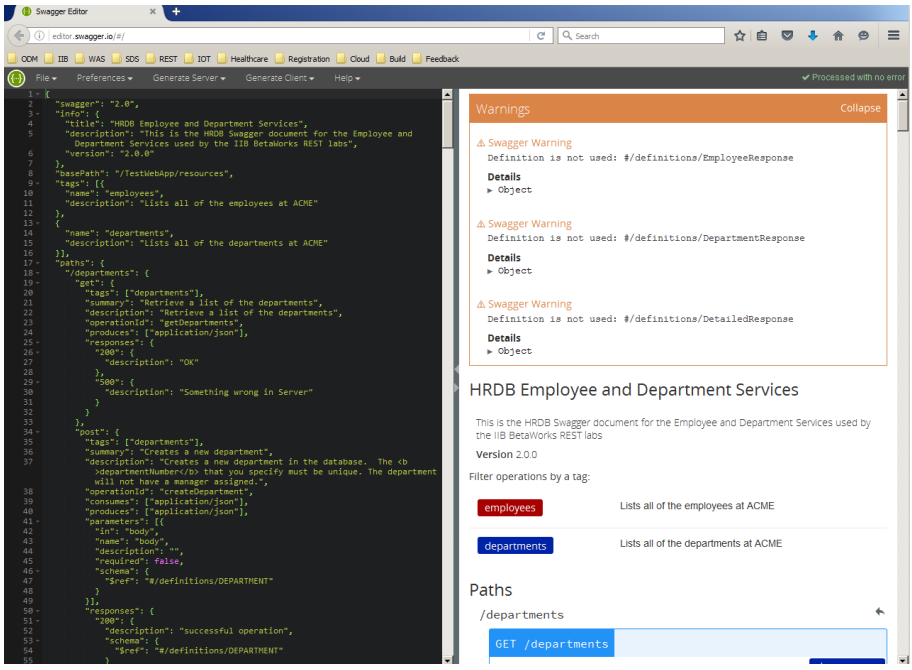
c:\student10\REST_API_HR_Service\resources\HR_Employee_and_Department_Services.json



4. If you want to edit the HR_Employee_andDepartment_Services.json file in the Swagger Editor (editor.swagger.io), you can click on the "Open Swagger Editor ..." link on the Toolkit wizard. This will automatically start the Swagger Editor in a browser window (shown below). You can copy/paste the ".json" file into the editor in the left pane of the Swagger Editor.

   You will see warnings about unused definitions. You can ignore these (they will be implemented later).

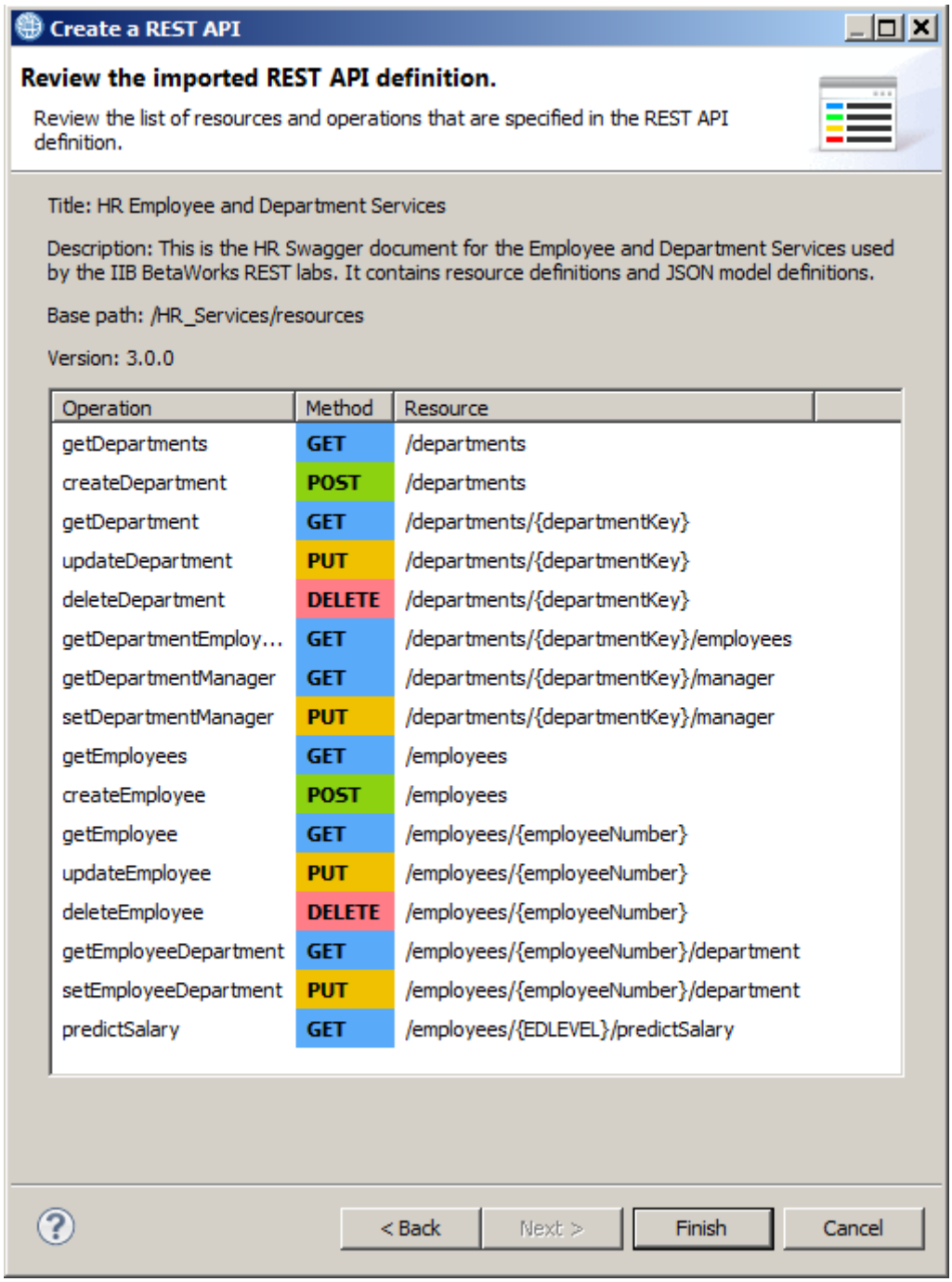
   However, in this lab, you will use the Integration Toolkit editor, so switch back to the Toolkit and click Next.



5. The summary window will show you all of the REST operations that were defined in the JSON document. These operations were constructed to match the EMPLOYEE and DEPARTMENT tables in the HRDB database.

   Note there is an operation named getEmployees (ie. retrieve a list of all employees), and an operation named getEmployee. This lab will implement the **getEmployee** operation.

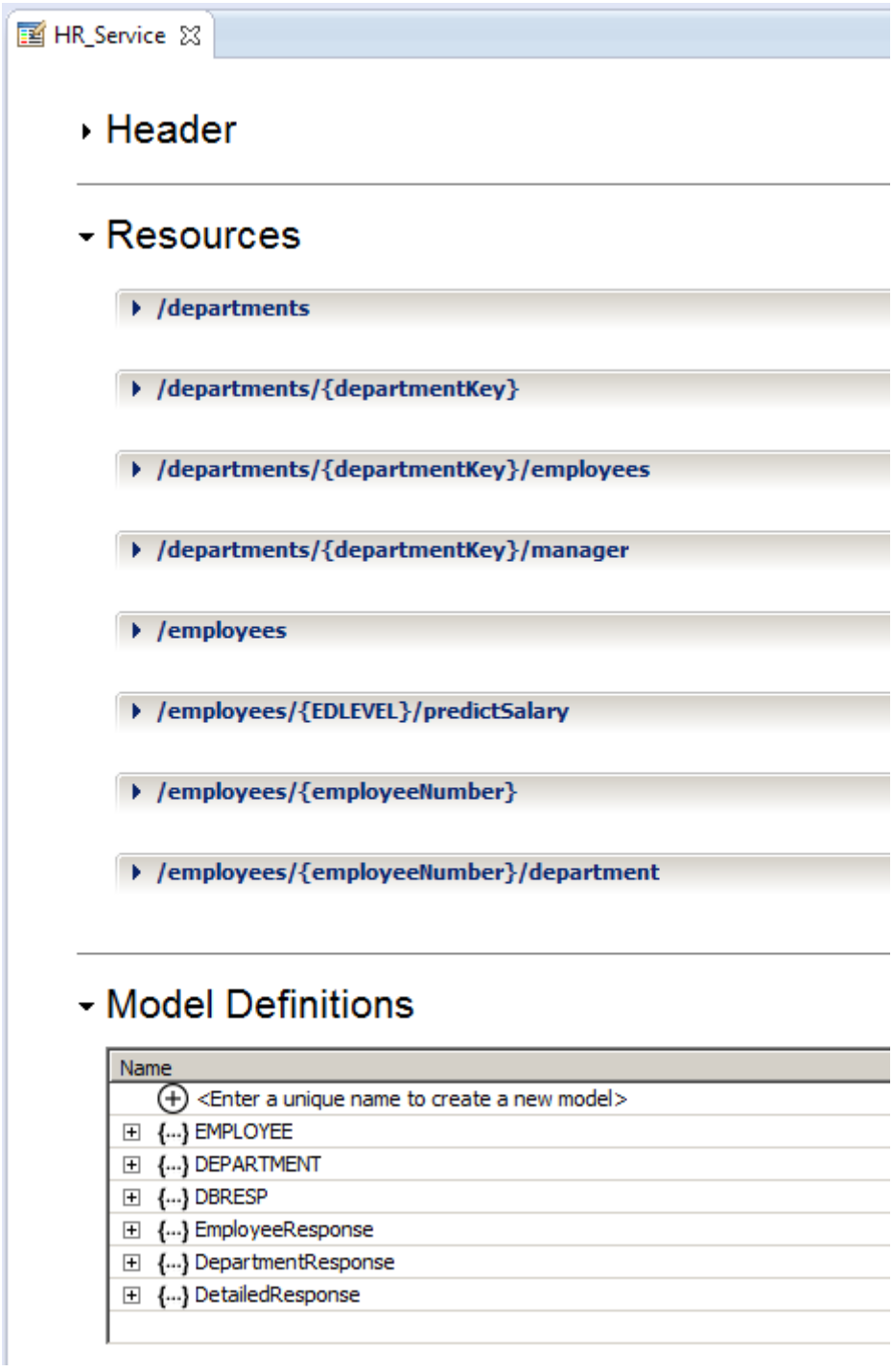
   Click Finish.



6. The swagger document has now been imported into the Integration Toolkit. The import process has also created a base REST API and a message flow that implements the REST API.

   The imported and generated items are split into five main sections in the REST API editor:

   - Header - containing the base URL for the REST API, title and description
   - Resources - containing all the resources from the swagger document, and all of the operations that are contained within each resource
   - Model Definitions - schema definitions for the input and output JSON objects
   - Error Handling - options to add some elements of runtime security
   - Security - basic security parameters



7. As an example of Resources, expand **/employees/{employeeNumber}**. (You may wish to collapse the **/departments** resource, for readability).

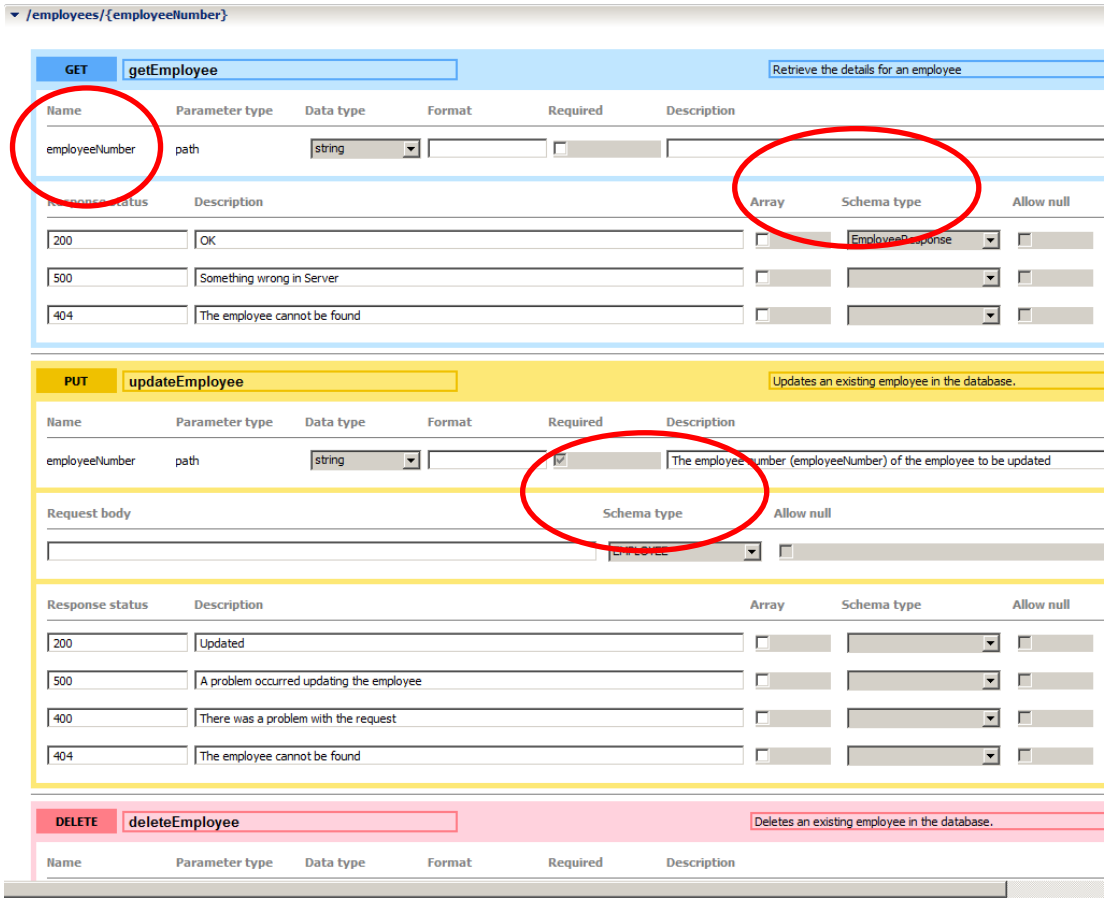
   You will see three operations, GET, PUT and DELETE.

   For some of the operations (for example, the updateEmployee PUT operation in this resource), the Schema type has been set (in this case to EMPLOYEE).

   For some other operations (for example the getEmployee GET operation), the input parameter is specified (employeeNumber). Because {employeeNumber} has been specified as part of the REST API URL path, the input parameter name (employeeNumber) is not editable.

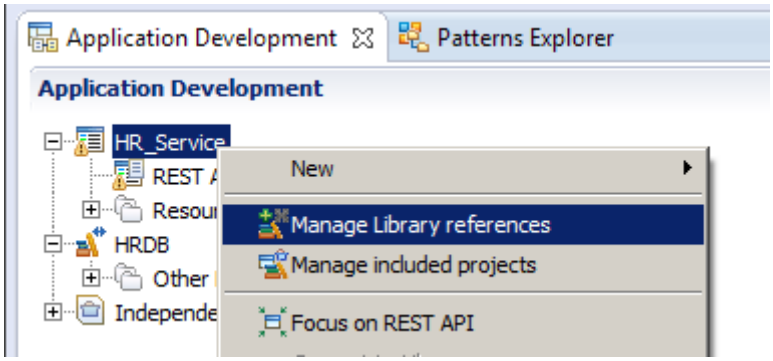
   The Schema Type of the successful (200) operation has been set to EmployeeResponse (originally specified in the swagger doc).

   You can use the Schema Type dropdowns to change the required schema for the operation. The available values are derived from the Model Definitions section. If no schema type is specified, the REST operation can dynamically specify the format of the output message.

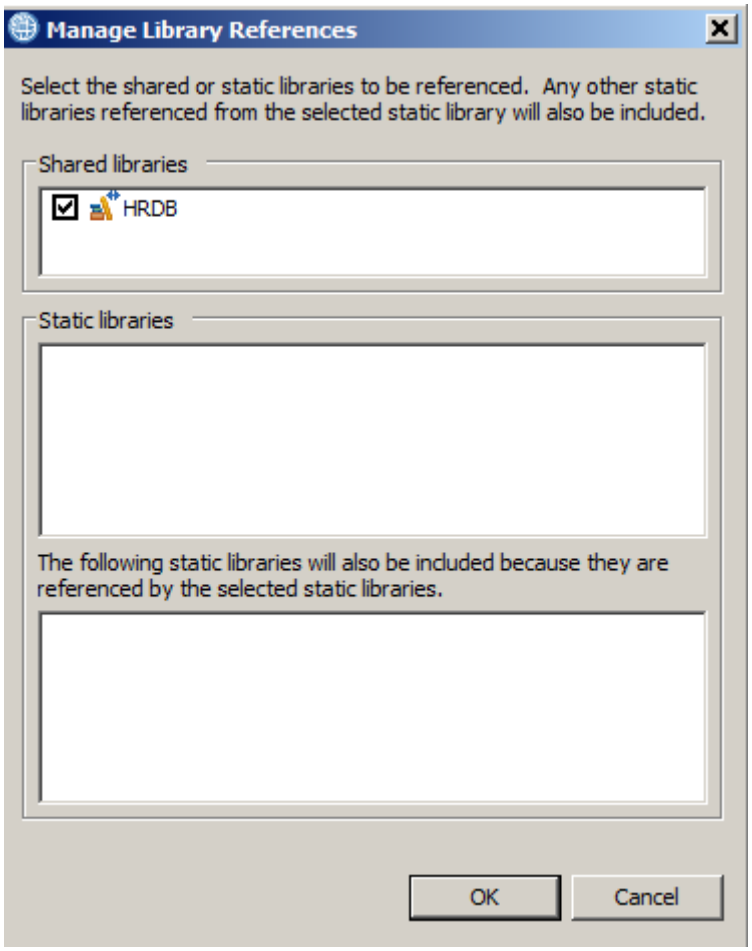


8. Before proceeding with the implementation, the REST API project has to reference the HRDB shared library. This is because you will use a map to retrieve data from the database; the map references elements in the HRDB.dbm file.

   In the navigator, right-click HR_Service and select "Manage Library references".



   Tick HRDB and click OK.



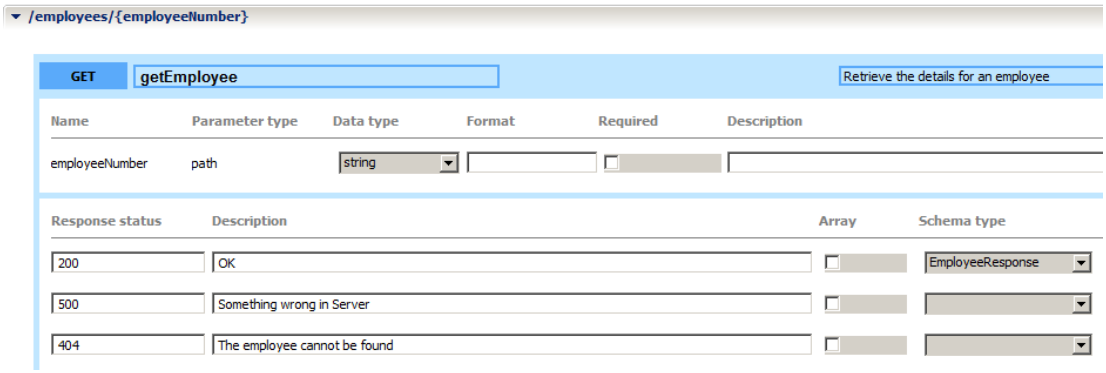
2.4 Implement the getEmployee operation

You will now implement the getEmployee operation that was defined by the EmployeeService json document.

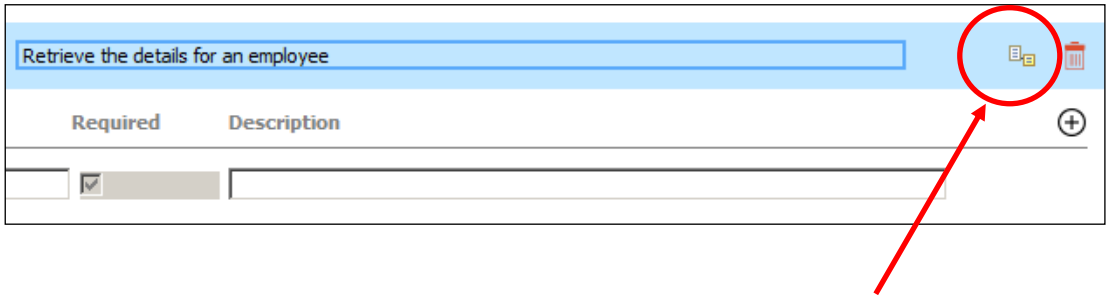
1. In the Integration Toolkit, switch to the HR_Service REST API.

   Expand the REST API Resources, and position the editor at the **/employees/{employeeNumber}** resource.

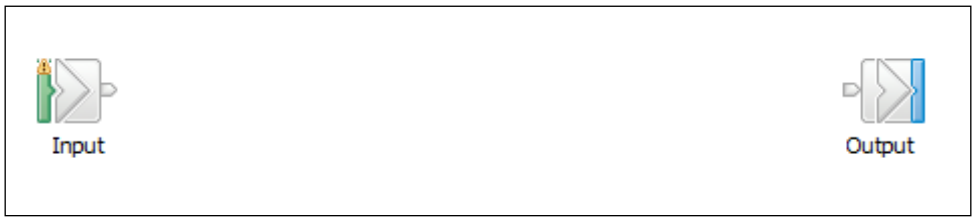
   The getEmployee operation is the first operation in this resource.



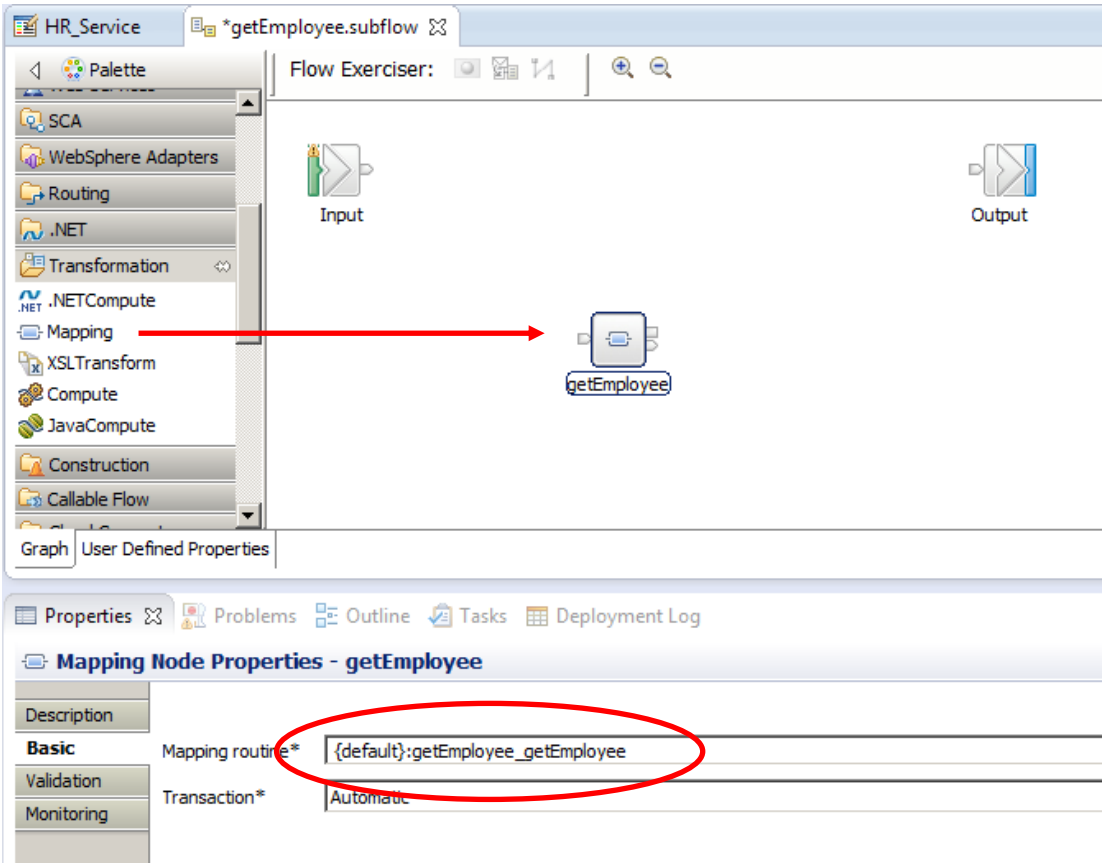
2. In the top right part of the description, click on the icon to "Create a subflow for the operation".



   This will create a skeleton subflow where you will provide the logic to implement the operation:



3. Drop a Mapping node onto the flow editor as shown. As soon as you drop the node onto the flow editor, change the name of the node to getEmployee, and click Return (this should automatically name the map file, as shown in the node properties, below. The map name is formed by concatenating the REST operation name (getEmployee in this example) with the label of the mapping node (also getEmployee in this example).



2.5 Implement the map

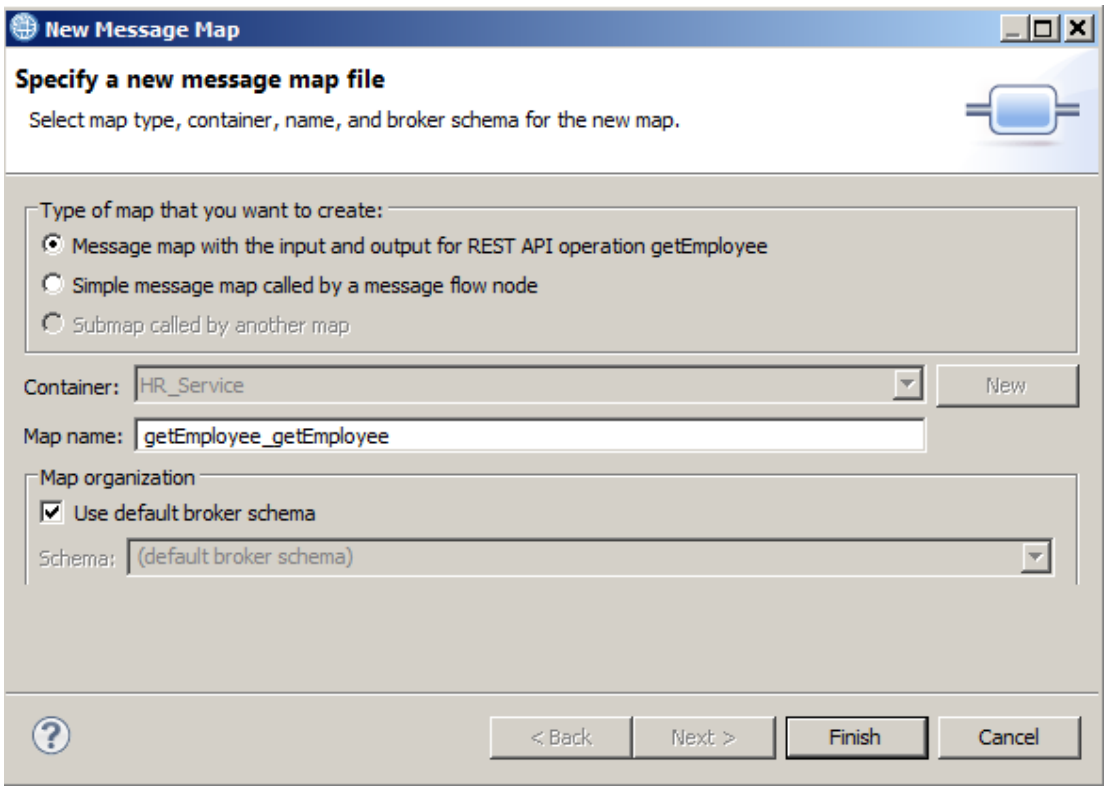
1. Double-click the Mapping node that you have placed on the flow editor.

   Note that the new Map wizard has provided two options:

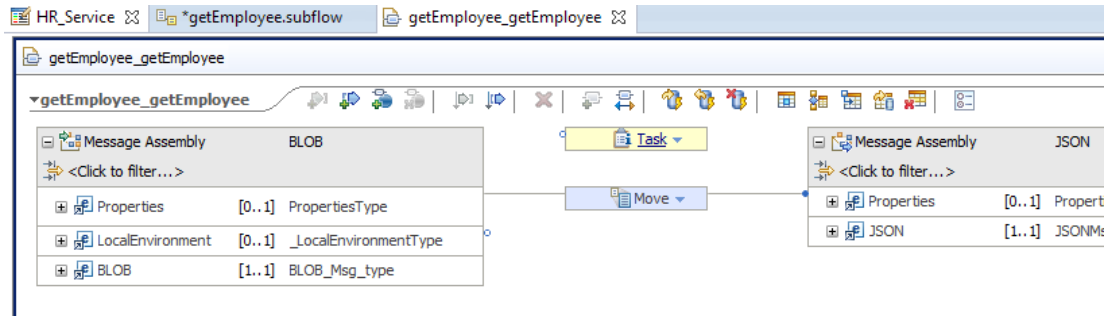
   - Message map for input/output REST operation getEmployee.
     This option was introduced in IIB v10.0.0.5, and is specifically designed for implementing REST APIs with the Mapping node. Ensure this option is selected.

   - Simple message map called by message flow node.
     This option enables the more general Mapping Node options. It could be used for REST APIs, but the map designer would then be responsible for manually specifying the inputs and outputs correctly.

   Click Finish (Next is not available as an option, since the map wizard automatically creates the inputs and outputs).



2. The basic map is shown. Because the getEmployee operation is a GET method, the input is defined as a BLOB and the output message is a JSON message. The input message can be defined as a BLOB because for a GET operation, the input parameters are provided in the message header, and it is not necessary to parse the message body.

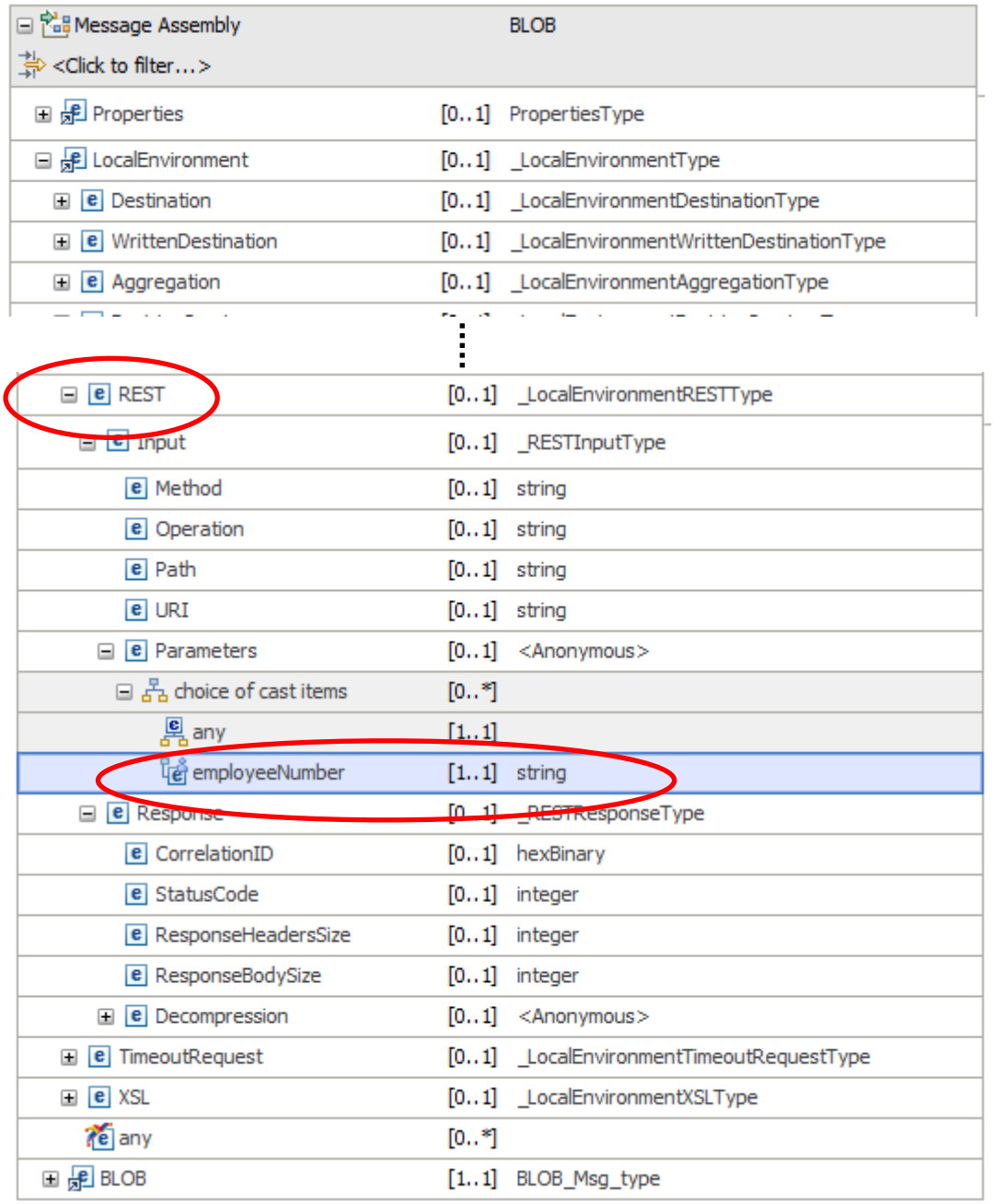


3. Note that the input message assembly is already shown with the LocalEnvironment folder. This is because IIB places some parts of the input of a REST API into the LocalEnvironment, and the Mapping Node will almost always need access to this for developing application logic.

   Expand the LocalEnvironment folder, and fully expand the REST subfolder.

   Note that employeeNumber has already been added to the REST folder - this was derived from the input Schema Type on the swagger definition of the getEmployee operation.

   Also note that IIB 10.0.0.6 has added the Response folder under the REST folder.

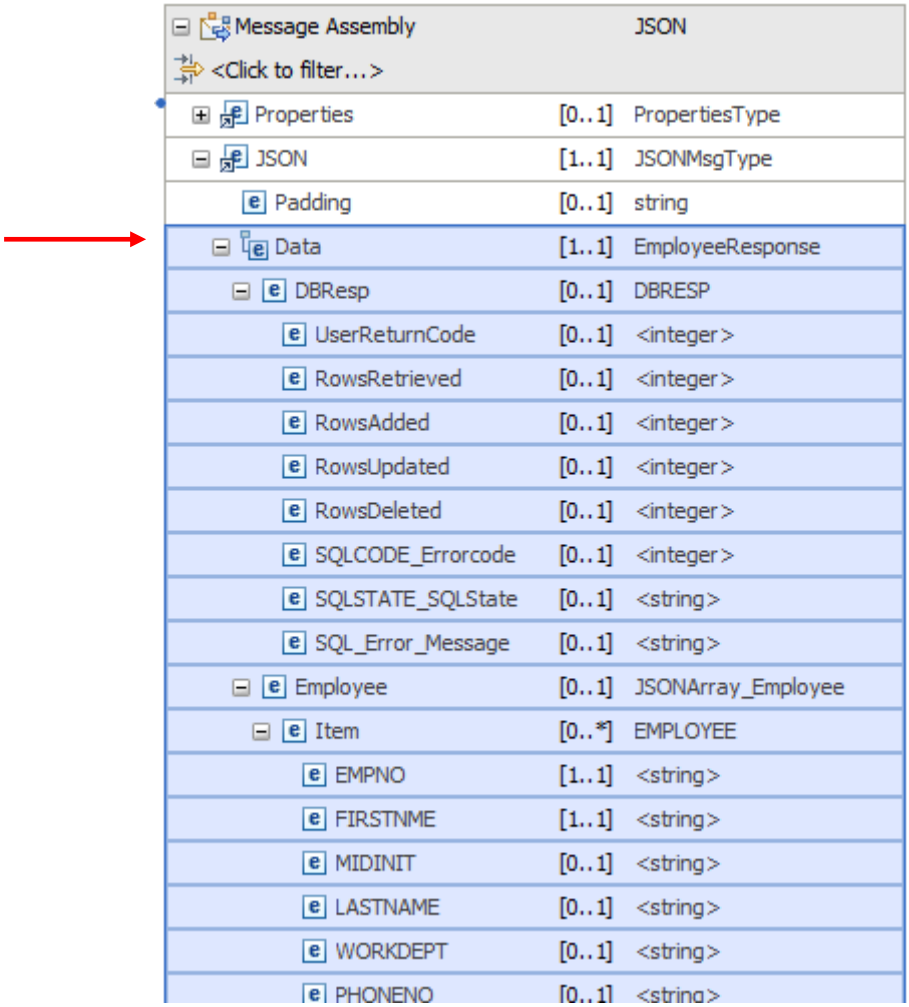


4. Now focus on the output message assembly. The Map wizard has provided a JSON output assembly (because the getEmployee operation specified a schema type of EmployeeResponse for the response message).

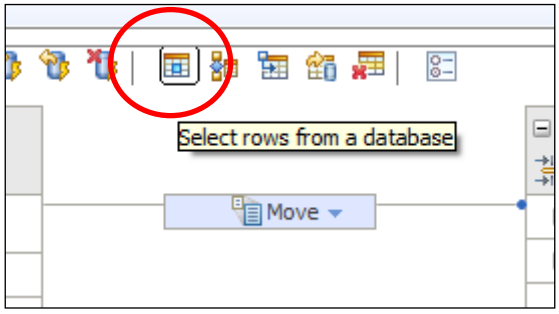
   Fully expand the output assembly. The JSON Data is of type EmployeeResponse.
   You will see the DBResp (JSON object) and Employee (JSON array) items underneath the Data element.

   Note the Item element (of type EMPLOYEE) is a repeating element.

   Ensure the Data (type EmployeeResponse) element is highlighted. This is important for the subsequent step which will automatically create a Select transform, and connect it to the highlighted element.

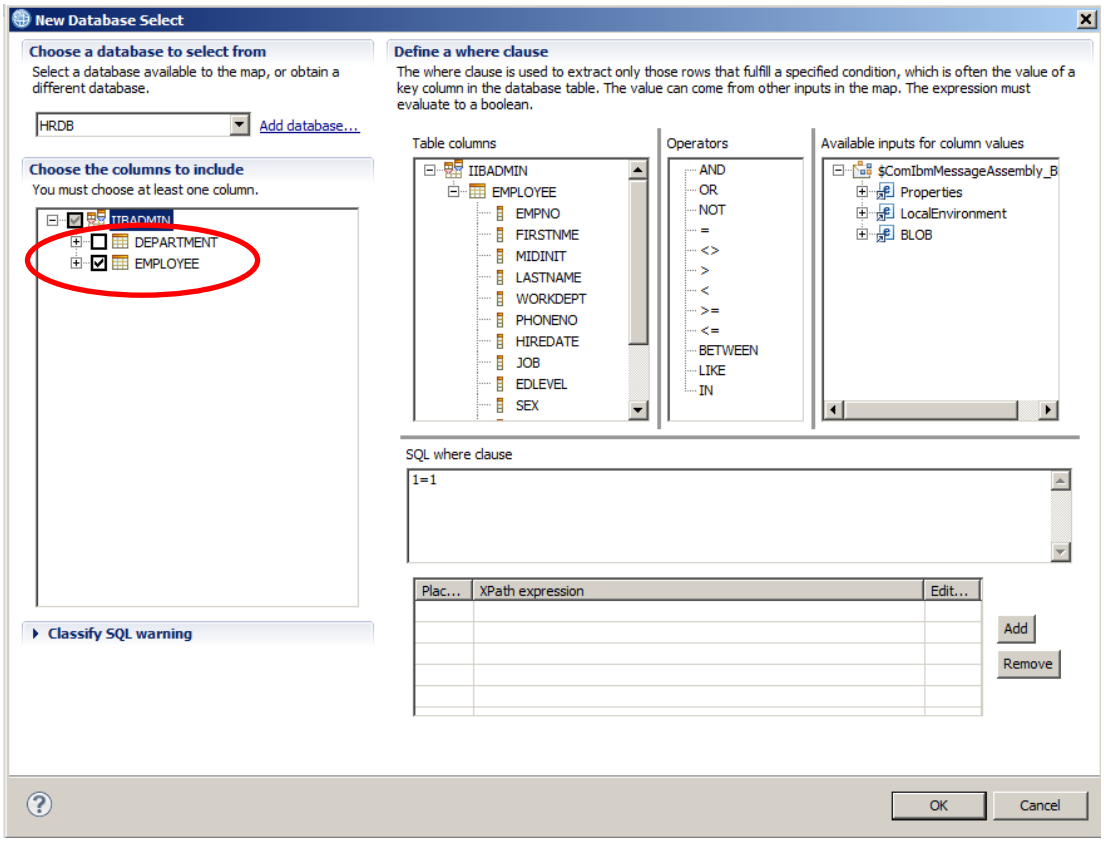


5. Now add a transform to retrieve data from the HRDB database. Click the "Select rows from a database" icon at the top of the map editor.

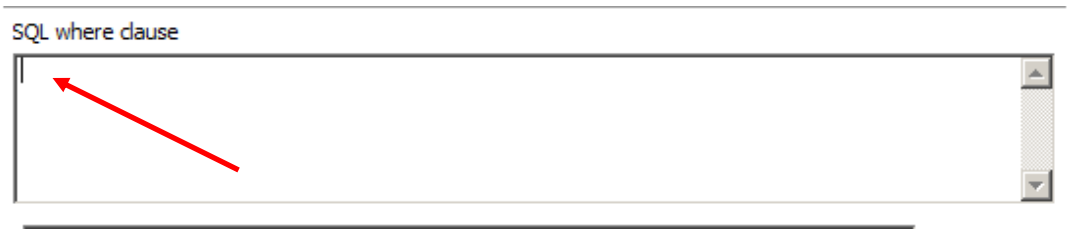


6. In the New Database Select window, the HRDB database should already be available and selected (because you set the HRDB Library reference).

   Select the EMPLOYEE table.



7. Remove the "1=1" statement from the "SQL where clause".



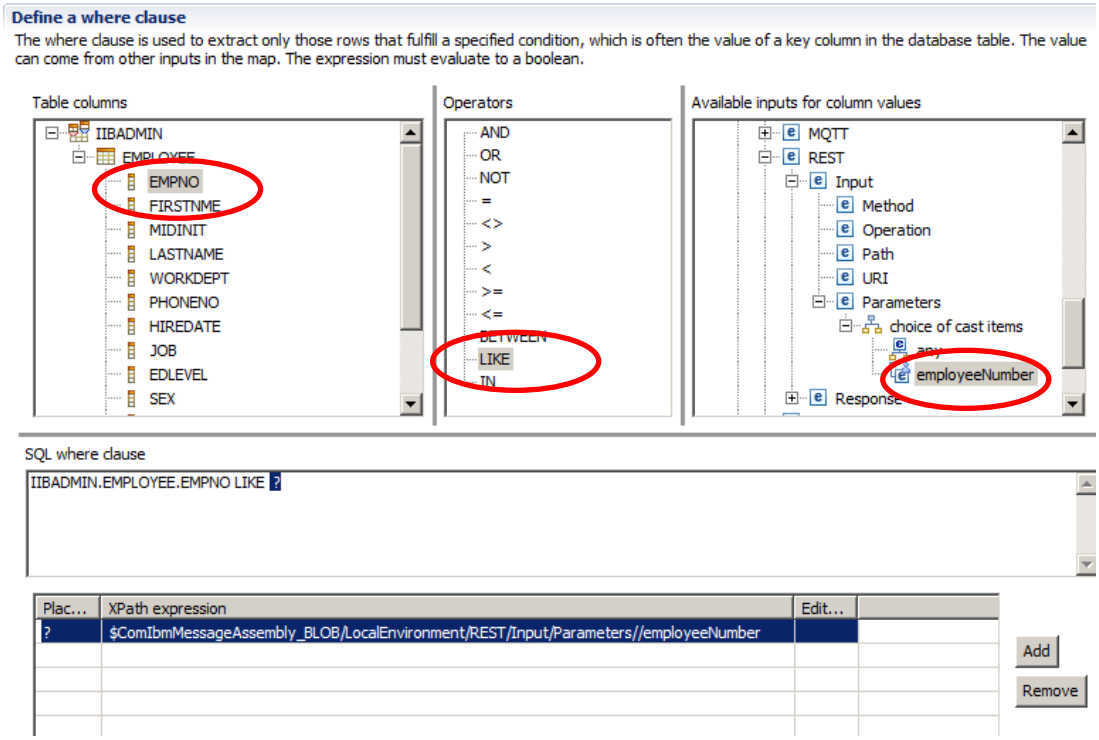
8. Create a new WHERE clause, as follows:

   - Double-click the table column EMPNO
   - Double-click the operator "LIKE".
   - Double-click the Available input
     LocalEnvironment/REST/Input/Parameters/choice…../employeeNumber

The SQL Where clause will be generated as follows:

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

A new XPath expression will have been created so that the SQL clause can reference the required input element from the LocalEnvironment (employeeNumber).



9. Refine the XPath expression.

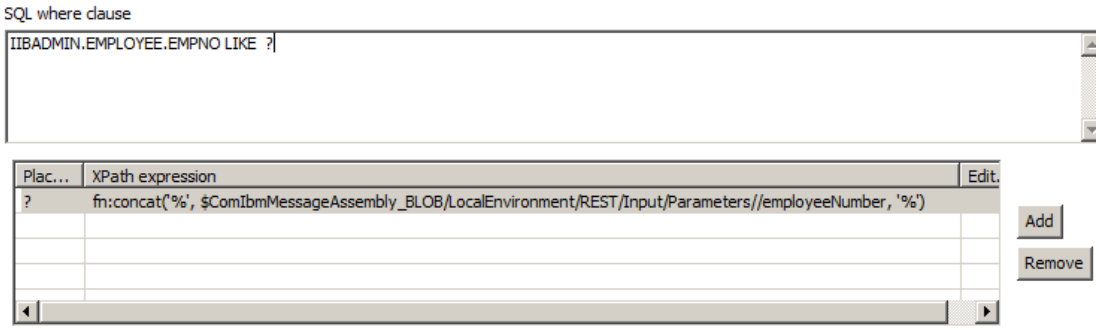
   The map will be built to retrieve all employee records that partially match the provided key. For example, if the provided key is "0020", we want the map to retrieve rows with a value of "000020", "00201", "00204", etc.

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

   To do this, edit the XPath expression as follows:

   ```
   fn:concat('%', $ComIbmMessage...../employeeNumber, '%')
   ```

   The result should look like this:

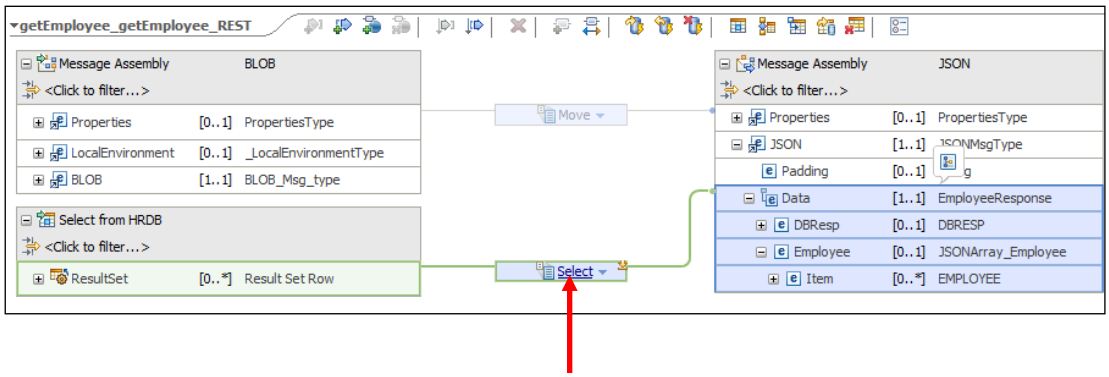


   Now double-check that you made the XPath changes **exactly** as shown above !!

   Click OK to close the Database Select window.

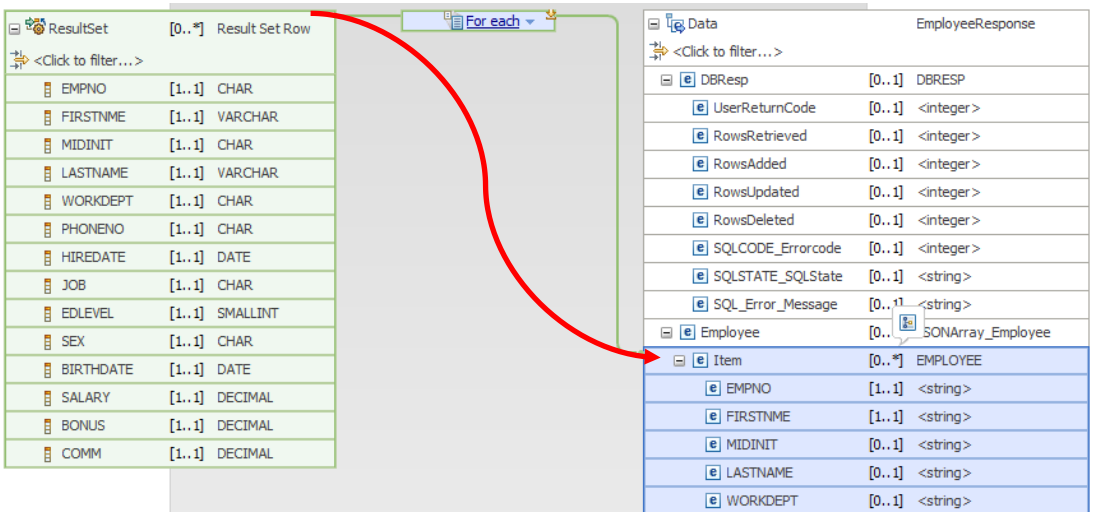
10. A Select transform will have been placed into the map, connecting to the output “Data” element (of type EmployeeResponse). If you didn’t previously highlight the Data element, the connection will be in the wrong place; you can adjust the connector by highlighting it, and dragging to the desired target.

    Click the Select transform (click the Select text, not the surrounding transform icon). This is where you will define the precise element mappings that you need. Clicking Select will take the editor to the next lower logical level of the map (synonymous with a “Do … End” programming construct).



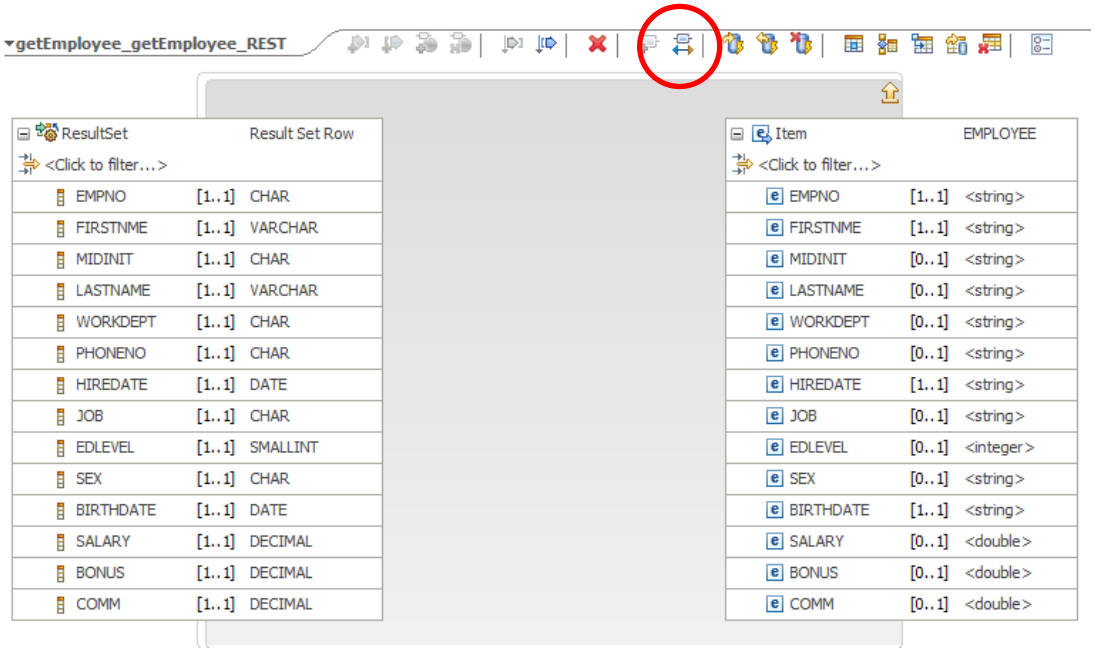
11. Expand the output Employee element to expose the **Item** element.

    Connect **ResultSet** Row to the **Item** array element in the output Data. This will generate a "For each" transform; this means that the transform will be performed for each row returned from the database. However, you still haven't defined the individual element mappings – see the next step.

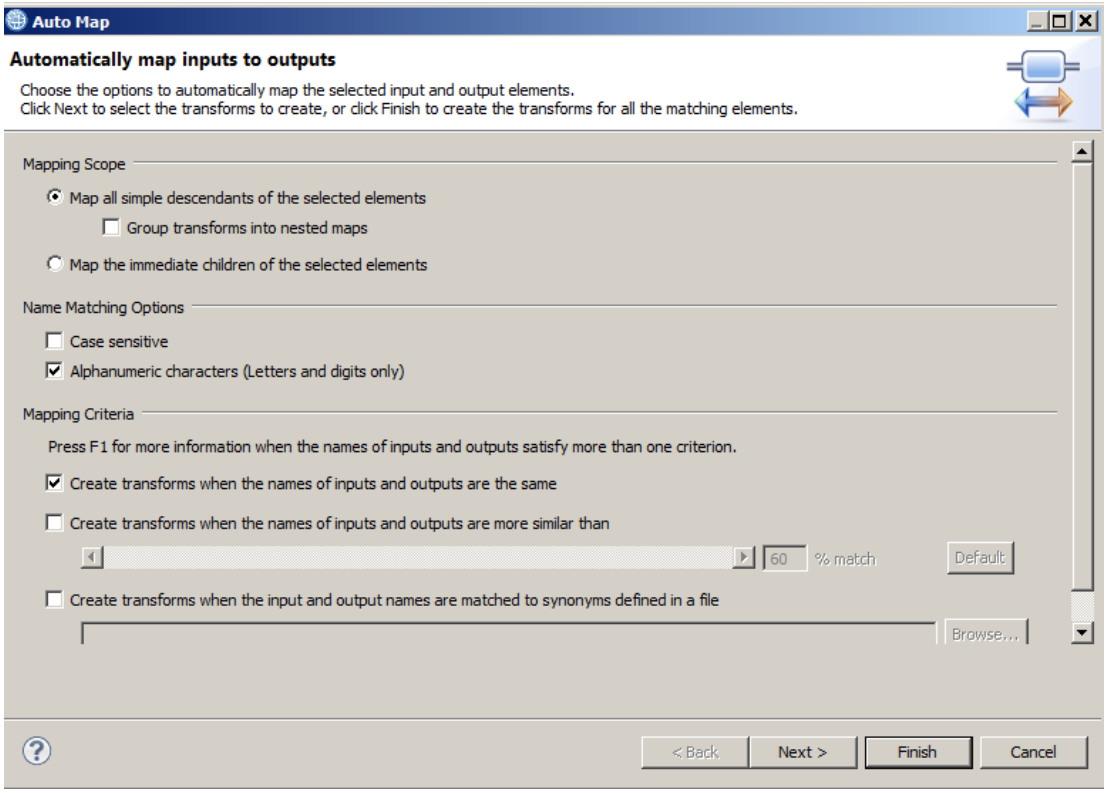


12. Click the text "For each" to specify the individual element mappings. This will take you to the next lower logical level of the map.

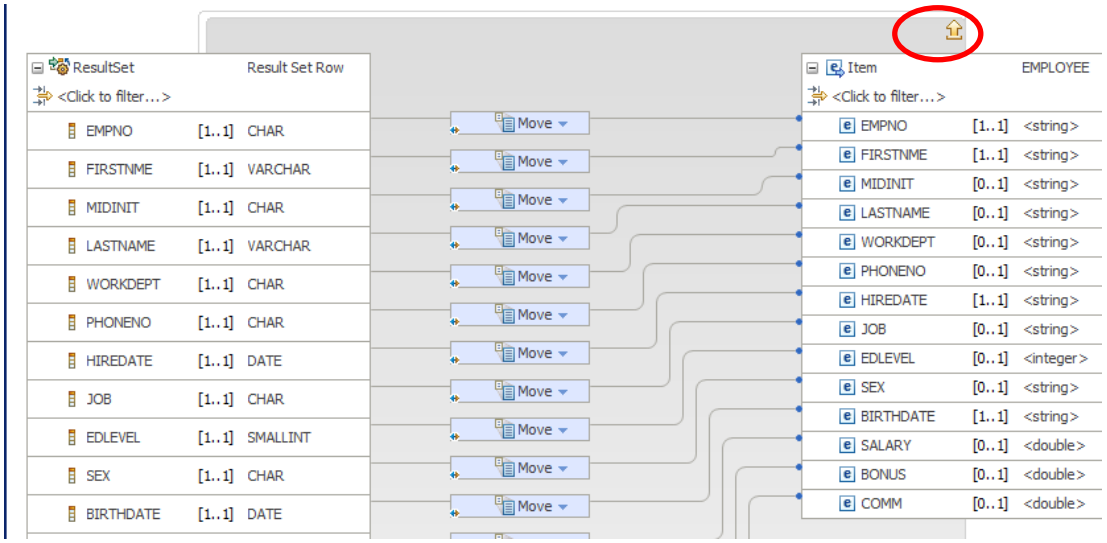
    Click the Automap icon to invoke the wizard to create the mappings automatically.



13. Accept the generated mappings, and click Finish.

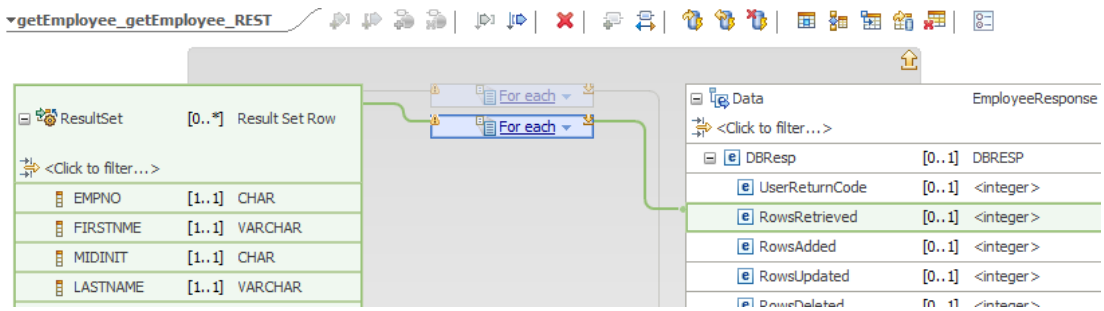


14. Return to the higher level of the map by clicking the yellow up arrow.

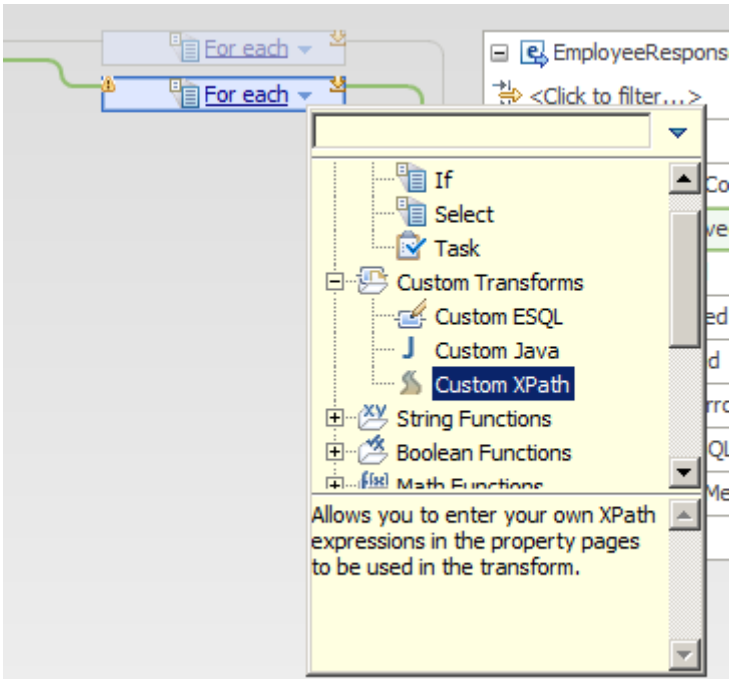


15. We want to set the value for RowsRetrieved.

    Expand DBResp, and connect the input ResultSet to the output DBResp/RowsRetrieved. Again, 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".



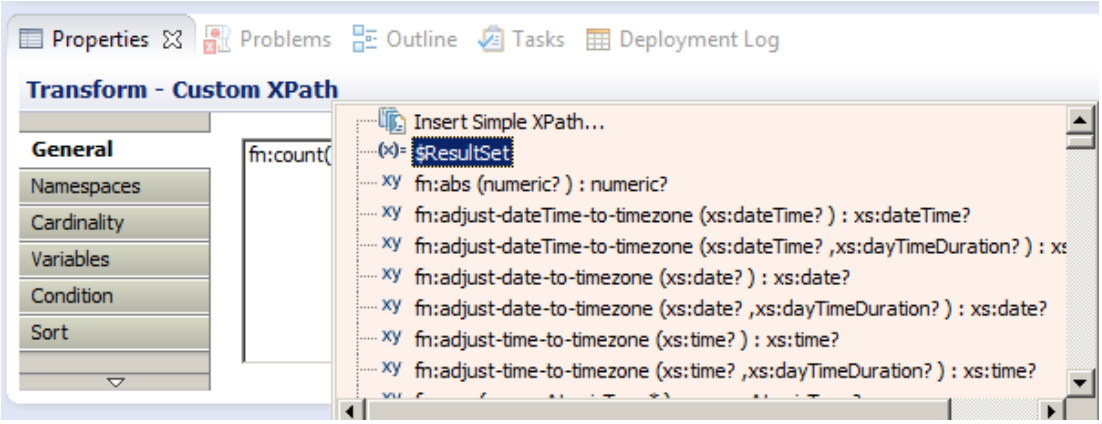
17. In the map editor, make sure the Custom XPath transform is selected.

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

    ```
    fn:count(
    ```

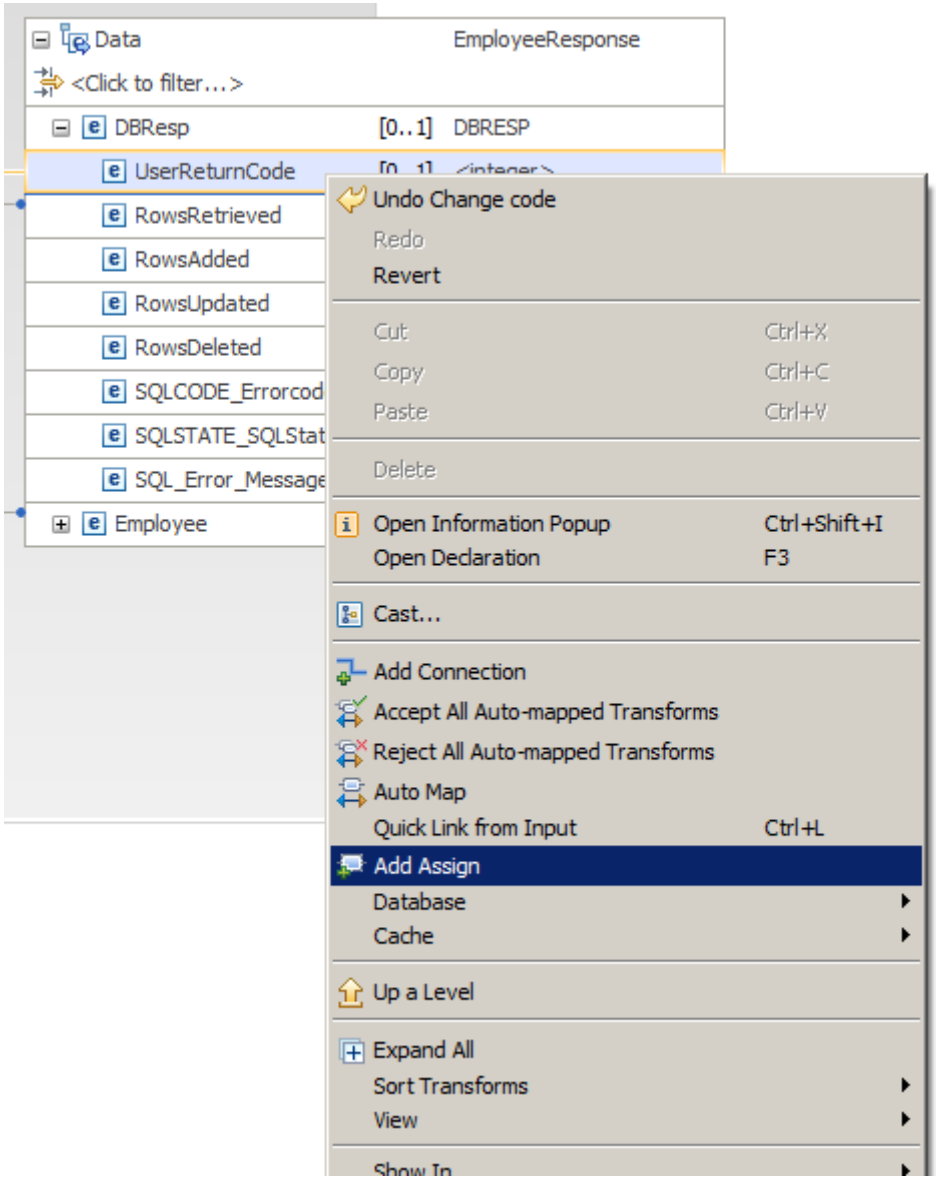
    Then, invoke the content assist function (Ctrl-space) which will show you the possible values for completion. Select the value that shows $ResultSet, or similar. Depending on whether you have made other editing changes, the value shown may be $ResultSet or have a suffixed number. The example shown below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

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

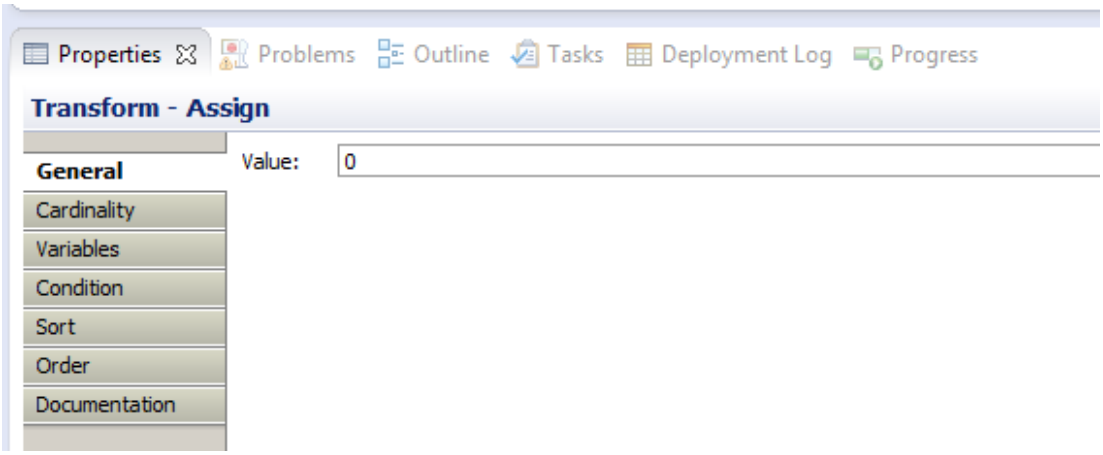


18. Set the value of UserReturnCode to 0. Do this by using an Assign transform.

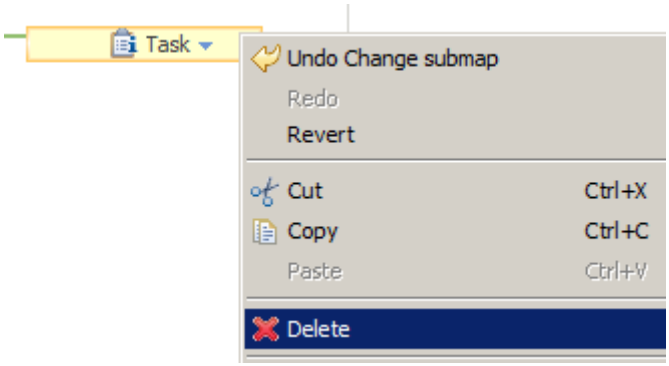
    Expand the output assembly DBResp, and right-click UserReturnCode. Select Add Assign.



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



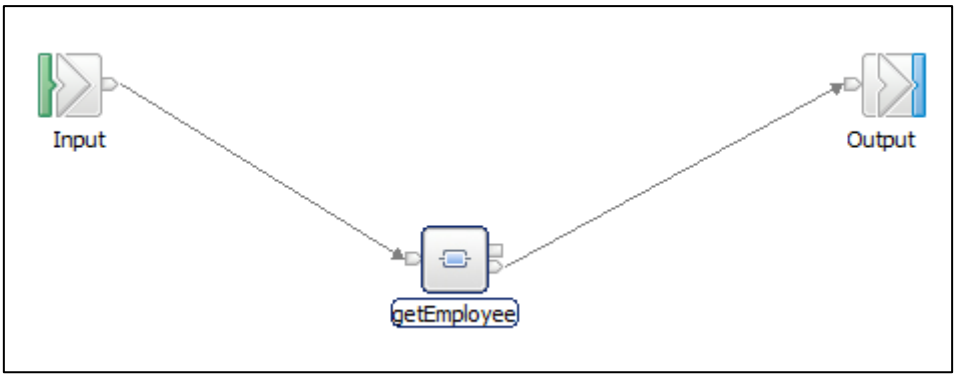
20. Finally, go up one level in the map (yellow up arrow), and delete the Task transform that was added by the Map Wizard.



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

2.6 Complete the subflow

1. Connect the nodes as shown. Save and close the subflow.

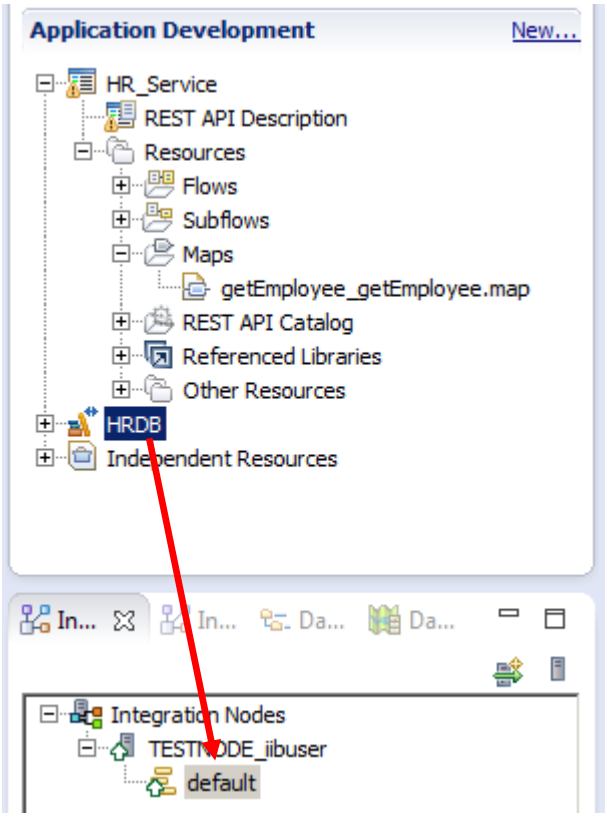


3. Test the HR_Service REST API

This chapter will show you how to use the SwaggerUI tool to send a REST request into the REST API that you have just created.

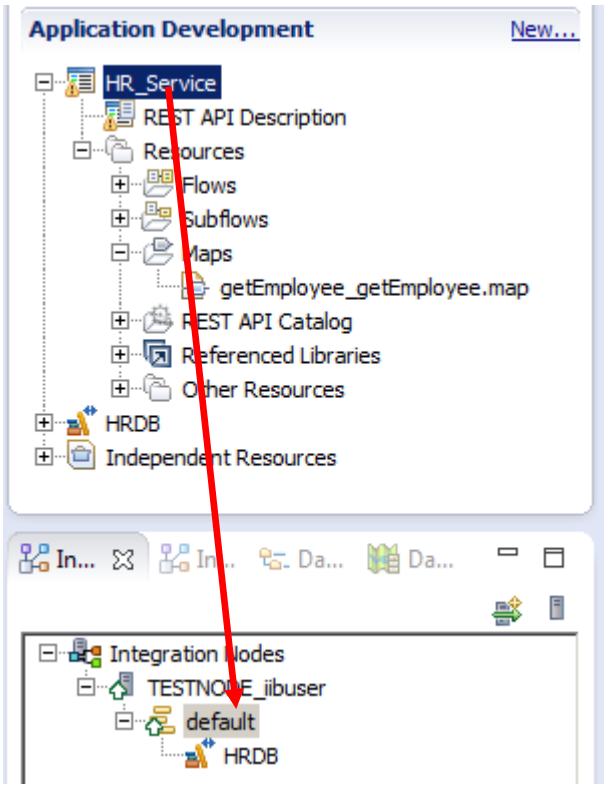
3.1 Deploy the Shared Library

1. In the IIB Toolkit navigator, deploy the HRDB shared library to the default server.



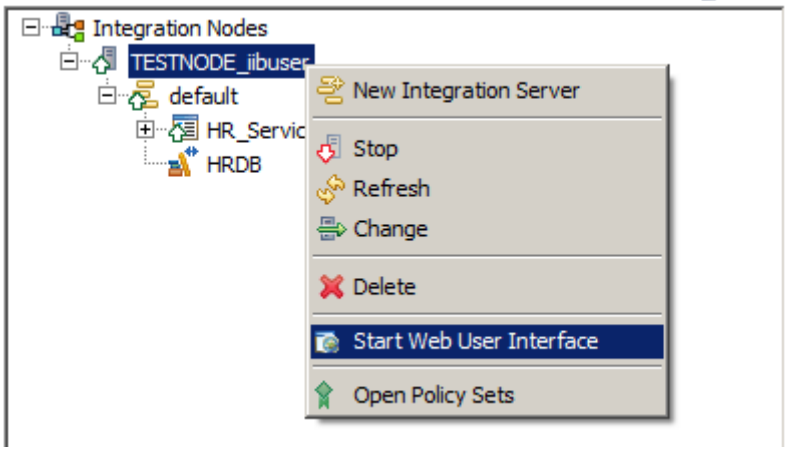
3.2 Deploy the service

2. In the IIB Toolkit navigator, deploy HR_Service to the default server.



3.3 Test the service

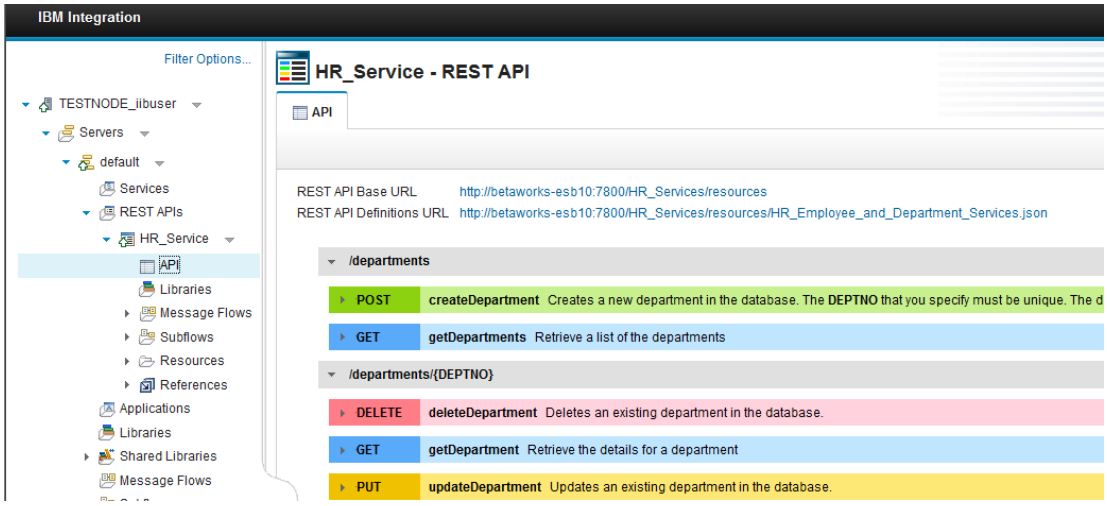
1. Open the IIB web UI by right-clicking TESTNODE_iibuser and selecting Start Web User Interface.



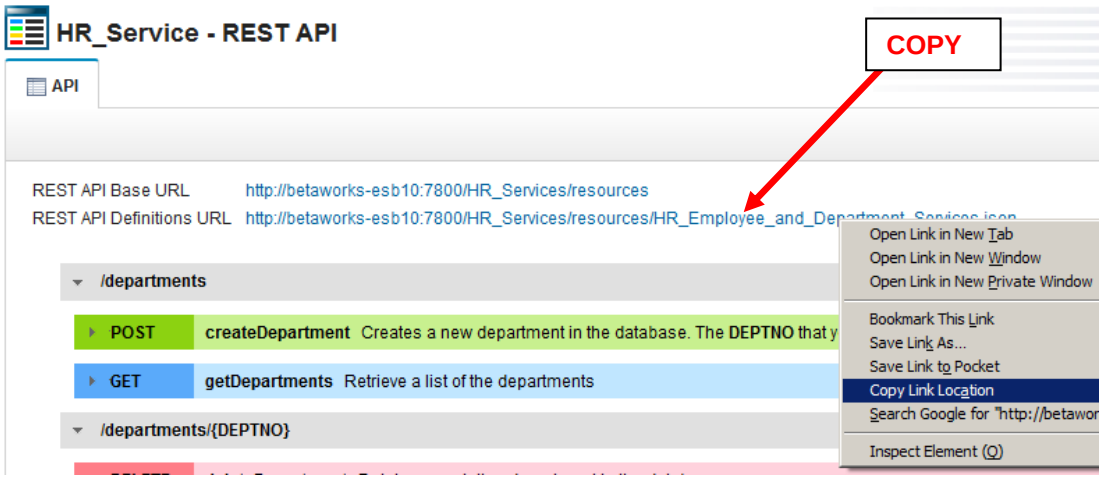
2. You will be switched to the default browser. Fully expand TESTNODE_iibuser, down to the HR_Service REST API, as shown below.

   Under HR_Service, click "API", which will show you the available operations in the REST API, and whether they have been implemented. Check that you have implemented the correct operation – it should be getEmployee.

   It will also show you the URLs for the REST API and the definitions (the .json file).

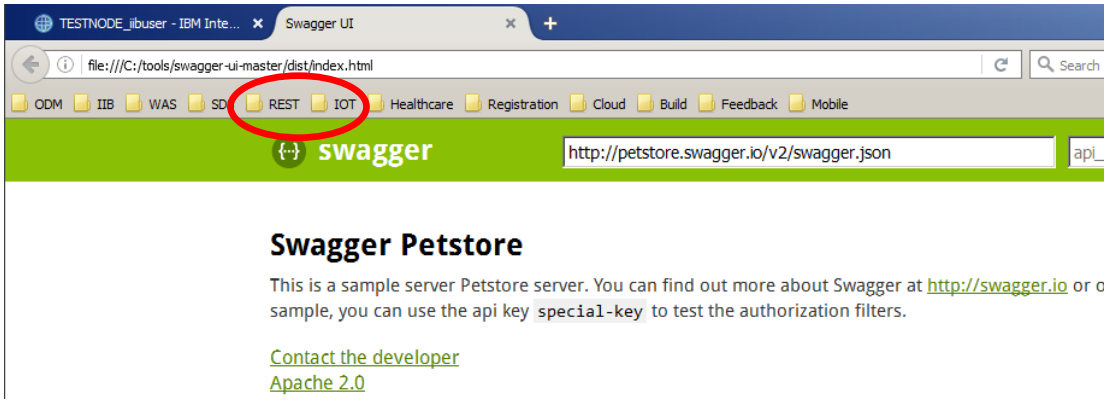


3. On the "REST API Definitions URL", right-click and select "Copy Link Location".



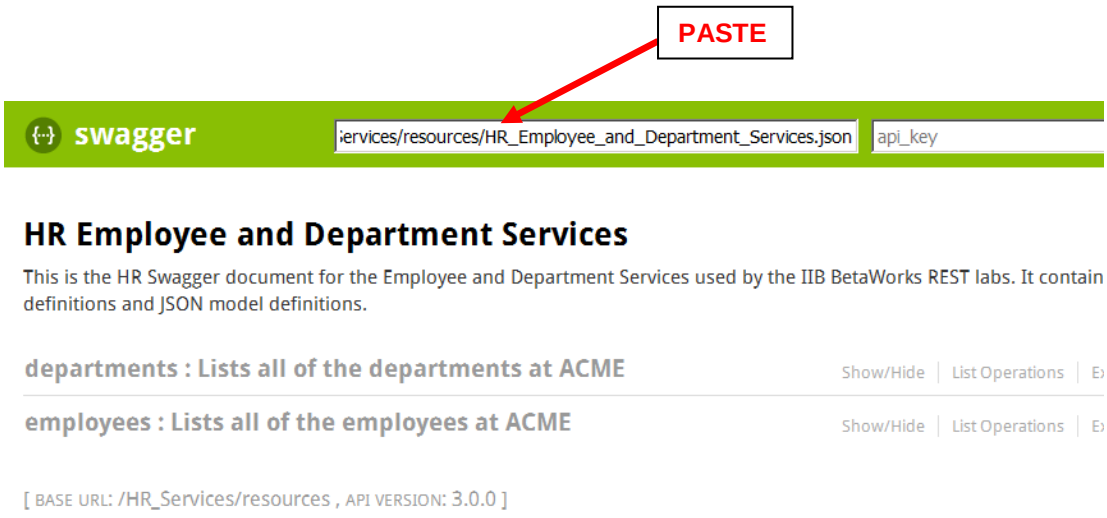
4. In Firefox, open a new tab, and open the SwaggerUI tool (using the bookmark in the REST folder).

   By default, this will open the Petstore Swagger document.



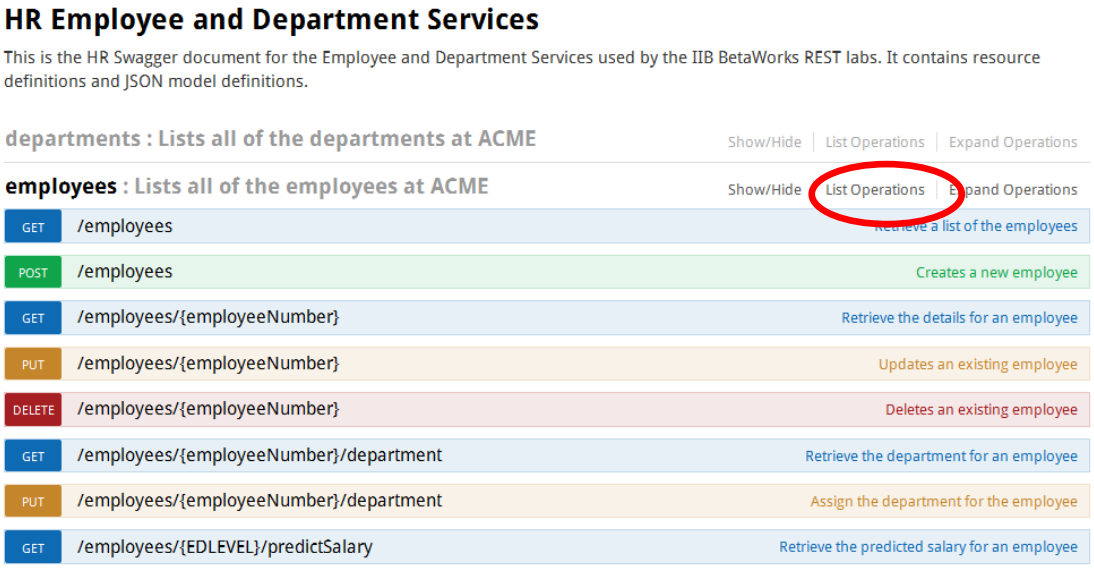
5. In the entry field (not the browser address field), paste the contents of the clipboard and click Explore.

   The two high-level functions, departments and employees, will be shown.



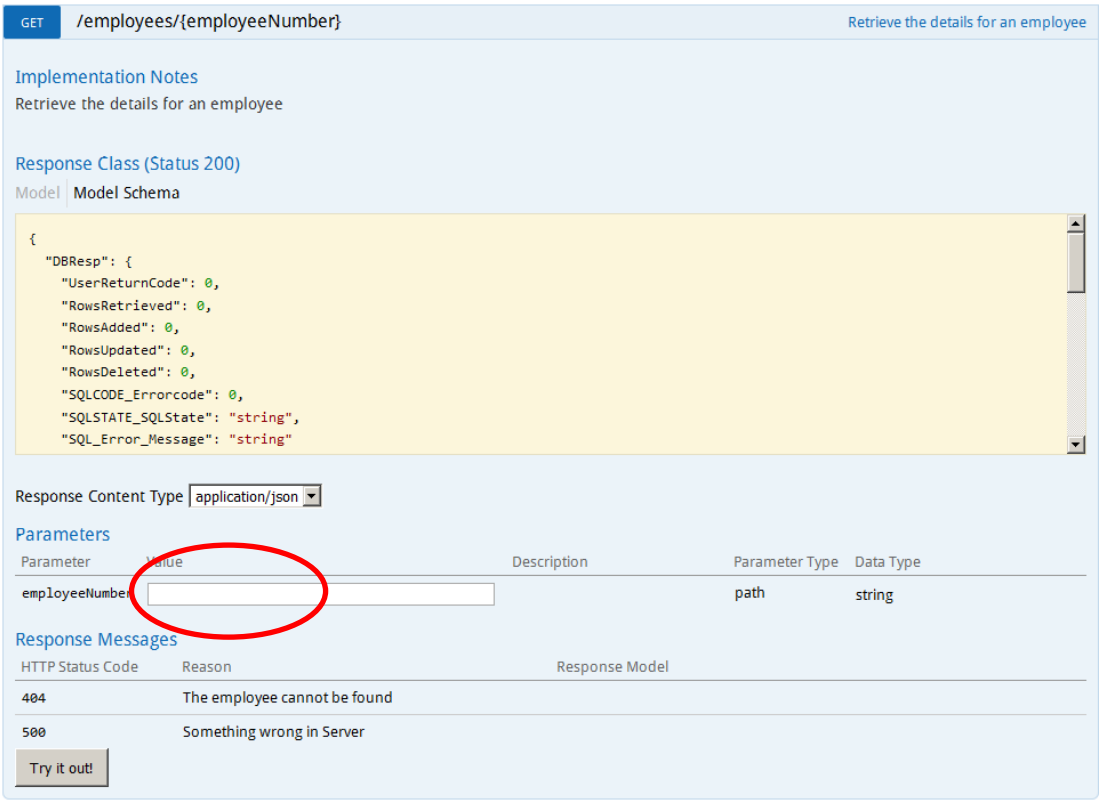
6. We are concerned with the getEmployee operation so click "List Operations" to show the operations related to employees.

   Note that SwaggerUI does not have any knowledge at this point of whether the operation has been implemented.



7. Expand the GET employees/{employeeNumber} operation by clicking it.

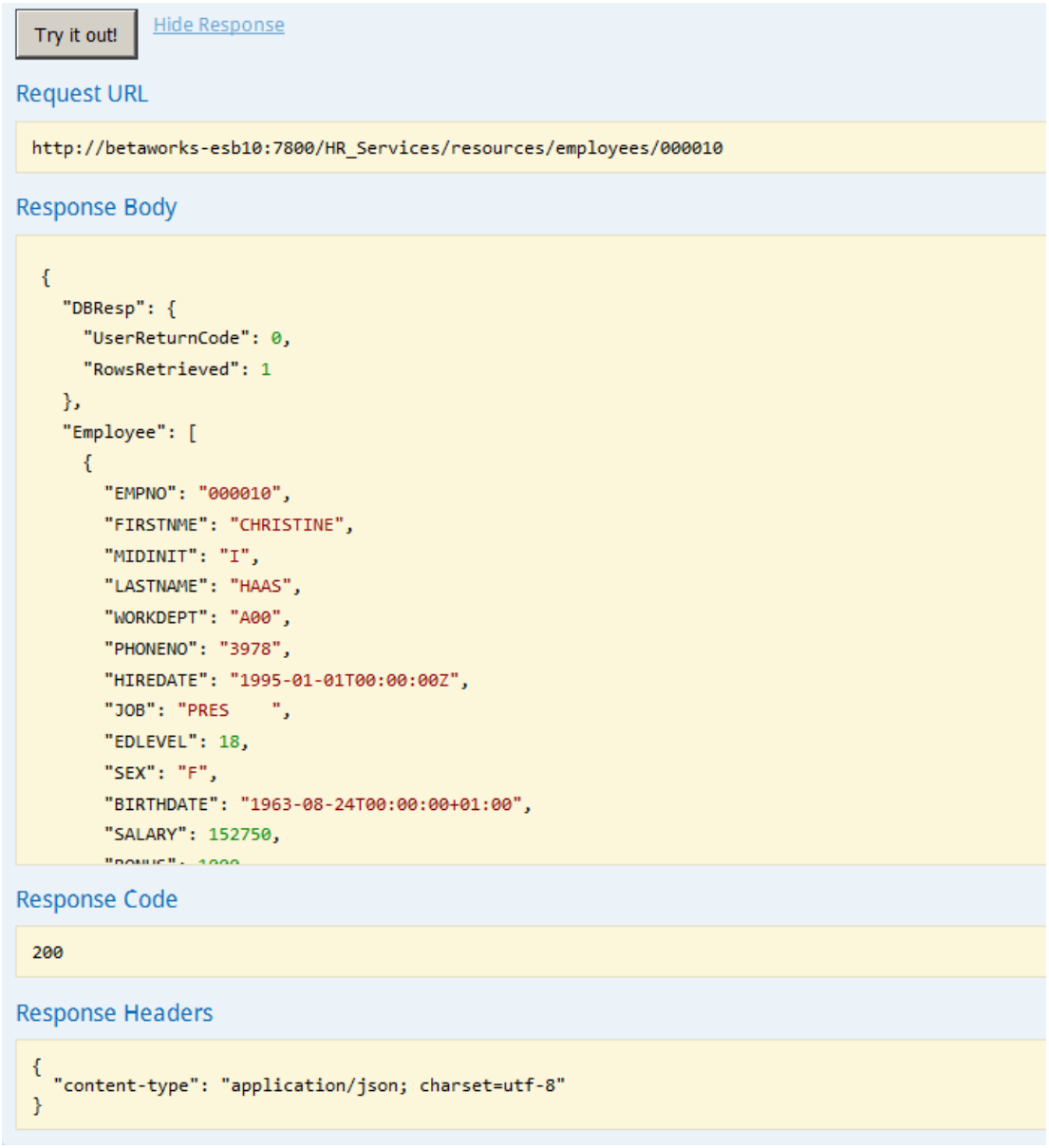
The input parameter is employeeNumber. Provide a suitable value, say 000010.



8. When you have provided an employeeNumber, click Try it out!

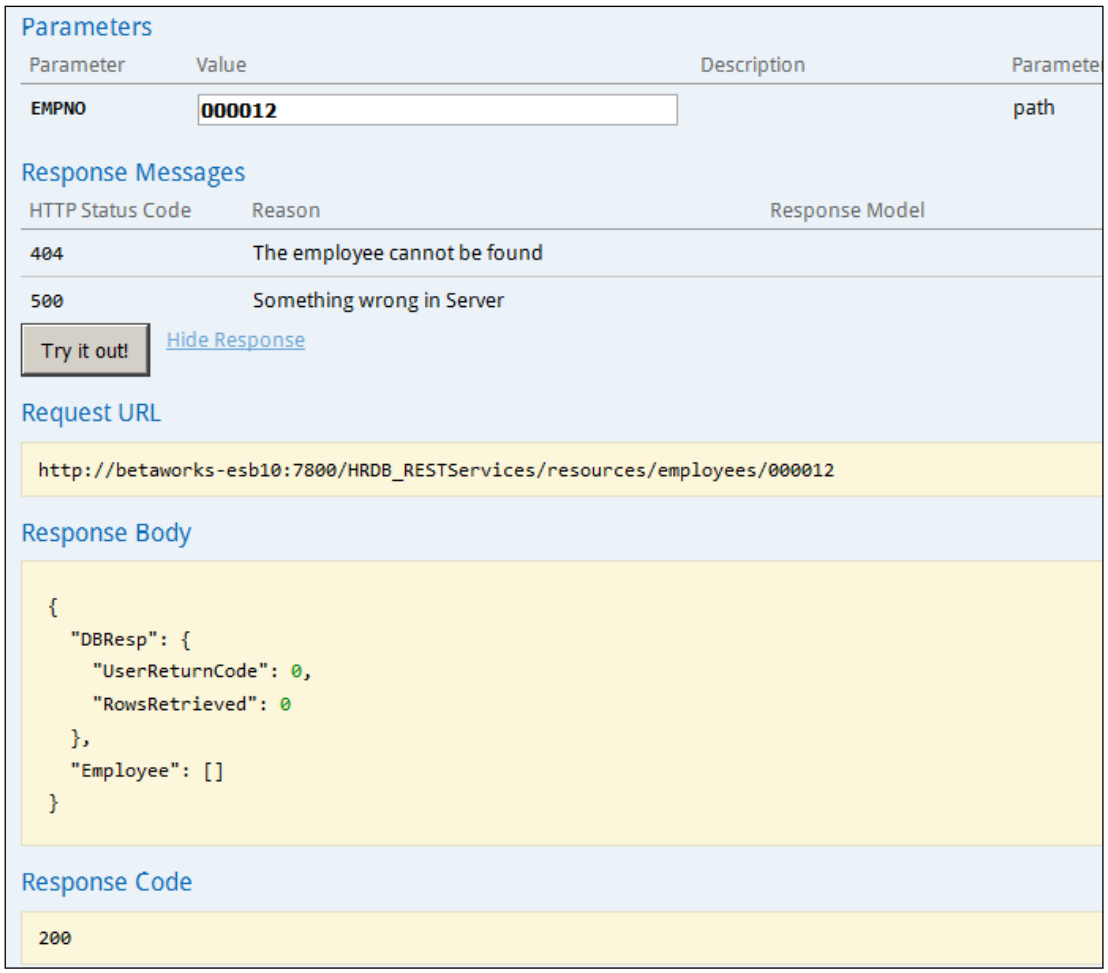
   If successful, the returned data will look something like this. Note the database response information (user return code, number of rows returned), as well as the user data.

   Note that you can copy the Request URL below, and paste directly into a browser.



9. Provide an employeeNumber that does not exist, for example 000012 (but make sure you use an employeeNumber that has 6 characters).

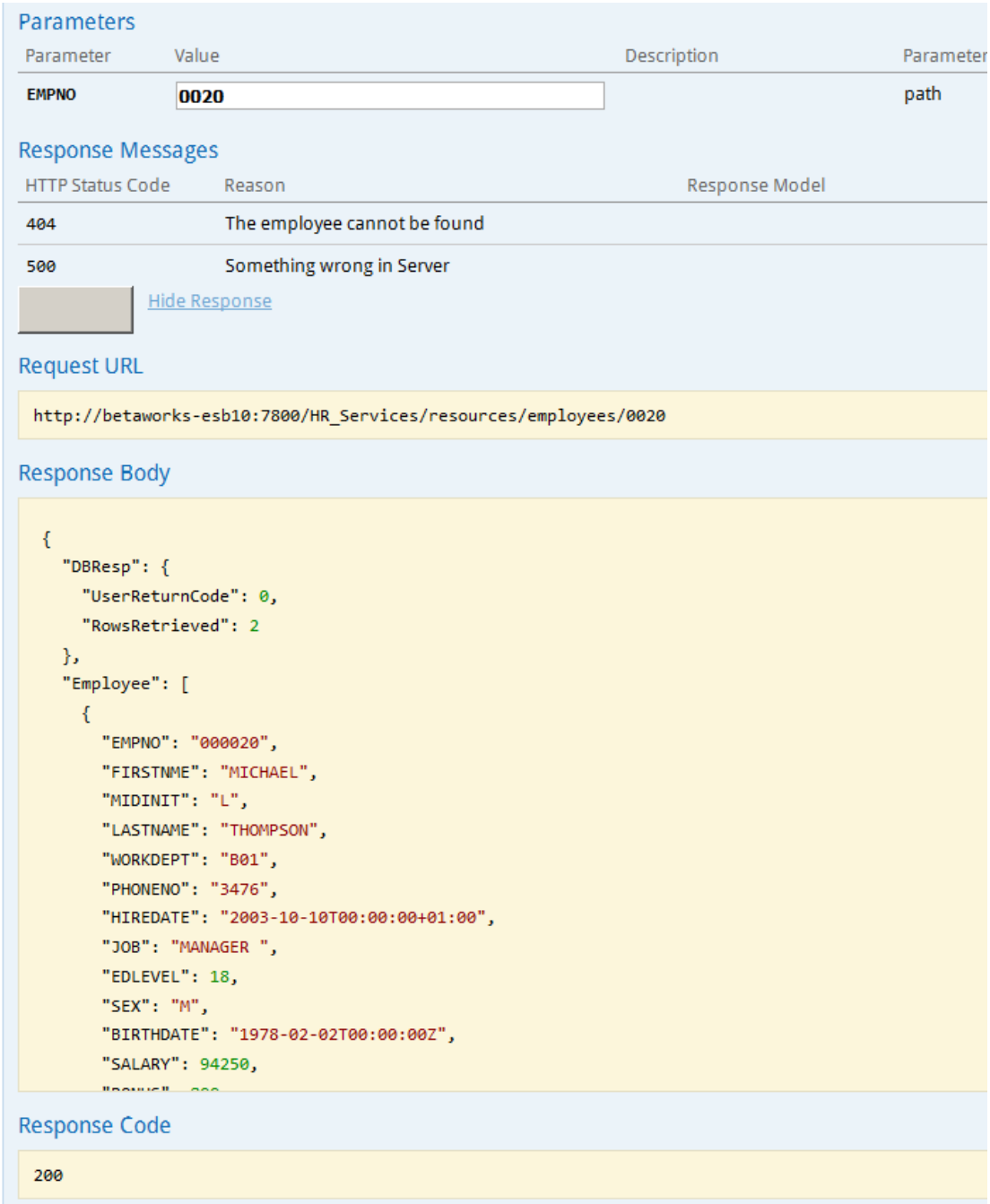
   You will see the service has worked (UserReturnCode = 0), but no data has been found (RowsRetrieved = 0).



10. Provide an employeeNumber that will generate several matches, for example "0020". This will use the "LIKE" operator in the SQL where clause.

    Two records will be retrieved from the database. You will see the element UserReturnCode = 0, and the element RowsRetrieved = 2.

    Use the slide bar to see the data of the two returned rows.



4. Appendix

4.1 Recreating the HRDB database and tables

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, the command **`1_Create_HRDB_database.cmd`** and **`2_Create_HRDB_Tables.cmd`** are provided for this. These files are contained in the c:\student10\Create_HR_Database folder.

If used in conjunction with the workshop VMWare image, these commands must be run under the user "iibadmin". Appropriate database permissions are included in the scripts to GRANT access to the user iibuser.

When an IIB node is created, or recreated, the appropriate JDBC and security definitions must be created. Run the command files 3_Create_JDBC_for_HRDB and 4_Create_HRDB_SecurityId.cmd.

END OF LAB GUIDE