IBM z/OS Connect EE V3.0

Developing RESTful APIs for Db2 Native REST Services

IBM Z
Wildfire Team –
Washington System Center

Lab Version Date: April 6, 2020

Table of Contents

Important: On the desktop there is a file named *Developing APIs CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

Mitch Johnson (mitchj@us.ibm.com)

Overview

Important – You do not need any skills with Db2 to perform this exercise. Even if Db2 is not relevant to your current plans performing this exercise will give additional experience using the API toolkit with developing APIs.

The objective of these exercises is to gain experience with working with z/OS Connect EE and the API toolkit. These two products allow the exposure of z/OS resources to JSON clients. More in-depth information about the customization of z/OS Connect EE, z/OS Connect EE security, the use of the API toolkit and other topics is provided by the 1-day *ZCONNEE – z/OS Connect Workshop.* For information about scheduling this workshop in your area contact your IBM representative.

If you have completed either the developing APIs exercise for MVS Batch or MQ you can start with section *z/OS Connect EE APIs and Db2* on page 8.

General Exercise Information and Guidelines

- ✓ This exercise requires using z/OS user identity *USER1*. The password for this user will be provided by the lab instructor.
- ✓ Any time you have any questions about the use of IBM z/OS Explorer, 3270 screens, features or tools do not hesitate to ask the instructor for assistance.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *Development APIs CopyPaste* file on the desktop
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise.

Mitch Johnson (mitchj@us.ibm.com)

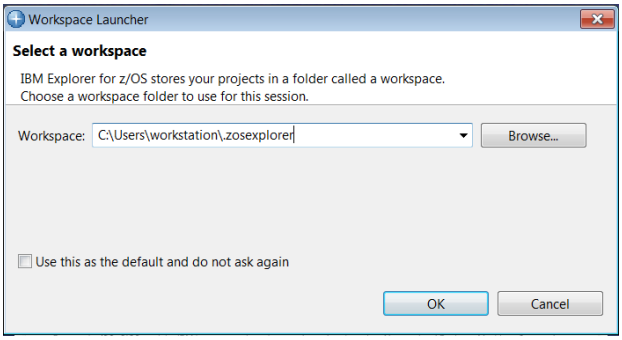
Connect to a z/OS Connect EE Server

Begin by establishing a connection to your z/OS Connect server from IBM z/OS Explorer. If you have performed one of the other exercises in this series of exercises this step may not be required.

> **Tech-Tip:** Windows desktop tools can be opened either by double clicking the icon or by selecting the icon and right mouse button clicking and then selecting the *Open* option.

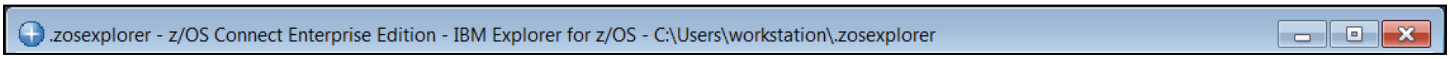
___1. On the workstation desktop, locate the *z/OS Explorer* icon and double click on it to open the Explorer.

___2. You will be prompted for a workspace:



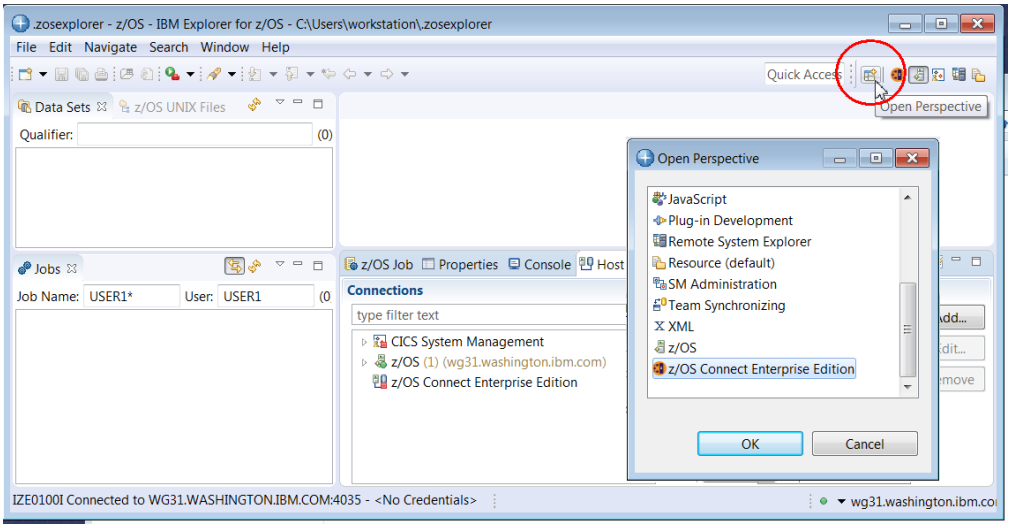
Take the default value by clicking **OK**.

___3. The Explorer should open in the *z/OS Connect Enterprise Edition* perspective. Verify this by looking in the upper left corner. You should see:



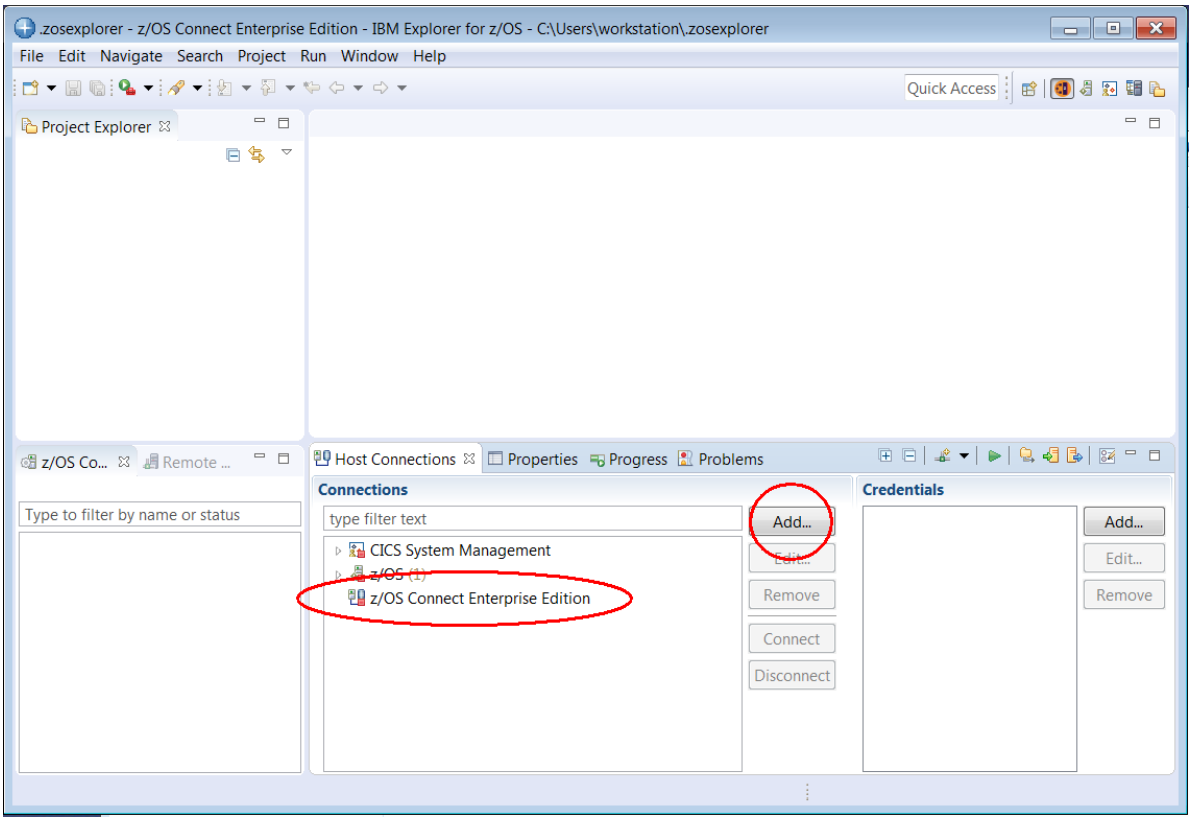
N.B. If a *Welcome* screen is displayed then click the white X beside *Welcome* to close this view.

___4. If the current perspective is not *z/OS Connect Enterprise Edition*, select the *Open Perspective* icon on the top right side to display the list of available perspectives, see below. Select **z/OS Connect Enterprise Edition** and click the **OK** button to switch to this perspective.



Mitch Johnson (mitchj@us.ibm.com)

___5. To add a connection to the z/OS Connect Server select *z/OS Connect Enterprise Edition* connection in the *Host connection*s tab in the lower view and then click the **Add** button.



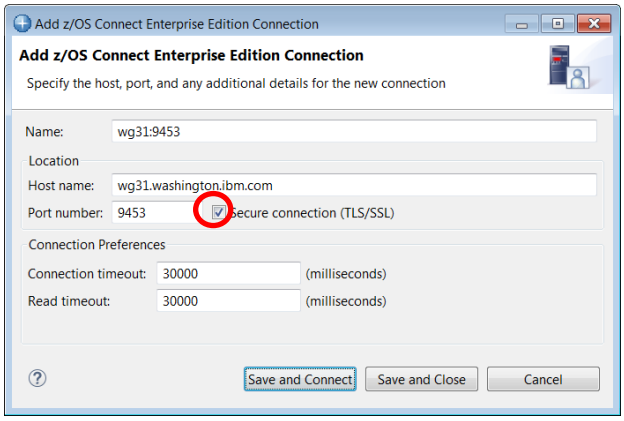
Tech-Tip: Eclipse based development tools like z/OS Explorer; provide a graphical interface consisting of multiple views within a single window.

A view is an area in the window dedicated to providing a specific tool or function. For example, in the window above, *Host Connections* and *Project Explorer* are views that use different areas of the window for displaying information. At bottom on the right there is a single area for displaying the contents of four views stacked together (commonly called a *stacked views*), *z/OS Host Connections, Properties, Progress* and *Problems*. In a stacked view, the contents of each view can be displayed by clicking on the view tab (the name of the view).

At any time, a specific view can be enlarged to fill the entire window by double clicking in the view's title bar. Double clicking in the view's title bar will be restored the original arrangement. If a z/OS Explorer view is closed or otherwise disappears, the original arrangement can be restored by selecting Windows → Reset Perspective in the window's tool bar.

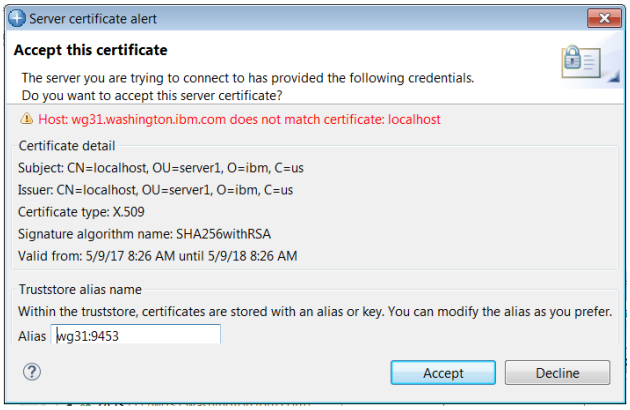
Eclipse based tools also can display multiple views based on the current role of the user. In this context, a window is known as a perspective. The contents (or views) of a perspective are based on the role the user, i.e., developer or administrator.

___6. In the pop-up list displayed select *z/OS Connect Enterprise Edition* and on the *Add z/OS Connect Enterprise Edition Connection* screen enter ***wg31.washington.ibm.com*** for the *Host name*, *9453* for the *Port Number*, check the box for *Secure connection (TLS/SSL)* and then click the **Save and Connect** button.



___7. On the *z/OS Connect Enterprise Edition – User ID* required screen create new credentials for a *User ID* of ***Fred*** and a *Password or Passphrase* of ***fredpwd*** (case matters). Remember the server is configured to use basic security. If SAF security had been enabled, then a valid RACF User ID and password will have to be used instead. Click **OK** to continue.

___8. Click the **Accept** button on the *Server certificate alert – Accept this certificate* screen. You may be presented with another prompt for a userid and password, enter ***Fred*** and ***fredpwd*** again.

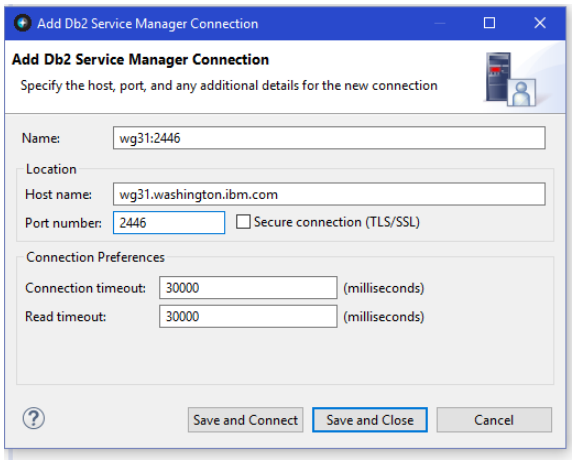


___9. The status icon beside *wg31:9453* should now be a green circle with a lock. This shows that a secure connection has been established between the z/OS Explorer and the z/OS Connect server. A red box indicates that no connection exists.

___10. Next add a connection to the Db2 subsystem. In the *Host connection*s tab in the lower view click the **Add** button.

Mitch Johnson (mitchj@us.ibm.com)

___11. In the pop-up list displayed select *Db2 Service Manager* and on the *Add DB2 Service Manager Edition Connection* screen enter ***wg31.washington.ibm.com*** for the *Host name*, *2446* for the *Port Number*, be sure the box for *Secure connection (TLS/SSL)* is unchecked and then click the **Save and Connect** button.



___12. On the *Db2 Service Manager – User ID* required screen create new credentials for a *User ID* of ***USER1*** and then USER1's password for *Password or Passphrase*. Remember Db2 uses RACF security so a valid RACF User ID and password is required. Click **OK** to continue.

___13. A connection to the remote z/OS system was previously added. In the *Host Connection* view expand *z/OS Remote System* under *z/OS* and select *wg31.washington.ibm.com*. If the connection is not active the **Connect** button will be enabled. Click the **Connect** button and this will establish a session to the z/OS system. This step is required when submitting job for execution and viewing the output of these jobs later in this exercise.

Summary

The next step is the creation of the service and the composing and deployment of the API and then the testing of the API functions.

Mitch Johnson (mitchj@us.ibm.com)

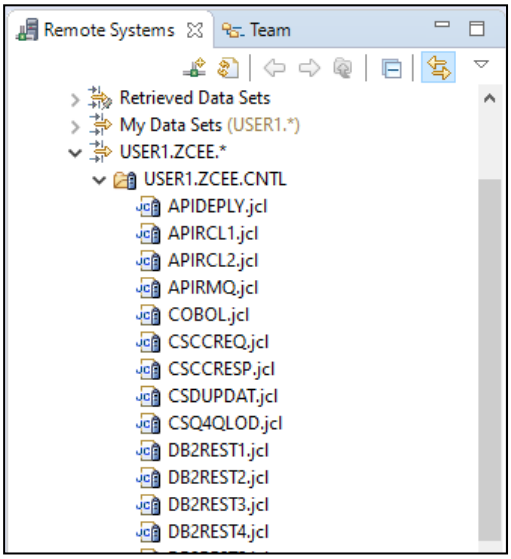
z/OS Connect EE APIs and Db2

Accessing Db2 from z/OS Connect EE differs from the ways in which z/OS Connect EE accesses other z/OS subsystems. The other subsystems are accessed by using standard subsystem interfaces (e.g., WOLA, OTMA, IPIC, JMS, etc.). A z/OS Connect EE server accesses Db2 not as a Db2 client using JDBC but rather as a RESTful client accessing a Db2 native REST Service. This may raise the question as to what value-add does z/OS Connect EE provide if Db2 native REST Services are required for z/OS Connect EE. The answer is that (1) the REST services support provided by Db2 only supports the POST method with only a few administrative services that support the GET method. There is no support for PUT or DELETE methods normally expected for a robust RESTful API service. Another reason (2) is that the API function of transforming JSON request or response messages, e.g. assigning values or removing fields from the interface is not available when using the Db2 native REST Services directly. A Swagger document (3) used for integration into API management products or development tools is only available from z/OS Connect EE. If a full function RESTful API with support for the major HTTP methods (POST, PUT, GET and DELETE) and transforming JSON payloads and generating a Swagger document is required then z/OS Connect EE is the solution. And finally (4) z/OS Connect EE provides security mechanism (e.g. OAUTH and JWT tokens) not available with Db2.

Creating Db2 Native REST Services

Native REST services for Db2 are defined either using a Db2 provided RESTful administrative service (DB2ServiceManager) or by using the Db2 BIND command using an update provided in Db2 PTF UI51748 and APAR PI98649 (PTF UI584231 or UI58425). Only the latter technique that will be shown in this exercise.

___1. In the IBM z/OS Explorer session switch to the *Remote System Explorer* perspective (see page 4) and expand data set *USER1.ZCEE.CNTL* in the *Remote System view* to display a list of its members.



Mitch Johnson (mitchj@us.ibm.com)

___2. Select member *DB2REST1* in *USER1.ZCEE.CNTL* and right- mouse button clicking and submit this job for execution. It should complete with a completion code of 0.

Tech-Tip: To submit a job for execution when using the IBM z/OS Explorer select the member and right mouse button click and select the *Submit* option. Click the **Locate Job** button on the *Job Confirmation* pop-up. This will display the job output in the *Retrieved Job* section under *JES* in the *Remote Systems* view. The job's output can be viewed right mouse button clicking and selecting the *Open* option.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT  DD *
  SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName",
        MIDINIT AS "middleInitial", LASTNAME as "lastName",
        WORKDEPT AS "department", PHONENO AS "phoneNumber",
        JOB AS "job"
  FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN  DD *
 DSN SYSTEM(DSN2)
 BIND SERVICE(SYSIBMSERVICE) -
  NAME("selectEmployee") -
  SQLENCODING(1047) -
  DESCRIPTION('Select an employee from table USER1.EMPLOYEE')
/*
```

This defines a Db2 native REST Services that select a single row from table USER1.EMPLOYEE based on the employee number (column EMPNO).

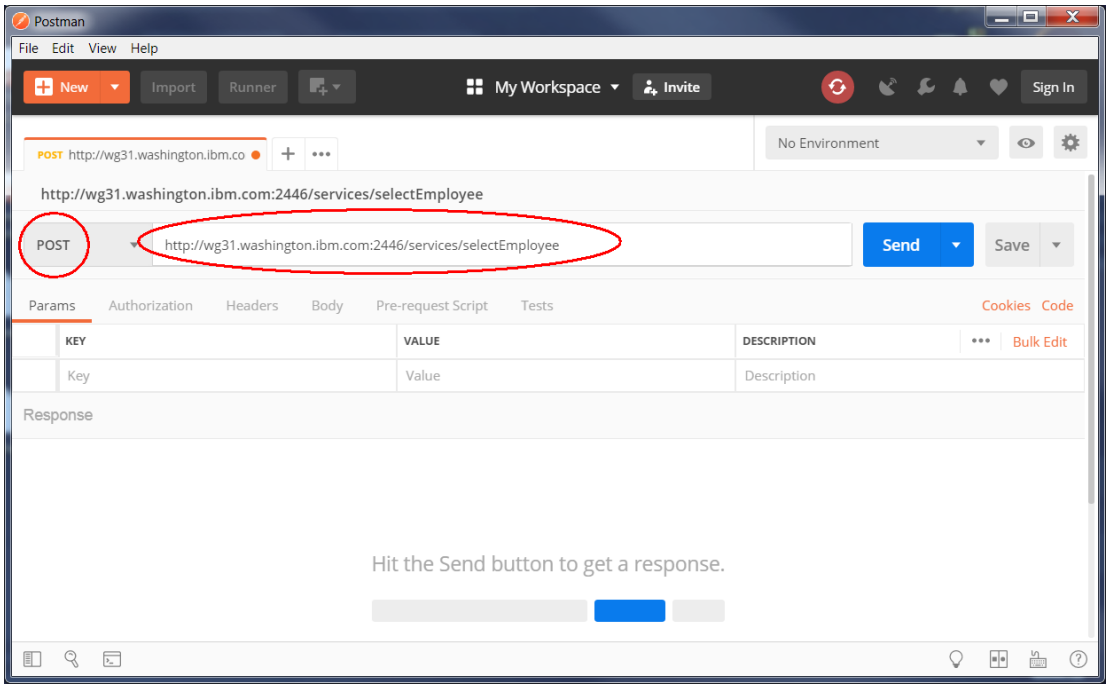
Important: The DBA creating this native Db2 REST service is excluding other table columns, e.g. SEX, SALARY, BONUS, COMMISION, etc. from the selection by omitting these columns from the SELECT statement. The DBA's use of the *AS* cause will also ensure the assigning of meaningful JSON property names rather than the original Db2 column names to the JSON request and response messages.

Tech-Tip: The input to DD DSNSTMT can be a CALL, DELETE, INSERT, SELECT, TRUNCATE, UPDATE, or WITH SQL statement.

To delete a service created by using the Db2 BIND command use the Db2 FREE command, e.g. FREE SERVICE(SYSIBMSERVICE."selectEmployee")

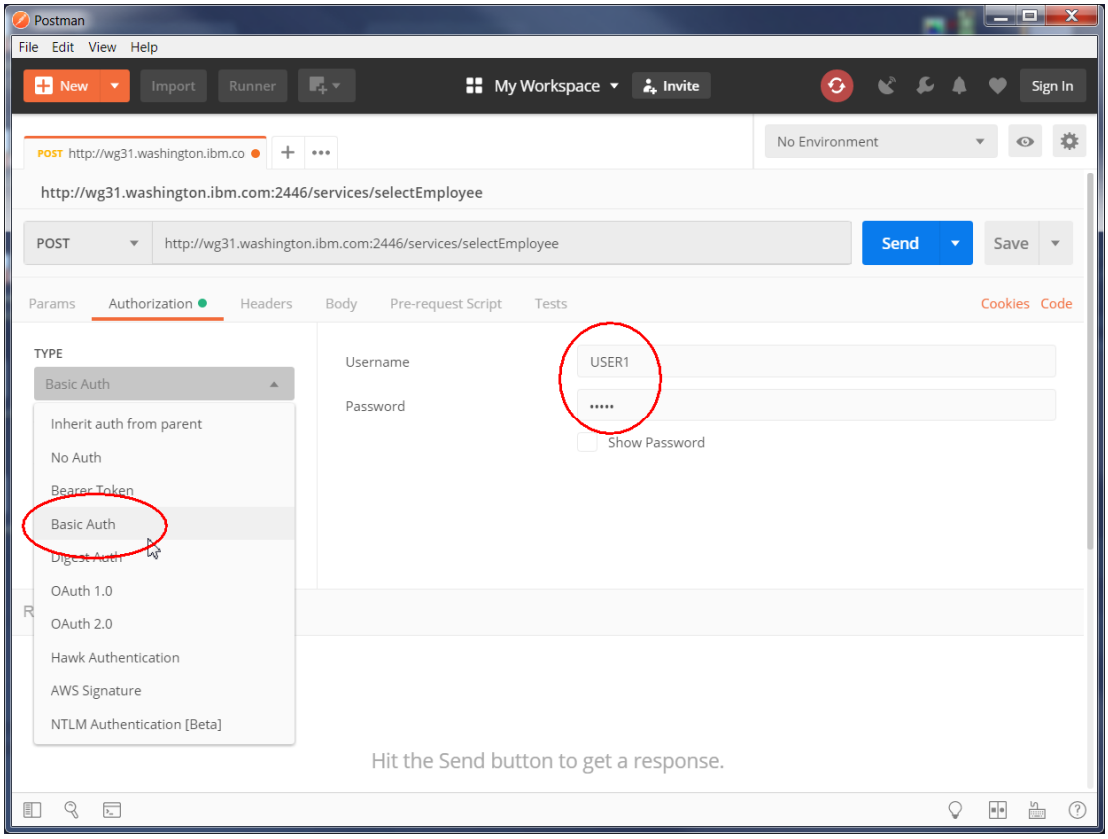
Mitch Johnson (mitchj@us.ibm.com)

___3. Open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages, use the down arrow to select **POST** and enter ***http://wg31.washington.ibm.com:2446/services/selectEmployee*** in the URL area (see below).



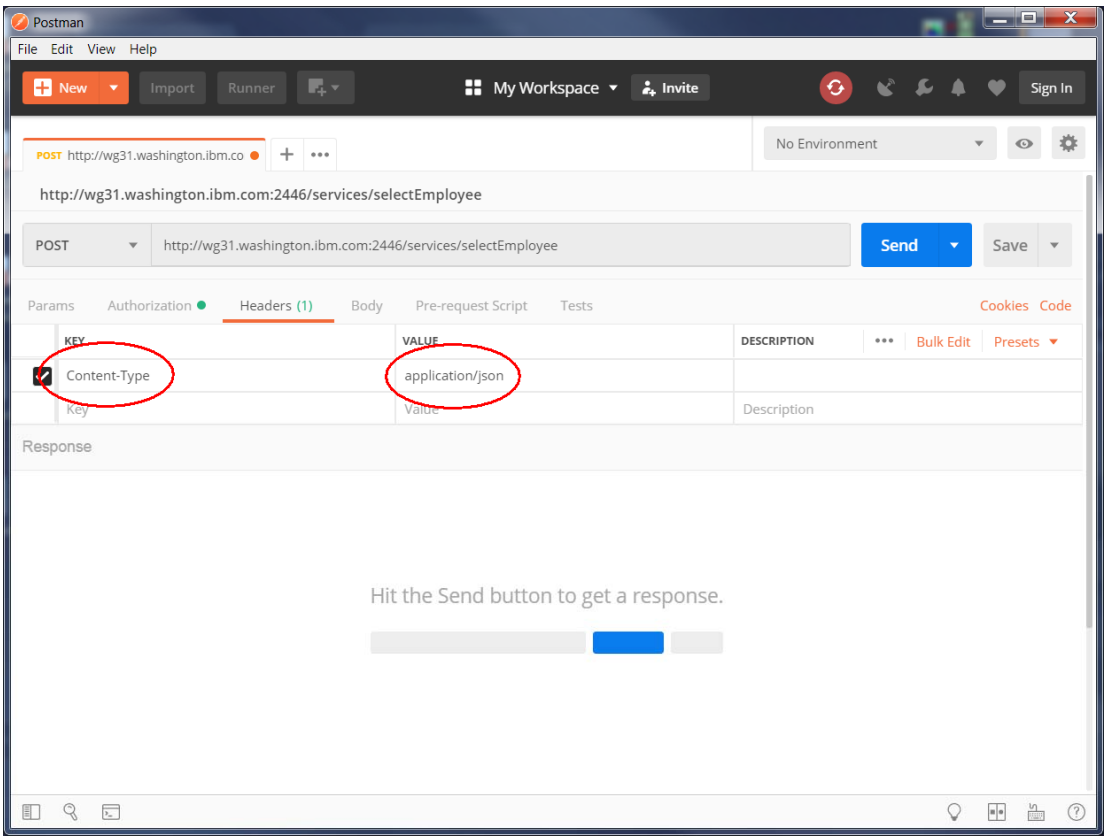
Mitch Johnson (mitchj@us.ibm.com)

___4. No *query* or *path* parameters are required so next select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter ***USER1*** as the *Username* and USER1's password as the *Password*.



Mitch Johnson (mitchj@us.ibm.com)

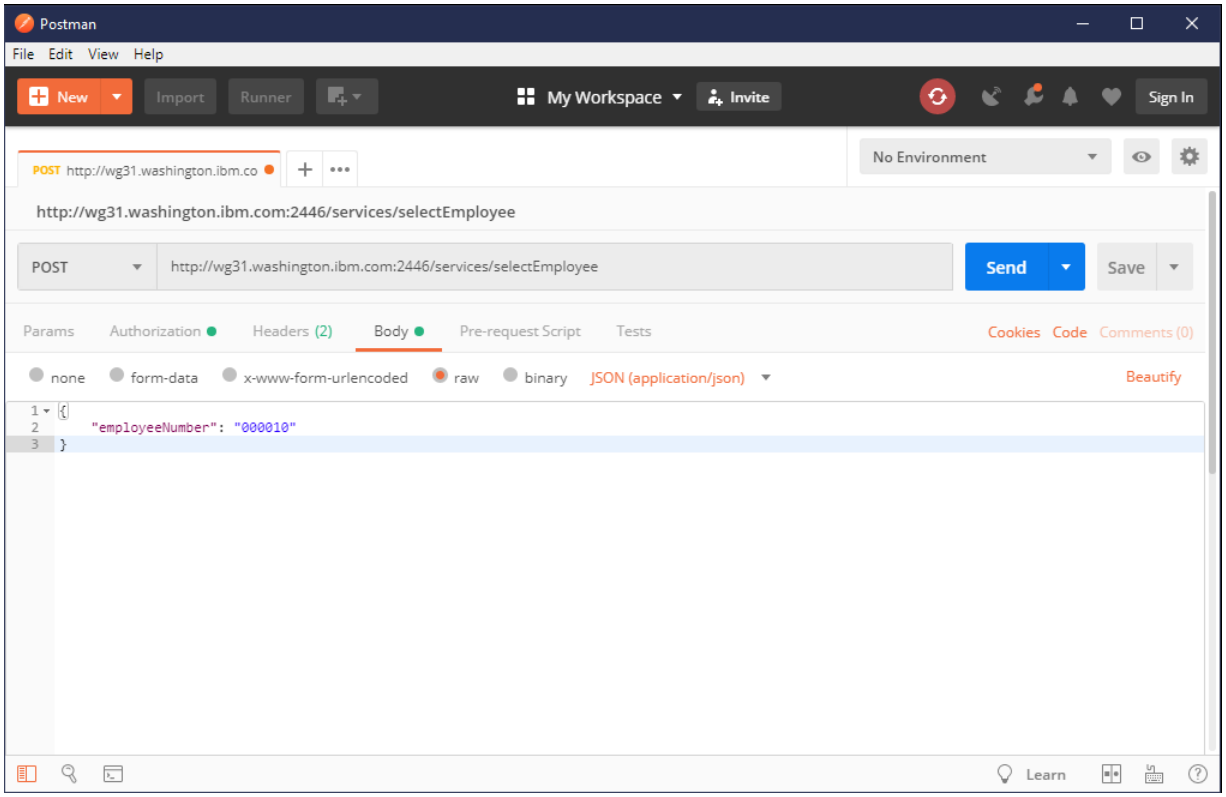
___5. Next select the *Headers* tab and under *KEY* use the code assist feature to enter ***Content-Type*** and under *VALUE* use the code assist feature to enter ***application/json***.



Tech-Tip: Code assist simply means that when text is entered in field, all the valid values for that field that match the typed text will be displayed. You can select the desired value for the field from the list displayed and that value will populate that field.

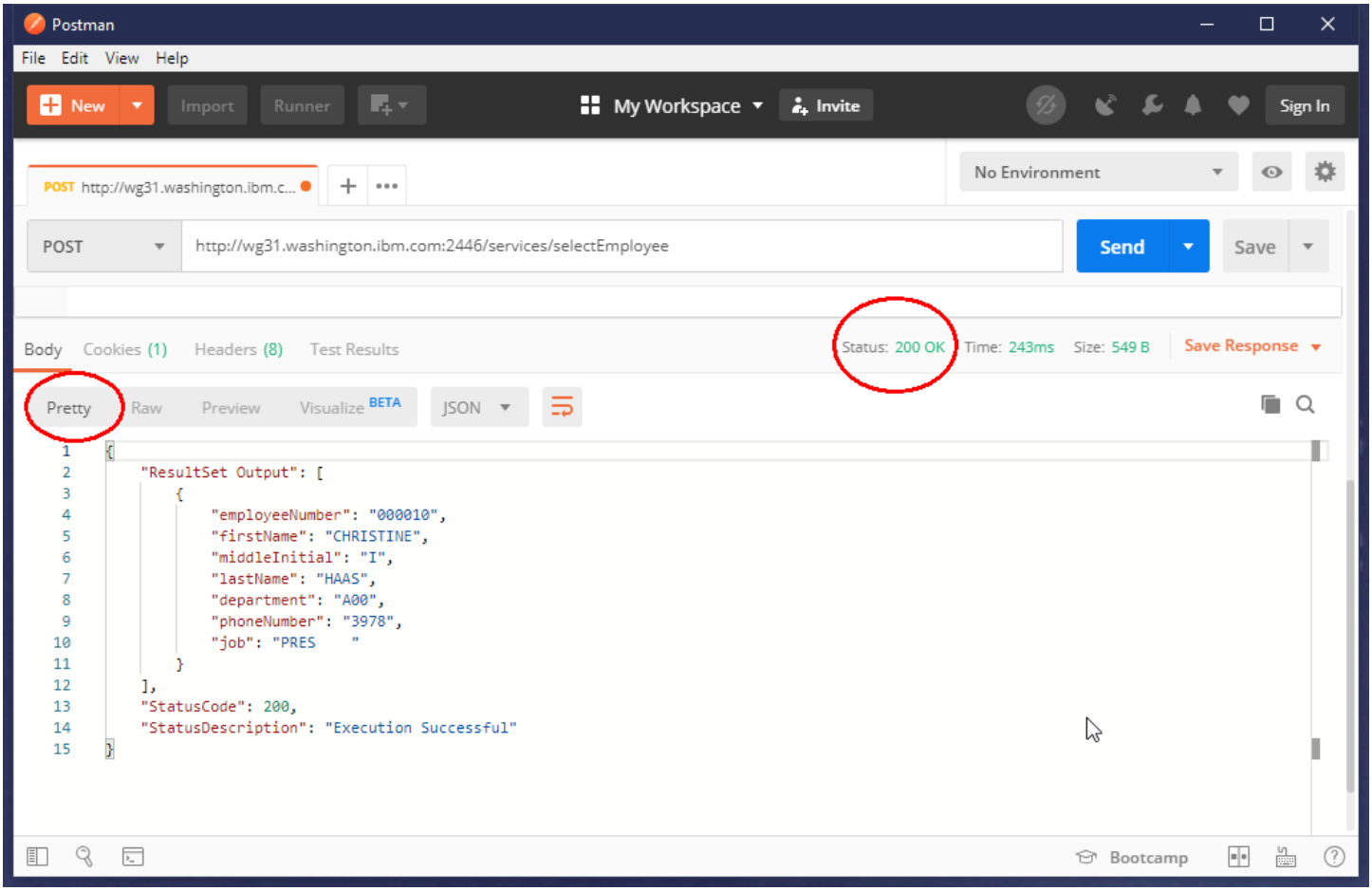
___6. Next select the *Body* tab and select the *raw* radio button and enter the JSON message below in the *Body* area and press the S**end** button.

```
{
      "employeeNumber": "000010"
}
```



Mitch Johnson (mitchj@us.ibm.com)

___7. Pressing the **Send** button invokes the API. The Status of request should be *200 OK* and pressing the *Pretty* tab will display the response message is an easy to read format, see below.



___8. Submit member DB2REST2 in *USER1.ZCEE. CNTL* for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT  DD *
DELETE FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN  DD *
 DSN SYSTEM(DSN2)

 BIND SERVICE(SYSIBMSERVICE) -
  NAME("deleteEmployee") -
  SQLENCODING(1047) -
  DESCRIPTION('Delete an employee from table USER1.EMPLOYEE')
/*
```

This creates a user Db2 native REST Service named *deleteEmployee* that deletes a row from table USER1.EMPLOYEE using the same JSON request message used in Step 6. Optionally test this service by using same Postman session with URL ***http://wg31.washington.ibm.com:2446/services/deleteEmployee*** and the JSON request message below.

```
{
      "employeeNumber": "000020"
}
```

You should see this result in the response area.

```
{
    "Update Count": 1,
    "StatusCode": 200,
    "StatusDescription": "Execution Successful"
}
```

___9. Submit member DB2REST3 in *USER1.ZCEE. CNTL* for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD  DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//DSNSTMT  DD  *
   SELECT EMPNO AS "employeeNumber", FIRSTNME AS "firstName",
         MIDINIT AS "middleInitial", LASTNAME as "lastName",
         WORKDEPT AS "department", PHONENO AS "phoneNumber",
         JOB AS "job"
   FROM USER1.EMPLOYEE WHERE JOB = :job AND WORKDEPT = :department
//SYSTSIN  DD  *
 DSN SYSTEM(DSN2)

 BIND SERVICE(SYSIBMSERVICE) -
  NAME("selectByRole") -
  SQLENCODING(1047) -
  DESCRIPTION('Select an employee based on job and department')
/*
```

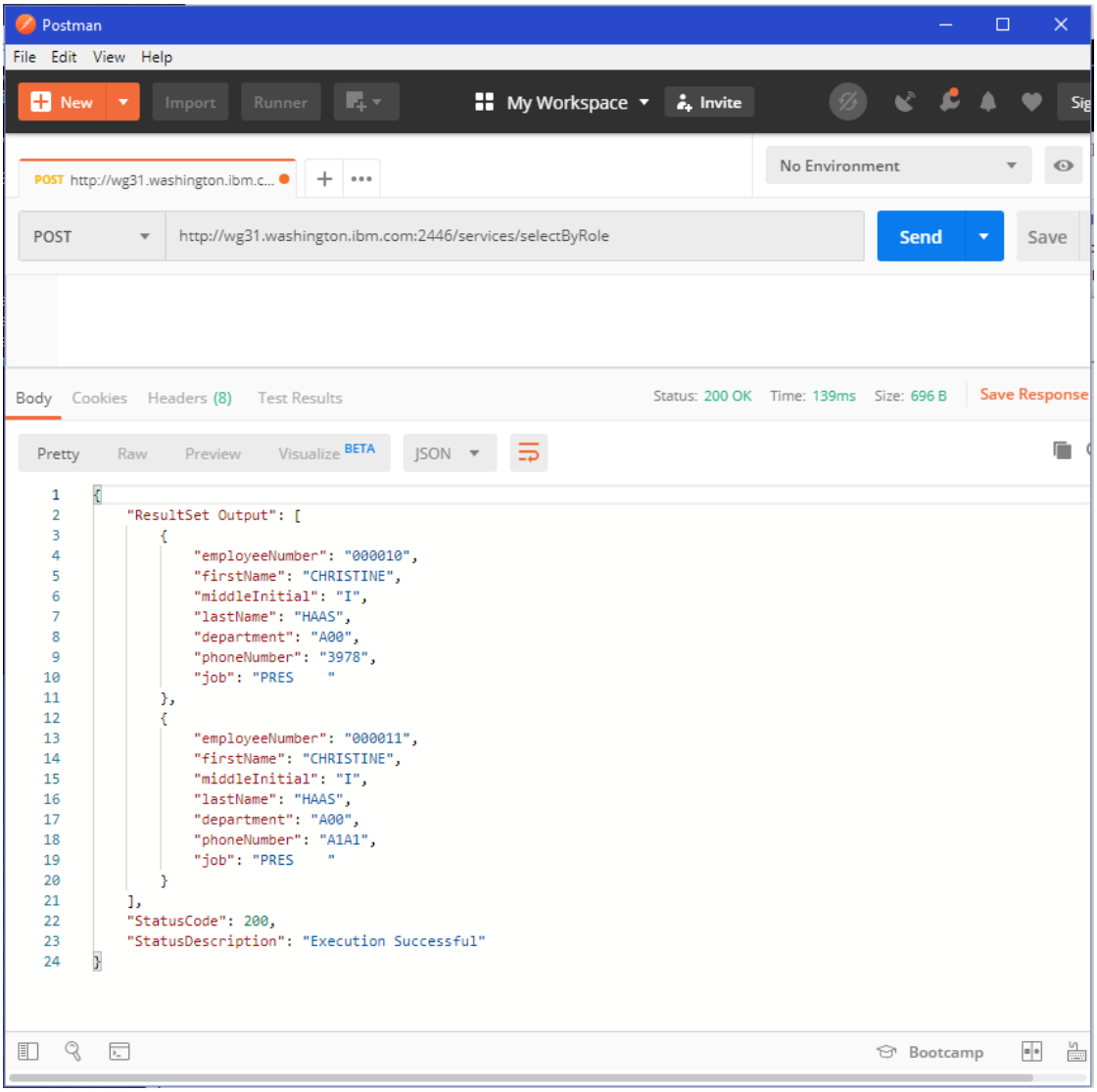
This creates a user Db2 native REST Service named *selectByRole* that selects rows from table USER1.EMPLOYEE based on the contents of the WORKDEPT and JOB columns.

Important: The DBA creating this native Db2 REST service is excluding other table columns, e.g. SEX, SALARY, BONUS, COMMISION, etc. from the selection by omitting these columns from the SELECT statement. The DBA's use of the *AS* cause will also ensure the assigning of meaningful JSON property names rather than the original Db2 column names to the JSON request and response messages.

Mitch Johnson (mitchj@us.ibm.com)

___11. Test this service by using a URL of ***http://wg31.washington.ibm.com:2446/services/selectByRole*** and a JSON request message of:

```
{
      "job": "PRES",
      "department": "A00"
}
```



___12. Submit member DB2REST4 in *USER1.ZCEE.CNTL* for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT  DD *
CALL EMPL_DEPTS_NAT(:whichQuery,:department1,:department2)
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE(SYSIBMSERVICE) -
NAME("selectByDepartments") -
SQLENCODING(1047) -
DESCRIPTION('Select employees by department')       /*
```

This creates a user Db2 native REST Service named *selectByDepartments* that invokes a stored procedure. The stored procedure selects rows from table USER1.EMPLOYEE based on the contents of the query type (WHICHQUERY) and either a specific department (query type 1) or a range of departments (query type not 1).

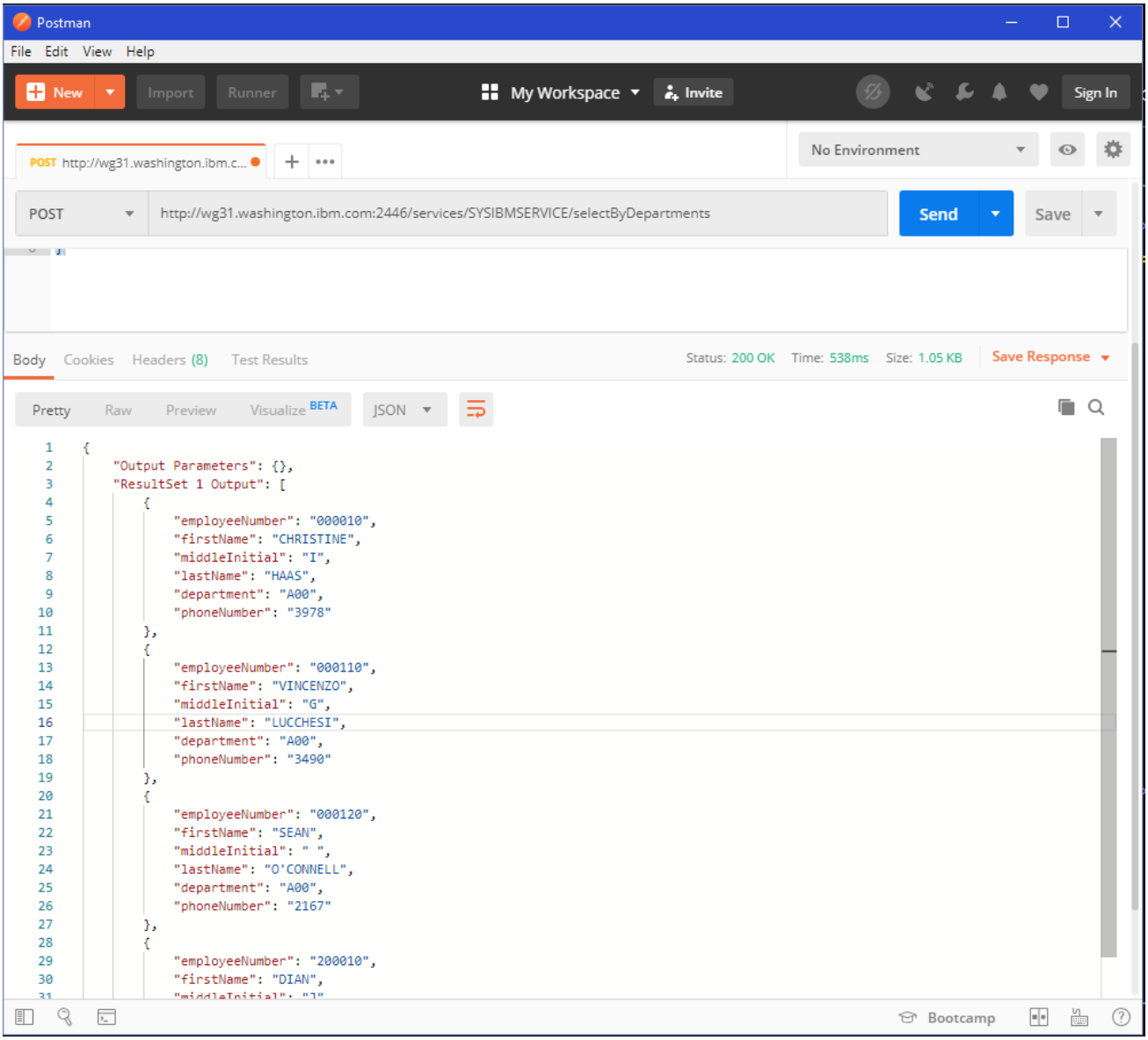
Tech-Tip: The DBA limited the results by not specifying some of the columns.

```
P1: BEGIN
DECLARE CURSOR1 CURSOR WITH RETURN FOR
     SELECT EMPLOYEE.EMPNO AS "employeeNumber",
            EMPLOYEE.FIRSTNME AS "firstName",
            EMPLOYEE.MIDINIT AS "middleInitial",
            EMPLOYEE.LASTNAME AS "lastName",
            EMPLOYEE.WORKDEPT AS "department",
            EMPLOYEE.PHONENO AS "phoneNumber"
            FROM DSN81210.EMP AS EMPLOYEE
            WHERE EMPLOYEE.WORKDEPT=DEPT1
            ORDER BY EMPLOYEE.EMPNO ASC;
DECLARE CURSOR2 CURSOR WITH RETURN FOR
     SELECT EMPLOYEE.EMPNO AS "employeeNumber",
            EMPLOYEE.FIRSTNME AS "firstName",
            EMPLOYEE.MIDINIT AS "middleInitial",
            EMPLOYEE.LASTNAME AS "lastName",
            EMPLOYEE.WORKDEPT AS "department",
            EMPLOYEE.PHONENO AS "phoneNumber"
            FROM DSN81210.EMP AS EMPLOYEE
         WHERE EMPLOYEE.WORKDEPT>=DEPT1 AND EMPLOYEE.WORKDEPT<=DEPT2
ORDER BY EMPLOYEE.WORKDEPT ASC, EMPLOYEE.EMPNO ASC;
CASE WHICHQUERY
    WHEN 1 THEN
        OPEN CURSOR1;
    ELSE
        OPEN CURSOR2;
END CASE;
```


Mitch Johnson (mitchj@us.ibm.com)

___13. Test this service by using a URL of ***http://wg31.washington.ibm.com:2446/services/selectByDepartments*** and a JSON request message of:

```
{
      "whichQuery": "1",
      "department1": "A00",
      "department2": "C01"
}
```



___14. Submit member DB2REST5 in *USER1.ZCEE.CNTL* for execution. It should complete with a completion code of 0.

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//         DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//DSNSTMT  DD *
  INSERT INTO USER1.EMPLOYEE
         (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,PHONENO,
          HIREDATE,JOB,EDLEVEL,SEX,BIRTHDATE,SALARY,BONUS,COMM)
     VALUES (:employeeNumber, :firstName, :middleINit, :lastname,
             :department, :phoneNumber, :hireDate, :job,
             :educationLevel, :sex, :birthDate,
             :salary, :bonus, :commission)
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE(SYSIBMSERVICE) -
NAME("insertEmployee") -
SQLENCODING(1047) -
DESCRIPTION('Insert an employee into table USER1.EMPLOYEE')
/*
```

This creates a user Db2 native REST Service named *insertEmployee* that inserts a new row into table USER1.EMPLOYEE).

Tech-Tip: The host variables specified in the VALUES clause will determine the JSON request and response message property names.

Mitch Johnson (mitchj@us.ibm.com)

___15. Test this service by using a URL of ***http://wg31.washington.ibm.com:2446/services/insertEmployee***

```
{
      "employeeNumber": "999999",
      "firstName": "Matt",
      "middleInitial": "T",
      "lastName": "Johnson",
      "department": "A00",
      "phoneNumber": "9999",
      "hireDate": "2013-01-01",
      "job": "Staff",
      "educationLevel": "27",
      "sex": "M",
      "birthDate": "1985-06-18",
      "salary": "100000",
      "bonus": "15000",
      "commission": "10000"
}
```

You should see this result in the response area.

```
{
    "Update Count": 1,
    "StatusCode": 200,
    "StatusDescription": "Execution Successful"
}
```

This completes the creation of the Db2 native REST Services that will be used in an API. In the next section, the Service Archives need for working with the API Editor will be created and deployed.

z/OS Connect EE APIs and Db2

This section of the exercise provides an opportunity to compose and deploy an API that accesses Db2.

Four z/OS Connect services will be created with each corresponding to one of the Db2 native REST services created earlier.

These 4 services will then be integrated into a RESTful API.

Create the Db2 Service Projects

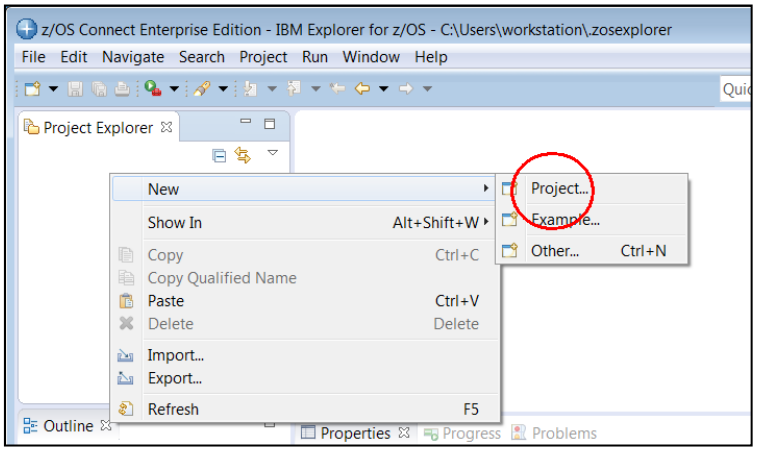
The first step is to create the 4 z/OS Connect services which provide the interaction with the 4 Db2 native REST services. Each service will correspond to the select or delete functions described above.

Switch to the *z/OS Connect Enterprise Edition* perspective. Start by creating the *selectEmployee* service.

___1. Select ***File*** on the tool bar and then on the pop up select **New → Project**. Expand the *General* folder and select *Project* to create a target project for exporting the Service Archive (SAR) files. Click **Next** to continue.

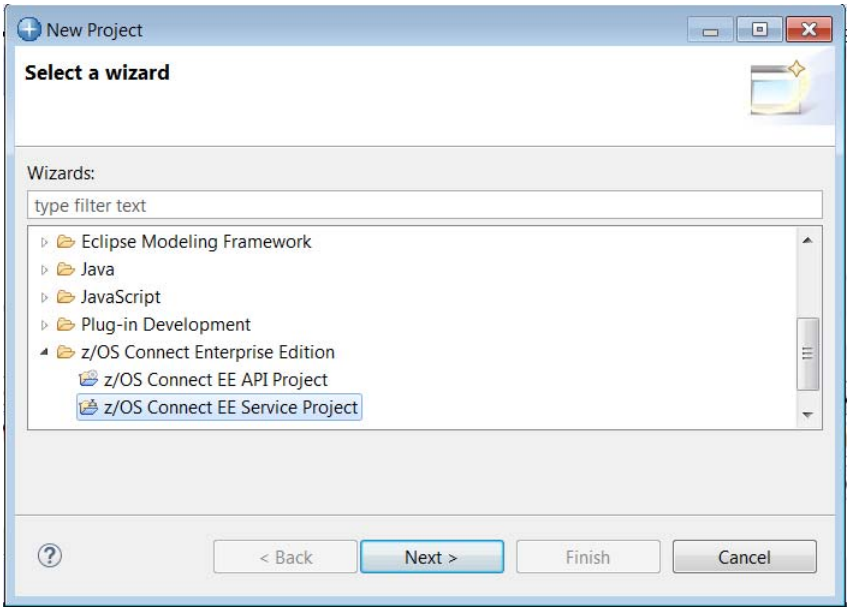
___2. On the *New Project* window enter ***Services*** as the *Project name*. Click **Finish** to continue. This action will add a new project in the *Project Explorer* named *Services*. If this project already exists continue with Step 3.

___3. In the upper left, position your mouse anywhere in the *Project Explorer* view and right-mouse click, then select *New →Project*:

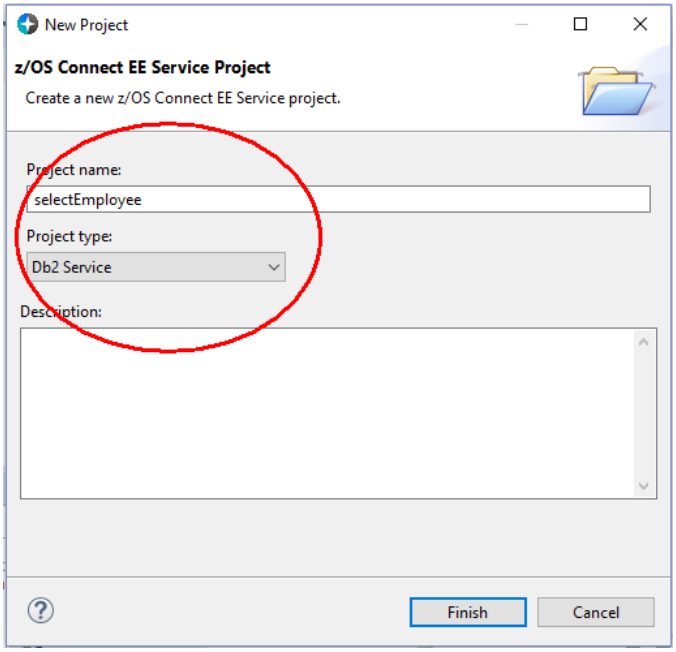


Mitch Johnson (mitchj@us.ibm.com)

___4. In the *New Project* window, scroll down and open the *z/OS Connect Enterprise Edition* folder and select *z/OS Connect EE Service Project* and then click the **Next** button.

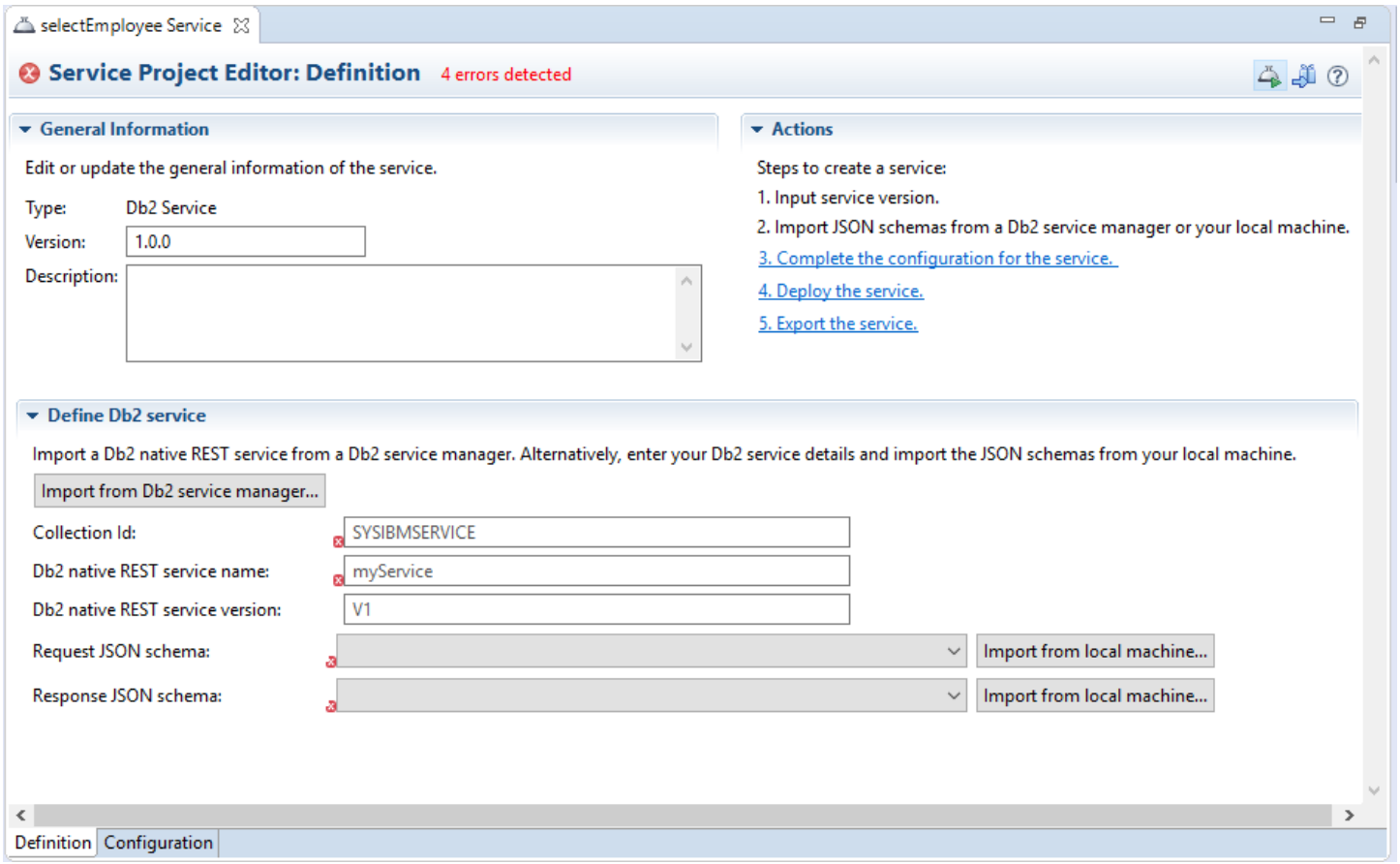


___5. On the new *New Project* window enter ***selectEmployee*** the *Project name* and use the pull-down arrow to select *Db2 Service* as the *Project type*. Click **Finish** to continue

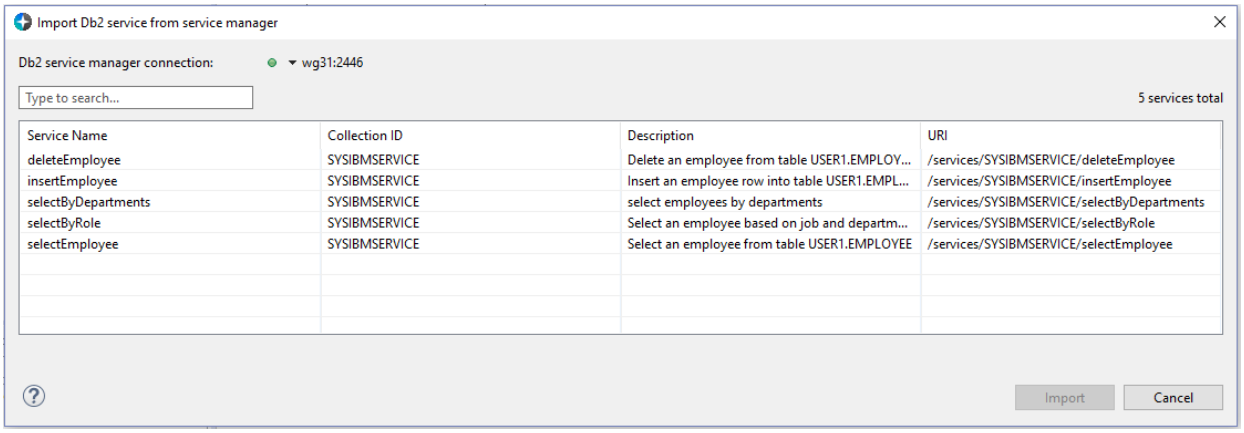


Mitch Johnson (mitchj@us.ibm.com)

___6. This will open the *Service Project Editor:Definition* window for the *selectEmployee* service. For now, disregard the message about the 4 errors detected, they will be addressed shortly.



___7. Next click the **Import from Db2 service manager** button. If the connection to the Db2 subsystem is not active (indicted by a red square rather than a green circle) use the pull-down arrow and select the *wg31:2446* connection.



___8. Select the *selectEmployee* service under *Service Name* and press the **Import** button to have the Db2 native REST service information retrieved from Db2 and stored in the local workspace.

Mitch Johnson (mitchj@us.ibm.com)

___9. This will return you back to the *Service Project Editor:Definition* view. All of the service required information (e.g., Collection Id, Db2 service name and version, and the layout of the JSON request and response schema) has been determined based on the information derived from the Db2 service manager. Next click on *3. Complete the configuration for the service* on right hand side of the view under *Actions*. This will switch the view to the view's *Configuration* tab.

___10. On the *Service Project Editor:Configuration* view enter the value ***Db2Conn*** in the area beside *Connection reference.* The connection reference binds this service to the *zosConnectionRestClientConnection* configuraton element with the same name (or ID) in the *server.xml* file (see below);

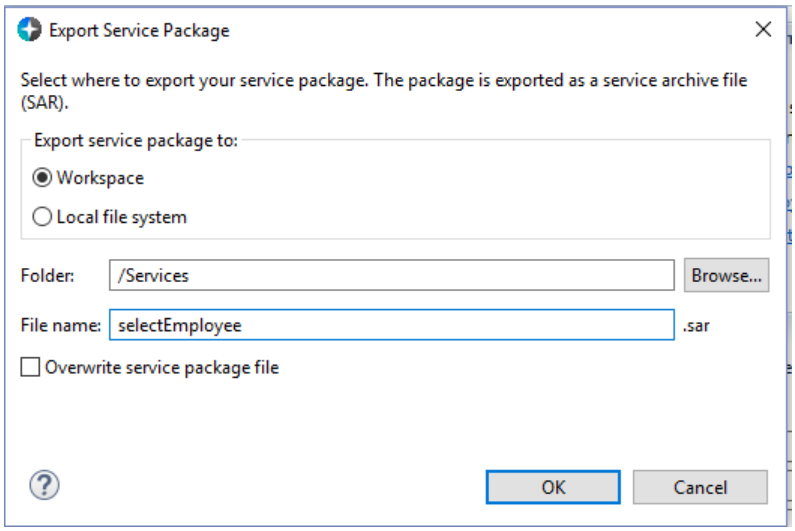
```
<zosconnect_zosConnectServiceRestClientConnection id="Db2Conn"
      host="wg31.washington.ibm.com"
      port="2446"
      basicAuthRef="dsn2Auth" />
```

___11. Switch back to the *Definition* view by clicking on the *Definition* tab at the bottom of the view.

___12. Save the changes made so far by using the key sequence **Ctrl-S** and close any open views.

___13. Next click on *4. Deploy the service* on right hand side of the view under *Actions*. This will open a *Deploy Service* window. On the *Deploy Service* window select the target server (wg31:9453) and click **OK** twice to have the service installed in the server.

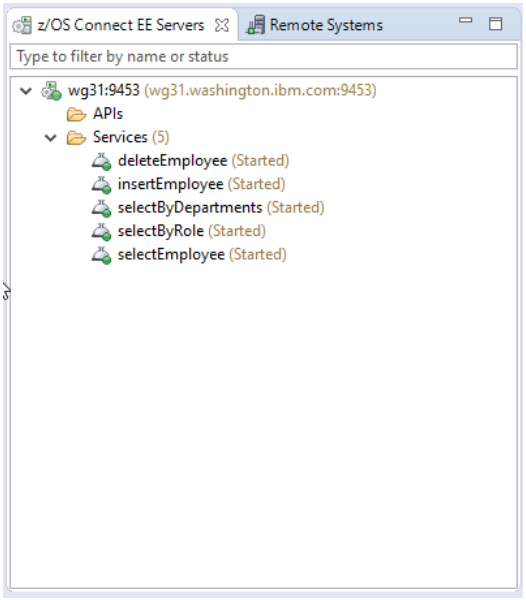
___14. Next click on *5. Export the service* on right hand side of the view under *Actions*. This will open a *Export Service* window. On the *Export Service* window select the radio button beside *Workspace* and use the **Browse** button to select the *Services* folder. Click **OK** to continue



Tech-Tip: To re-access the *Service Project Editor* just double click on the *service.properties* file in the Project Explorer view.

___15. Repeat steps 3 through 13 to create services for z/OS Connect services *selectByDepartments*, *insertEmployee* *deleteEmployee* and *selectByRole* where each z/OS Connect services invokes the corresponding Db2 native REST service.

___16. When finished the target server (*wg31:9453*) should have 5 services installed and started.

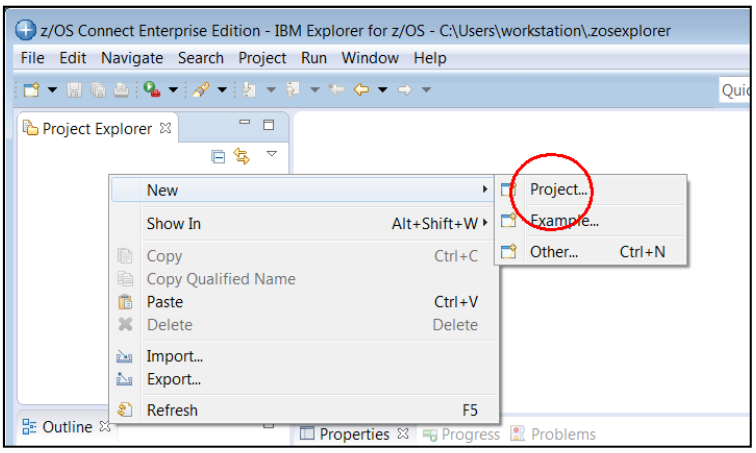


Summary

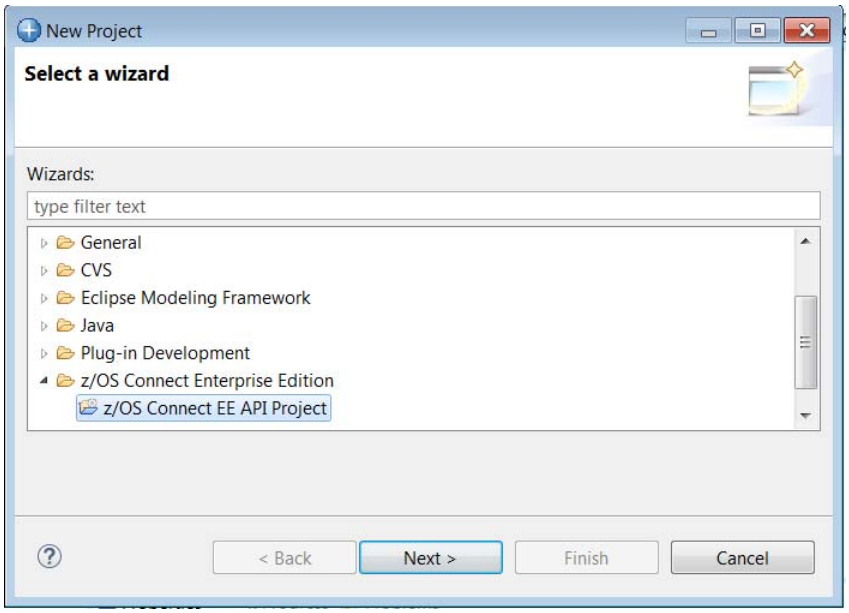
The JSON document generated by Db2 has been used to define the request and response JSON schema for each of the services as well as other configuration information. All of this information has been stored in a service archive file for use in accessing the Db2 native REST services.

Create the Db2 API Project

___1. In the *z/OS Connect Enterprise Edition* perspective of the z/OS Explorer create a new API project by clicking the right mouse button and selecting *New →Project*:

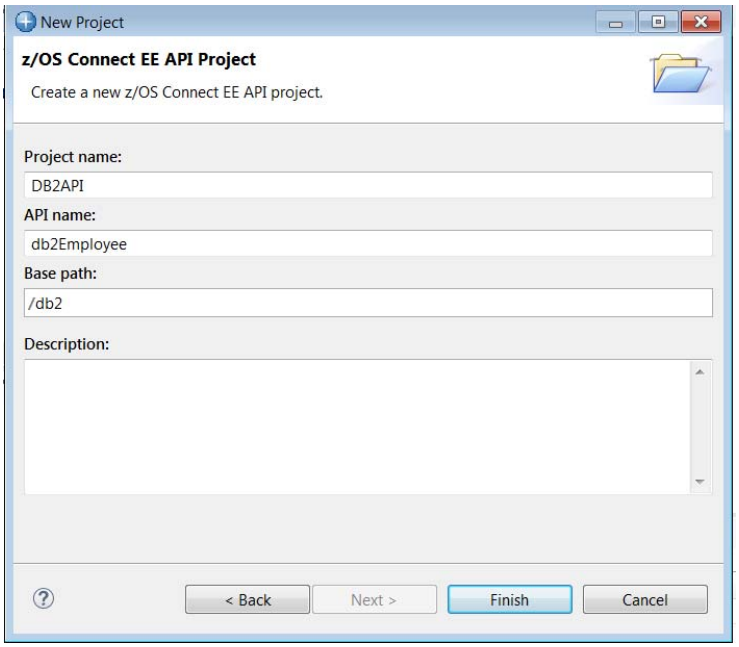


___2. In the *New Project* screen, scroll down and open the *z/OS Connect Enterprise Edition* folder and select *z/OS Connect EE API Project* and then click the **Next** button.



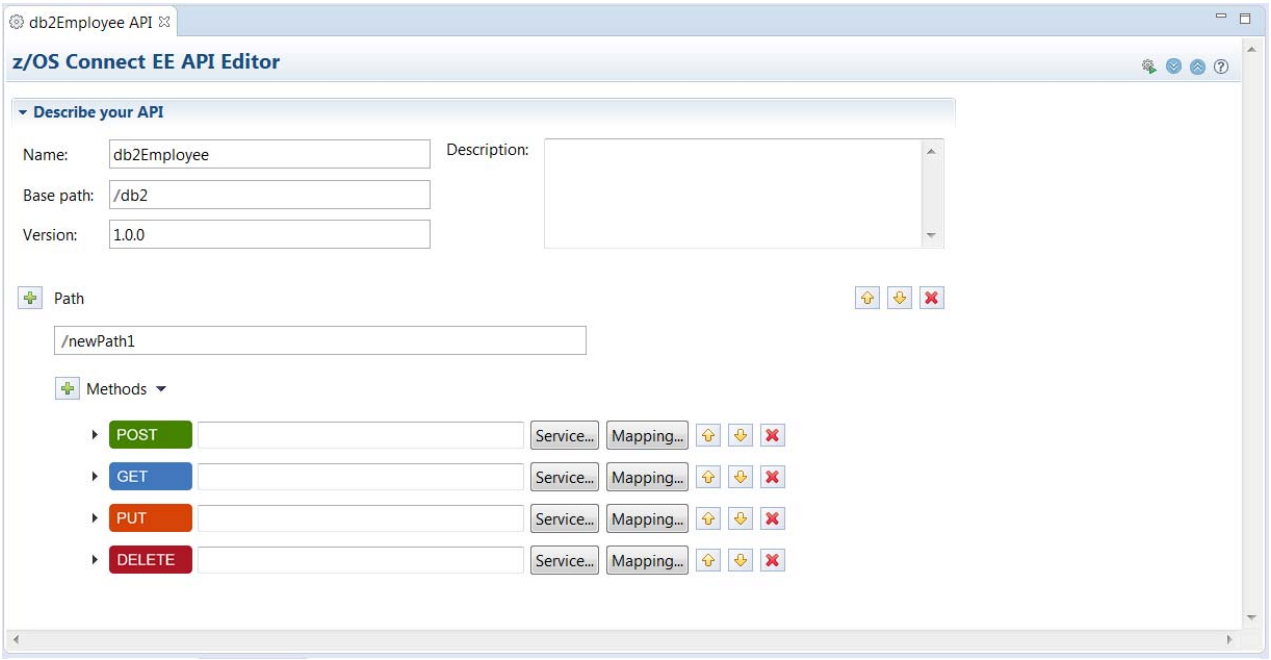
Mitch Johnson (mitchj@us.ibm.com)

___3. Enter ***DB2API*** for the *Project name.* Set the *API name* is set to ***db2Employee*** and the *Base path is* set to ***/db2.*** Click **Finish** to continue.



Important: The values are somewhat arbitrary, but they do relate to later tasks. If you use the values and cases as supplied, then the subsequent commands and the use of subsequent URLs will work seamlessly.

___4. You should now see something like the view below. The view may need to be adjusted by dragging the view boundary lines.

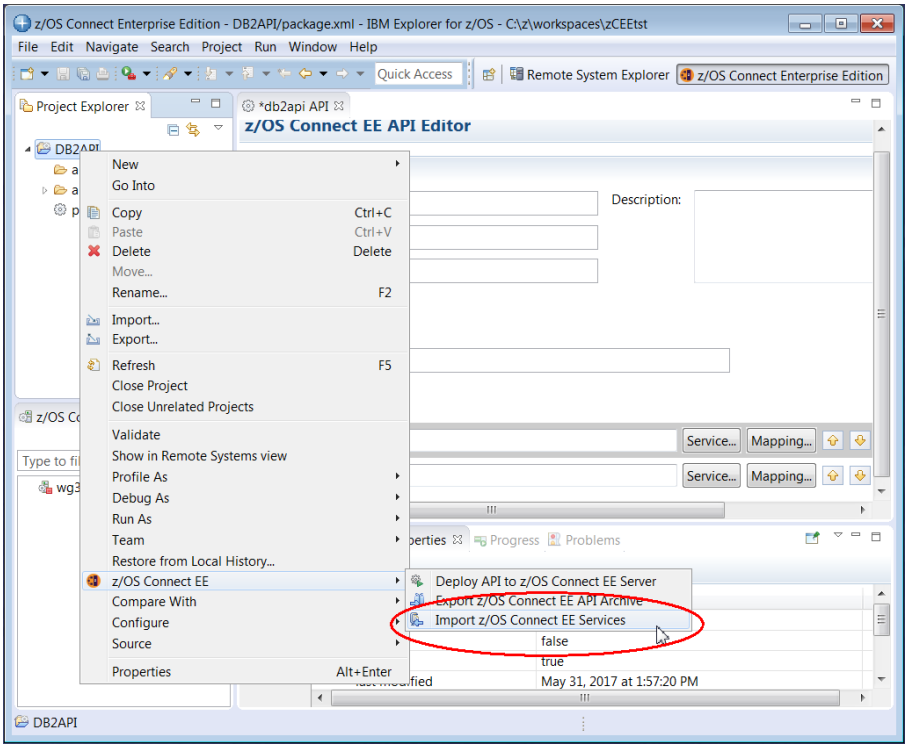


Summary

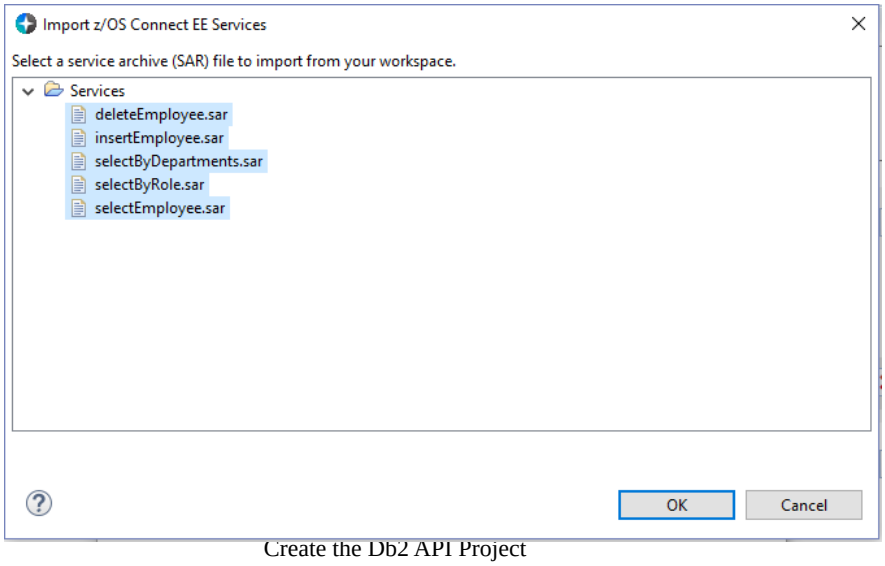
This created the basic framework for the API project in the API editor

Import the SAR files

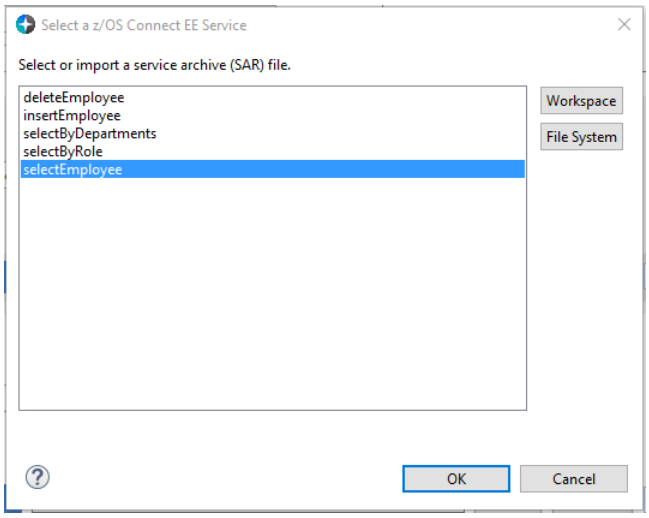
___1. In the z/OS Explorer in the *z/OS Connect Enterprise Edition* perspective in the the *Project Explorer* view (upper left), right-click on the *DB2API* project, then select *z/OS Connect EE* and then *Import z/OS Connect EE Services* (see below):



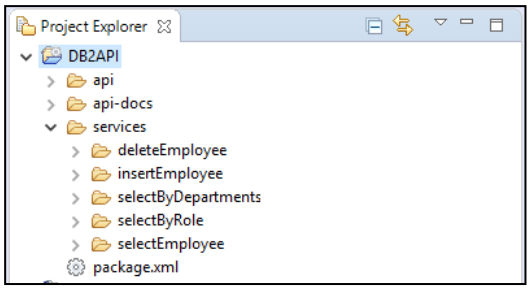
___2. In the *Import z/OS Connect EE Services* window click on the **Workspace** button and expand the *Services* folder. Select the 5 SAR files and click on the **OK** button twice.



___3. The five service archive files should appear in the *Import z/OS Connect Services* screen. Click the **OK** button to import them into the workspace.



___4. In the *Project Explorer* view (upper left), expand the *services* folder to see the the imported service:



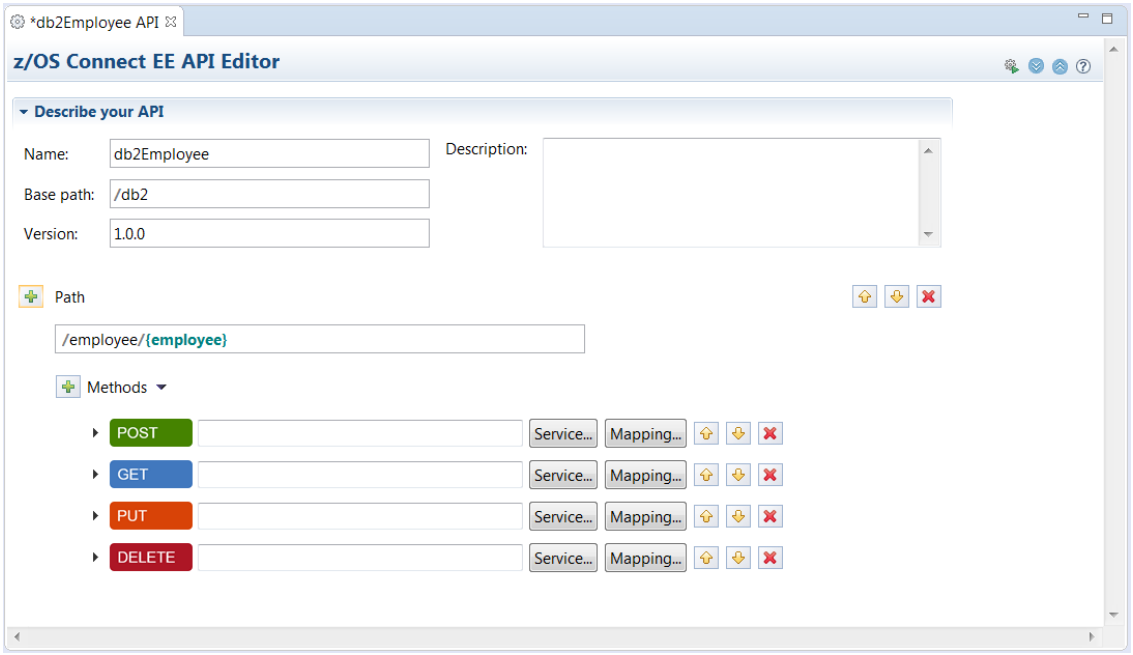
Tech-Tip: To re-access the *API Editor* just double click on the *package.xml* file in the Project Explorer view.

Summary

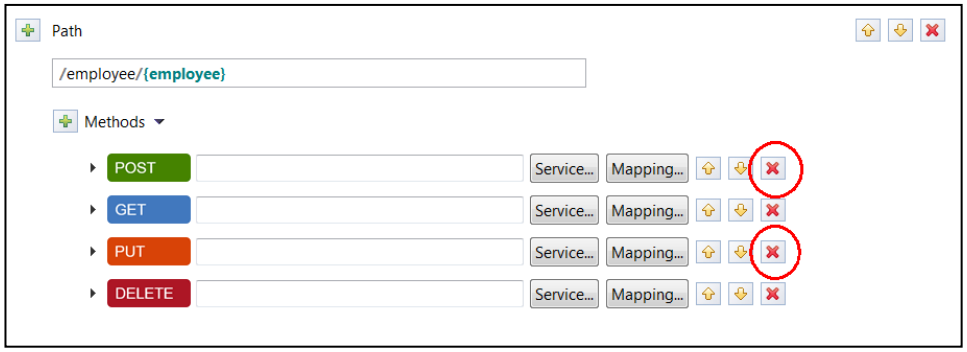
The SAR file created earlier have been imported into the API editor. That provides the editor with information about the underlying services and the JSON schemas.

Compose an API for Db2 native REST Services

___1. Start by entering a *Path* of ***/employee/{employee}*** in the *z/OS Connect EE API Editor* view as shown below:



___2. For the *Db2 API* when the *employee* path parameter is present the supported HTTP methods will be **GET** and **DELETE**. Remove the **POST** and **PUT** methods by clicking the *X* icon to the right of each method.



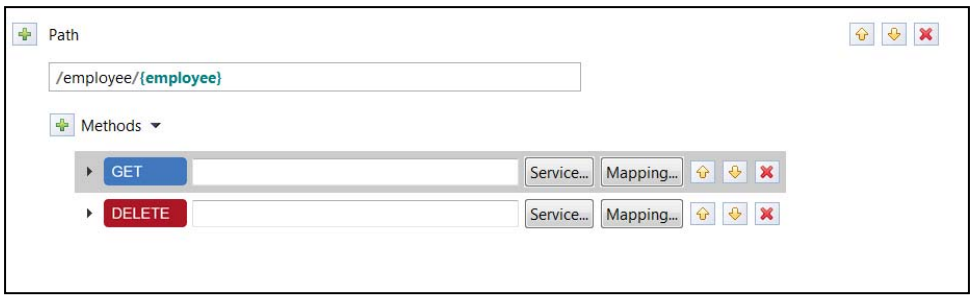
Note: The */employee* path element again is somewhat arbitrary, but is used to distinguish this request from other requests that may be configured in the same API.

The *{employee}* element is a path parameter in the URL that will be used to provide the key of the record for get and delete REST requests.

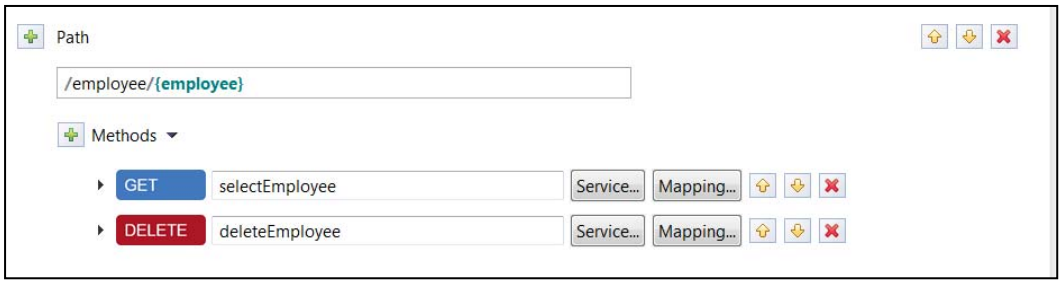
The full URL to invoke the methods for this particular path will be
https://hostname:port/db2/employee/######
where ###### is the employee record of a row in USER1.EMPLOYEE

Mitch Johnson (mitchj@us.ibm.com)

That should leave you with the **GET** and **DELETE** methods.



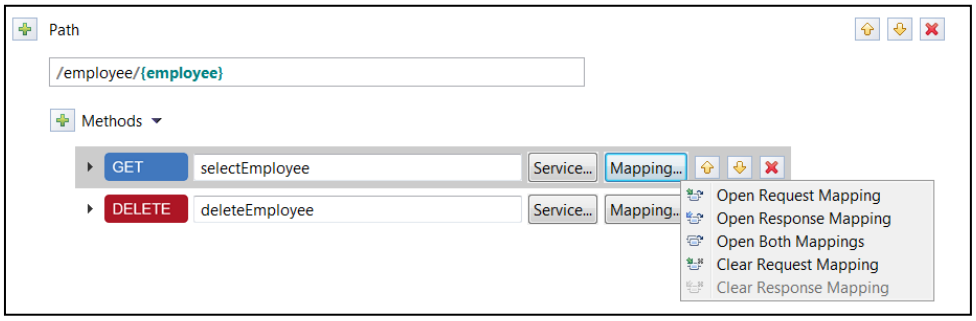
___3. Click on the **Service** button to the right of the **GET** method. Then select the *selectEmployee* service from the list of service archive files and click **OK**. This will populate the field to the right of the method. Repeat this for the **DELETE** methods selecting the *deleteEmployee* service.



___4. Save the changes made so far by using the key sequence **Ctrl-S.**

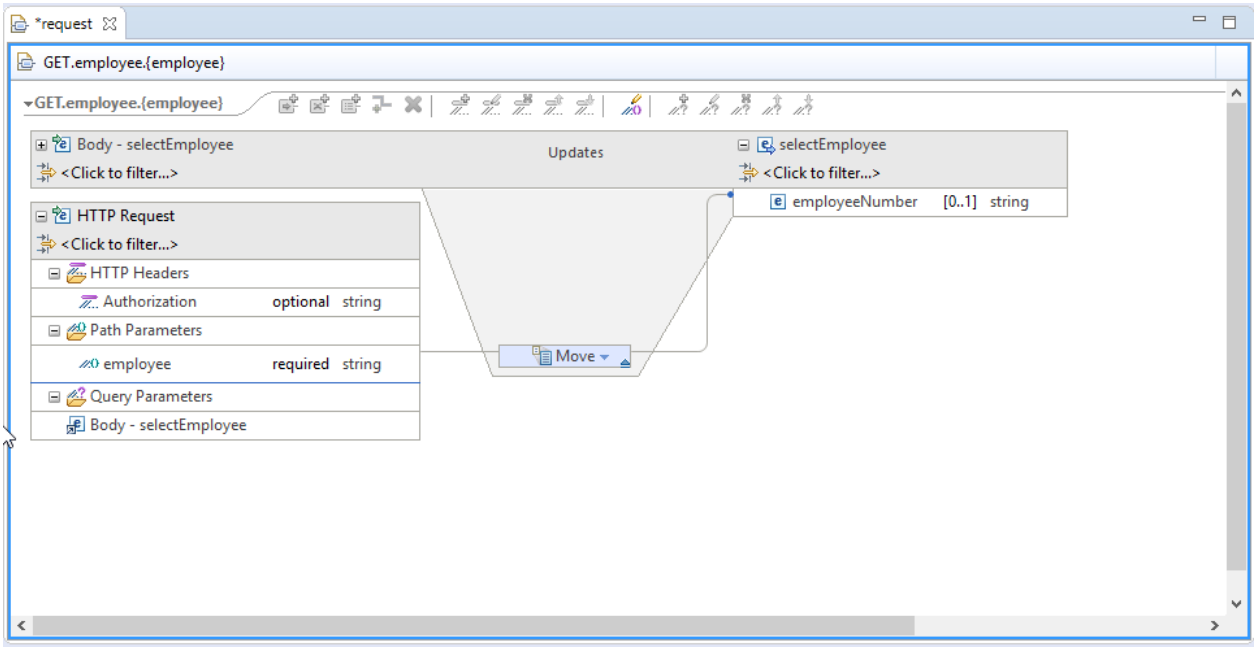
> **Tech-Tip:** If any change is made in any edit view an asterisk (*) will appear before the name of the artifact in the view tab, e.g. **package.xml*. Changes can be saved at any time by using the **Ctrl-S** key sequence.

___5. Next, click on the **Mapping** button beside the **GET** method and then select *Open Request Mapping*:



Mitch Johnson (mitchj@us.ibm.com)

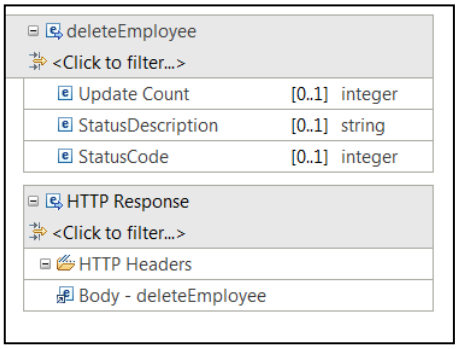
___6. In the mapping view that opens, use your mouse to select the *employee* field under *Path Parameter* and drag it over to the *employeeNumber* field on the right hand side.The result is a line that maps a move of the value of *employee* from the URL to the field *employeeNumber.* This means the value of *{employee}* parameter specified in a URL will be moved to the *employeeNumber* field.



___7. Use the ***Ctrl-S*** key sequence to save all changes and close the *GET.employee.{employee}* view.

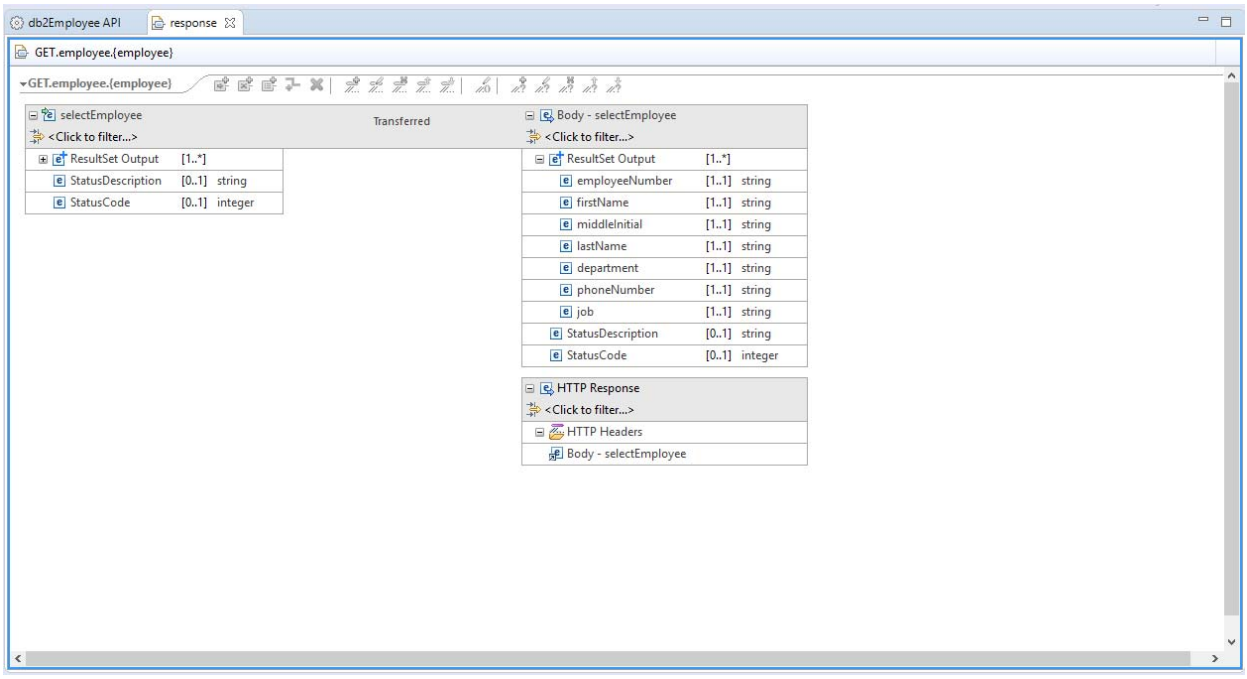
___8. This same pattern used Steps 5 through 7 are repeated to configure the request mapping for the **DELETE** method.

___9. No Response mappings for the **DELETE** methods is required. If you open the response mapping for the DELETE method, you will see no fields from the deleted row are returned in the response.



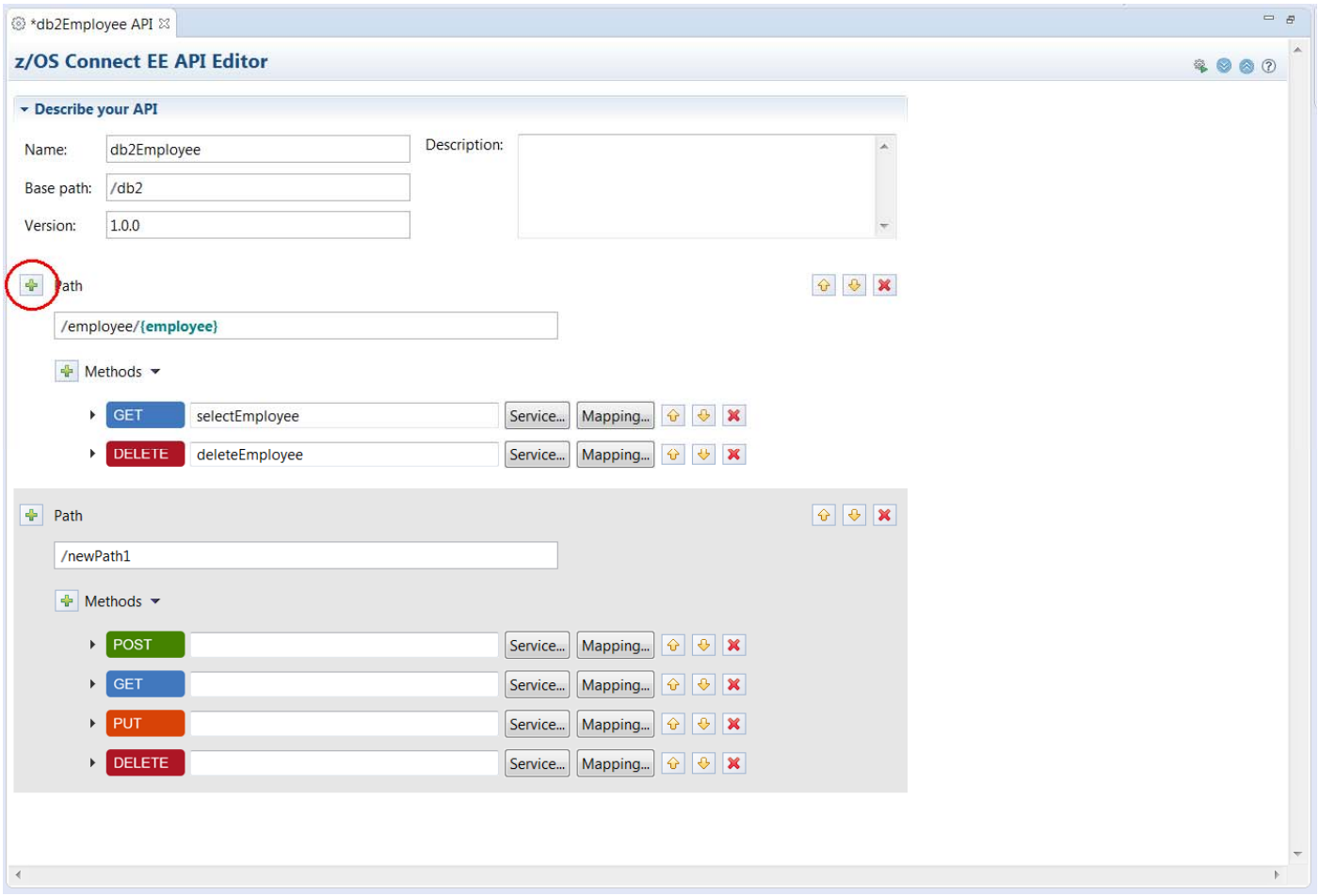
___10. For the **GET** method the default response mapping will return the columns names exposed in the native Db2 REST service to the REST client. Click the **Mapping** button beside the **GET** method and select the *Open Default Response Mapping* option.

___11. Use the slider bar to fully expose the *ResultSet Output* structure. You will see the columns names as provided when the native Db2 REST service was created in the *AS* clauses.



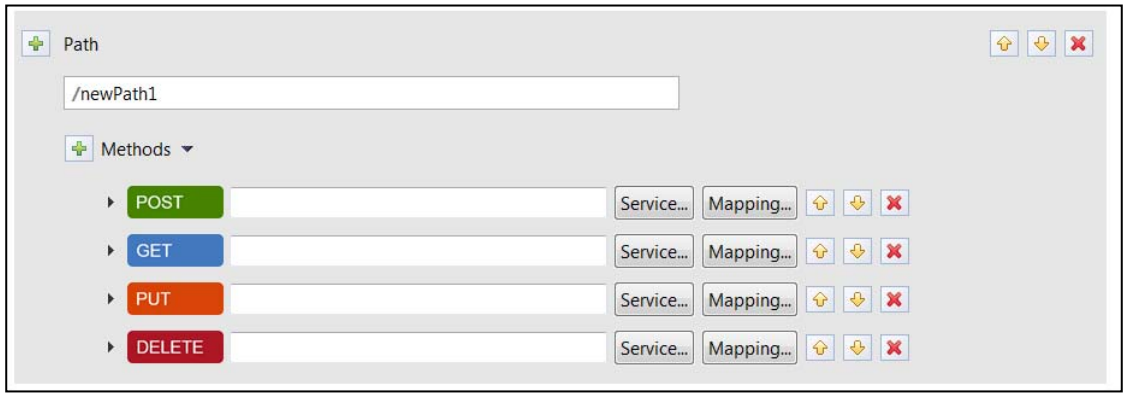
___12. Save all changes with the **Ctrl-S** key sequence and close the response view.

___13. Next, we want to add a Path for a **GET** method for the *selectByRole* service. The values for the *JOB* and *WORKDEPT* columns are to be included in the URL so no JSON request message is required. Click the plus icon beside Path on the z/OS Connect EE API Editor view to add another path to the API.

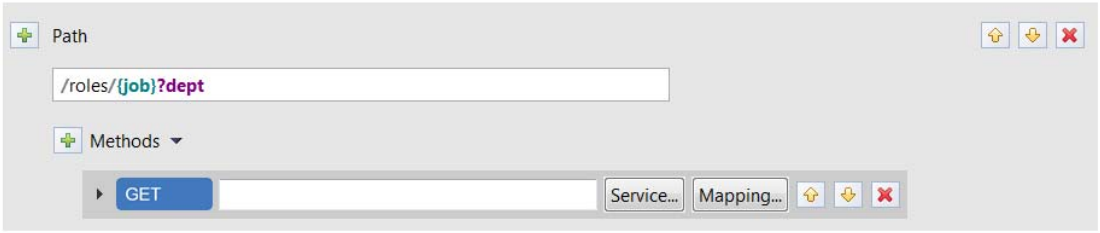


Mitch Johnson (mitchj@us.ibm.com)

The result is another full set of methods for the new *PATH*.



___14. Enter a path value of ***/roles/{job}?dept*** and remove the **POST**, **PUT** and **DELETE** methods.



Tech-Tip: Additional *Paths* can be added by clicking the + icon beside *Path and a*dditional *Methods* can be added by clicking the + icon beside *Methods*.

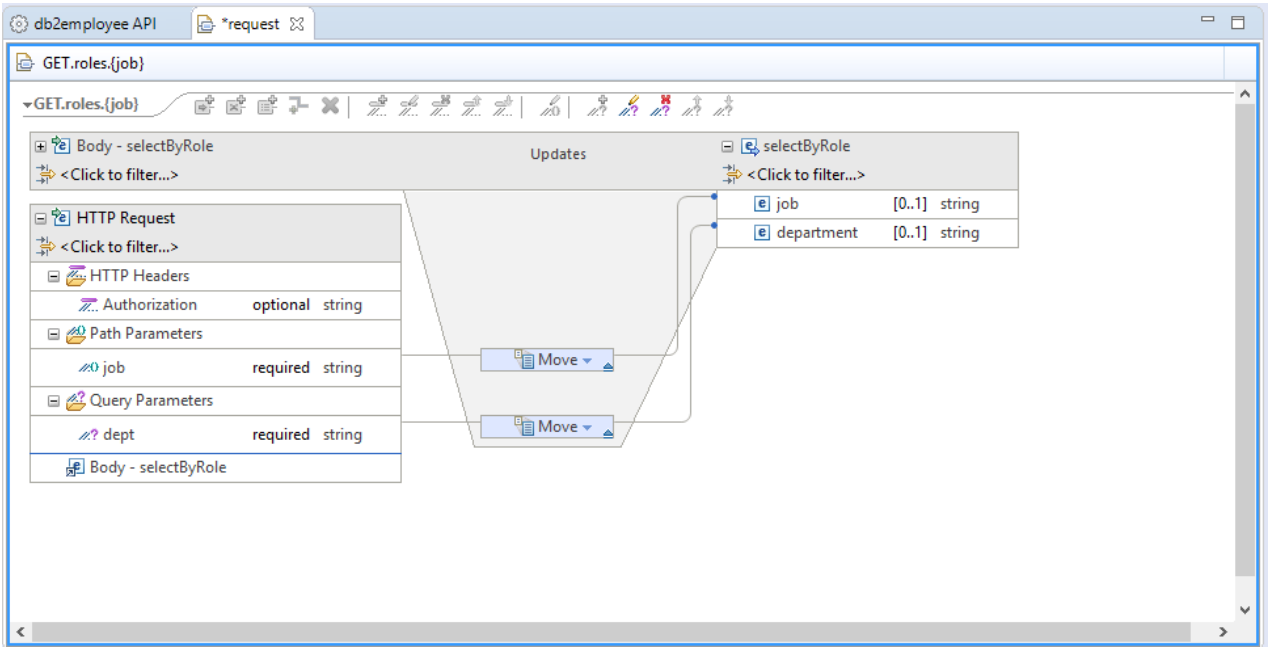
___16. Click the **Service** button beside **GET** and select the *selectByRole* service:

___17. Save the changes by using the key sequence **Ctrl-S**.

___18. Click on *Mapping* → *Open request mapping*.

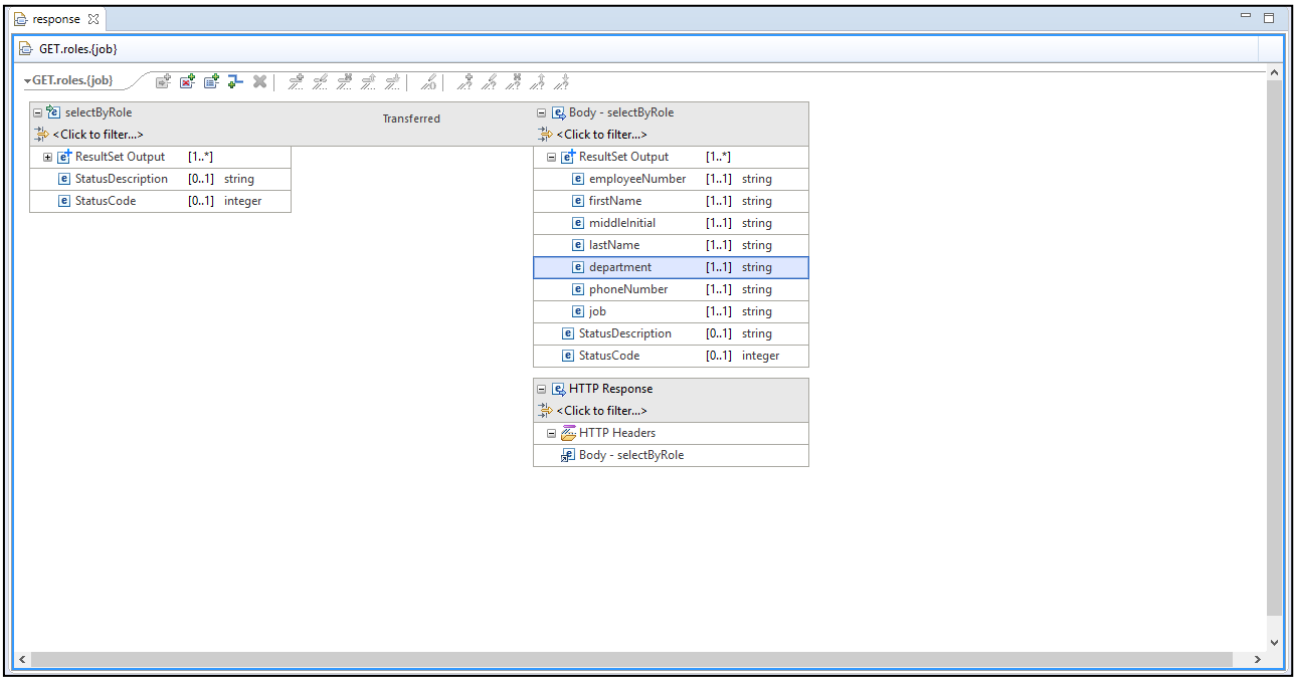
Mitch Johnson (mitchj@us.ibm.com)

___19. Use the left mouse button to drag the *job Path Parameters* from the left-hand side to the *job* field on the right side. Use the left mouse button to drag the *dept Query Parameter* from the left-hand side to the *departmentT* field on the right-hand side.

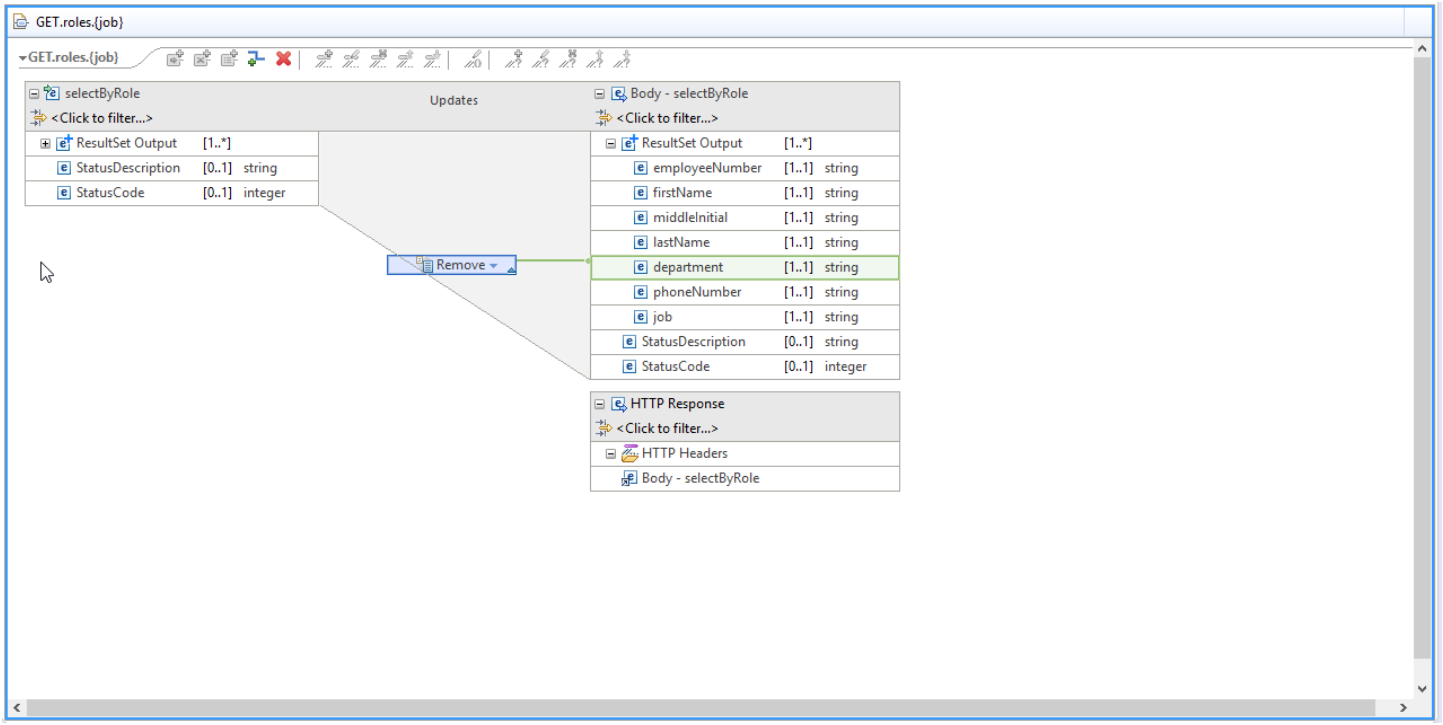


___20. For the **GET** method the default response mapping will return all columns to the REST client. Some columns, like *BIRTHDATE, SALAY, BONUS* and *COMM* were not exposed. Back on the *z/OS Connect EE API Editor view* click the **Mapping** button beside the **GET** method and select the *Open Default Response Mapping* option.

___21. Fully expose the *ResultSet Output* structure. Use the left mouse button and draw a dotted line box that fully encloses the field. *department.* Released the mouse button and this field should be selected (the background should be blue).



___22. Right mouse button click on the field and select the *Add Remove transform* option to remove the selected fields from the response. The REST client will not see *department* field in the response from a **GET** request.



___23. Save all changes with the **Ctrl-S** key sequence and close the response view.

___24. Next, we want to add a Path for a **GET** method for the *selectByDepartment* service. Click the plus icon beside Path on the z/OS Connect EE API Editor view to add another path to the API. Enter ***/department*** for the value of *Path* and delete the *POST, PUT* and *DELETE* methods.

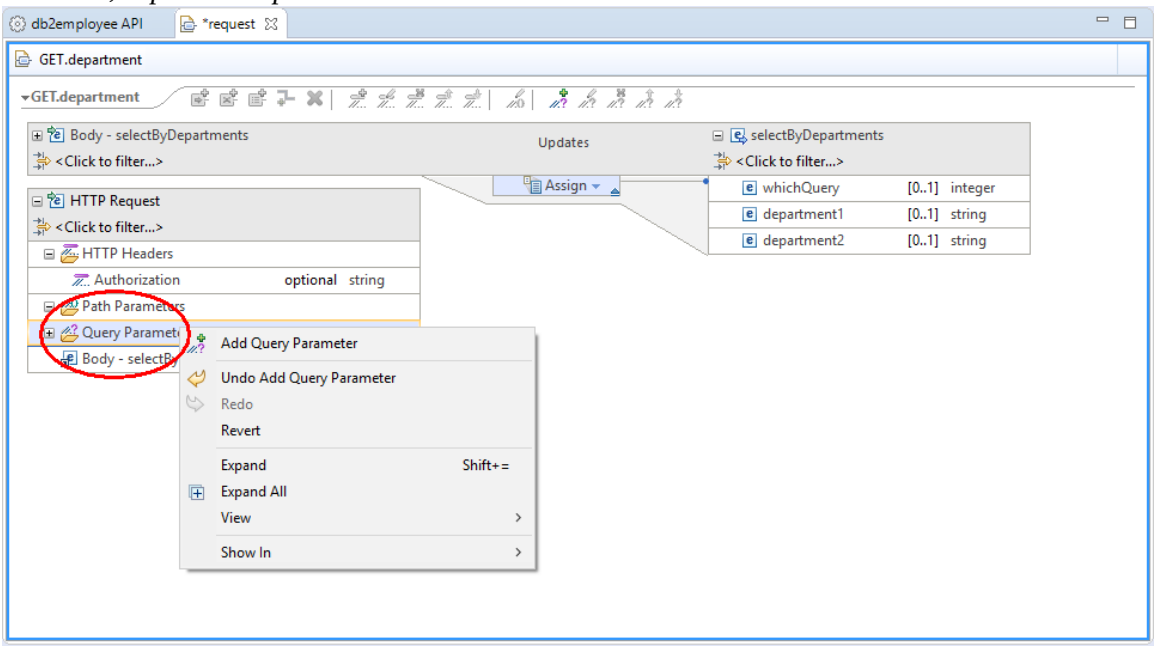
___25. Click on the **Service** button to the right of the **GET** method. Then select the *selectByDepartments* service from the list of service archive files and click **OK**. This will populate the field to the right of the method.

___26. Save the changes so far by using the key sequence **Ctrl-S.**

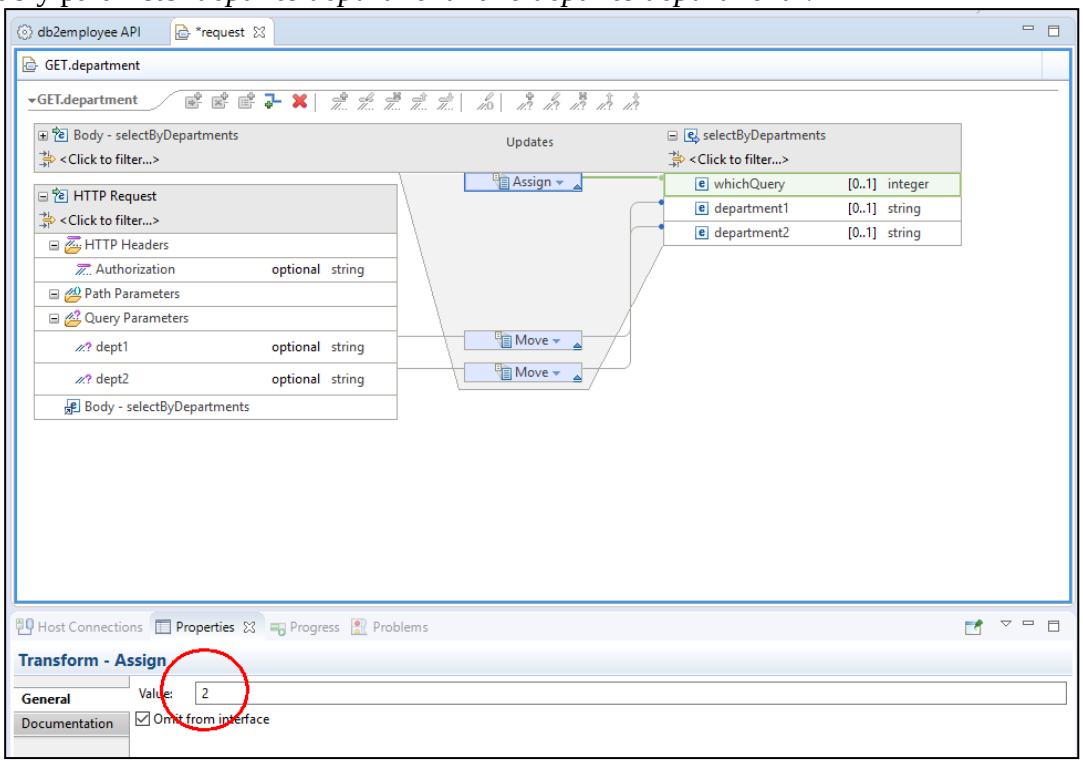
___27. Next, click on the **Mapping** button beside the **GET** method and then select *Open Request Mapping*.

___28. Select the *WHICHQUERY* field and right button click and select the *Add Assign transform* option. This action opens a *Properties* tab in the lower view. In this tab, a value can be entered which will be used to populate this field when a **GET** method is invoked. Enter ***2*** in the area beside *Value* in the *Properties* tab.

___29. Select *Query Parameters* on the left-hand side and right mouse button click and select *Add Query Parameter.* Add two query parameters, *dept1* and *dept2*.



___30. Map query parameter *dept1* to *department1* and *dept2* to *department2*.

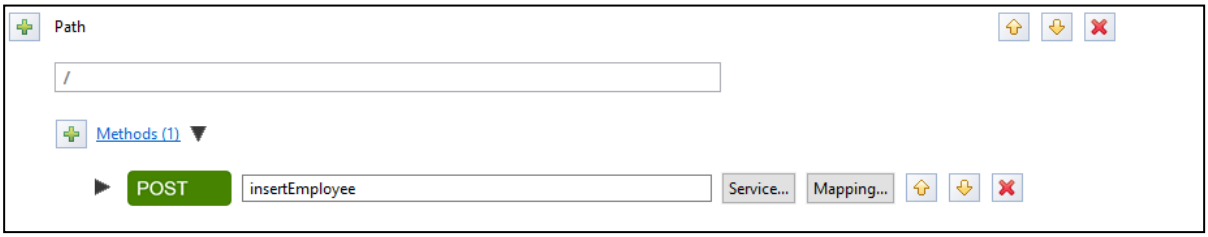


Mitch Johnson (mitchj@us.ibm.com)

___31. Close and save all open *request* or *response* mapping tabs.

___32. Next, we want to add a Path for a **POST** method for the *insertEmployet* service. Click the plus icon beside Path on the z/OS Connect EE API Editor view to add another path to the API. Enter / (just a slash) for the value of *Path* and delete the *GET, PUT* and *DELETE* methods.

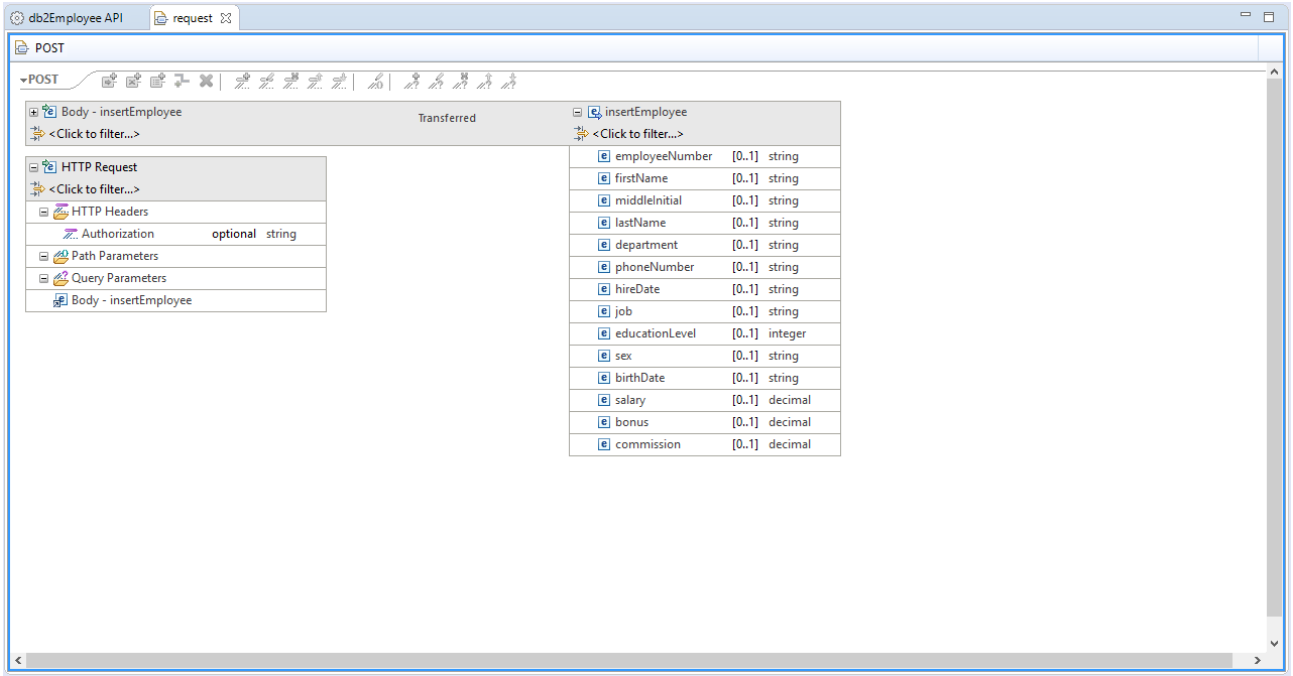
___33. Click on the **Service** button to the right of the **GET** method. Then select the *insertEmployee* service from the list of service archive files and click **OK**. This will populate the field to the right of the method.



___34. Save the changes so far by using the key sequence **Ctrl-S.**

___35. Next, click on the **Mapping** button beside the **POST** method and then select *Open Request Mapping*.

___36. Note that the fields identified by the DBA are the property names in the request message. No mapping needs to be done.



___37. Close and save all open *request* or *response* mapping tabs.

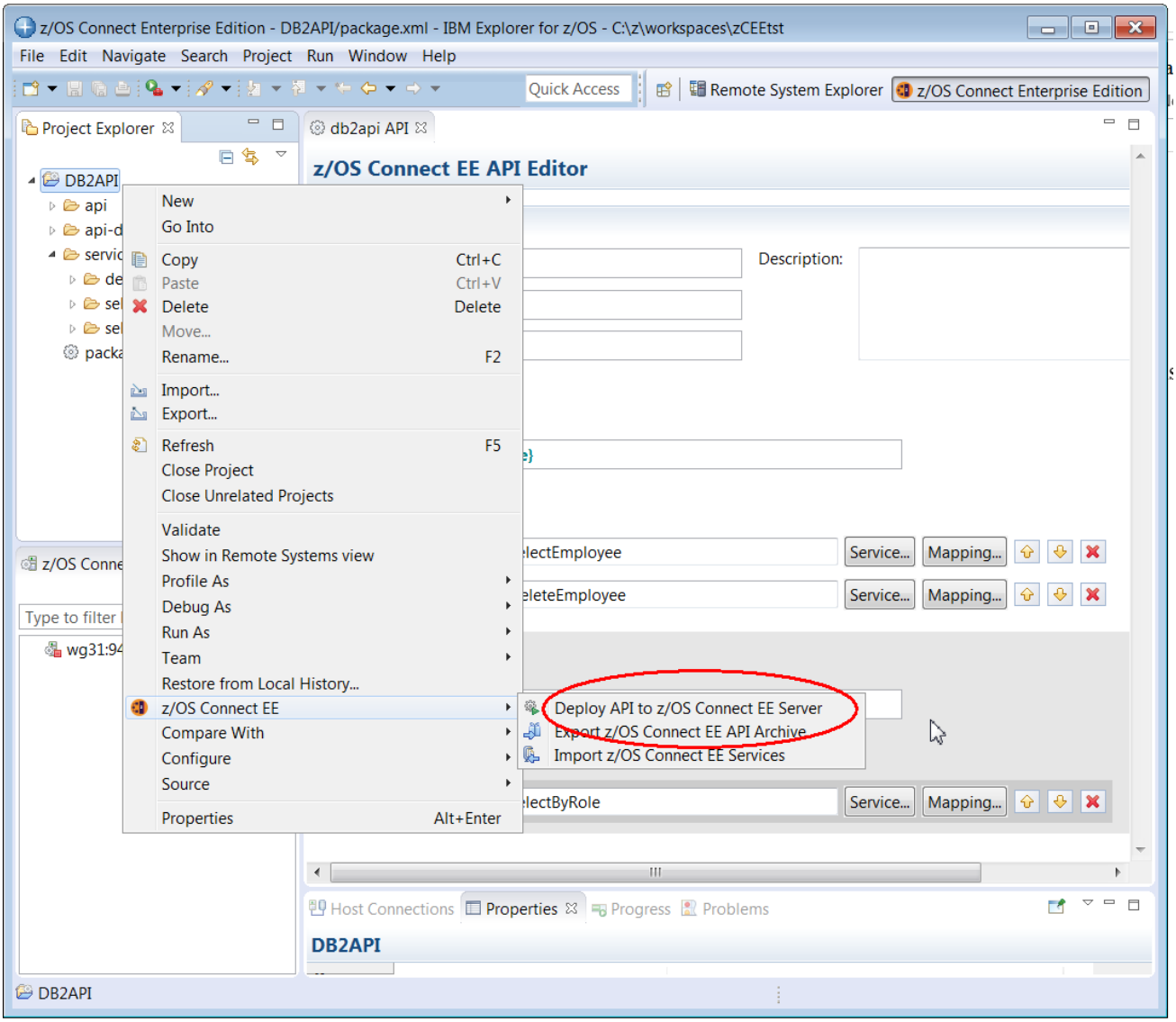
Mitch Johnson (mitchj@us.ibm.com)

Summary

You created the API, which consists of two paths and the request and response mapping associated with each. That API will now be deployed into z/OS Connect EE V3.0.

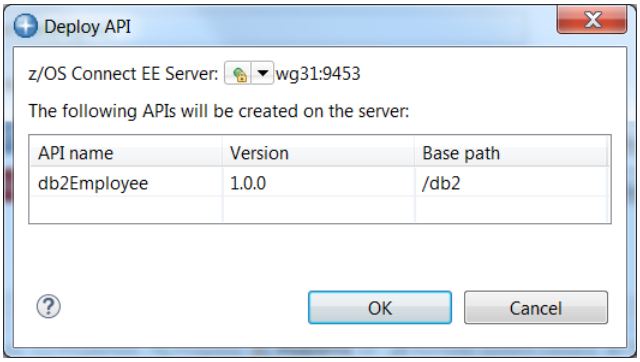
Deploy the API to a z/OS Connect EE Server

___1. In the *Project Explorer* view (upper left), right-mouse click on the *DB2API* folder, then select *z/OS Connect EE→ Deploy API to z/OS Connect EE Server*.

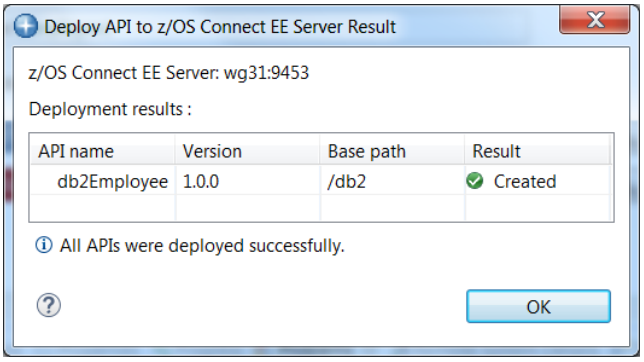


Mitch Johnson (mitchj@us.ibm.com)

___2. If the z/OS Explorer is connected to only one z/OS Connect server there is only one choice (*wg31:9453*). If z/OS Explorer had multiple connections to z/OS Connect servers then the pull-down arrow would allow a selection to which server to deploy, select *wg31:9453* from the list. Click **OK** on this screen to continue.



___3. The API artifacts will be transferred to z/OS in an API archive (AAR) file and copied into the */var/ats//zosconnect/servers/se/resources/zosconnect/apis* directory.

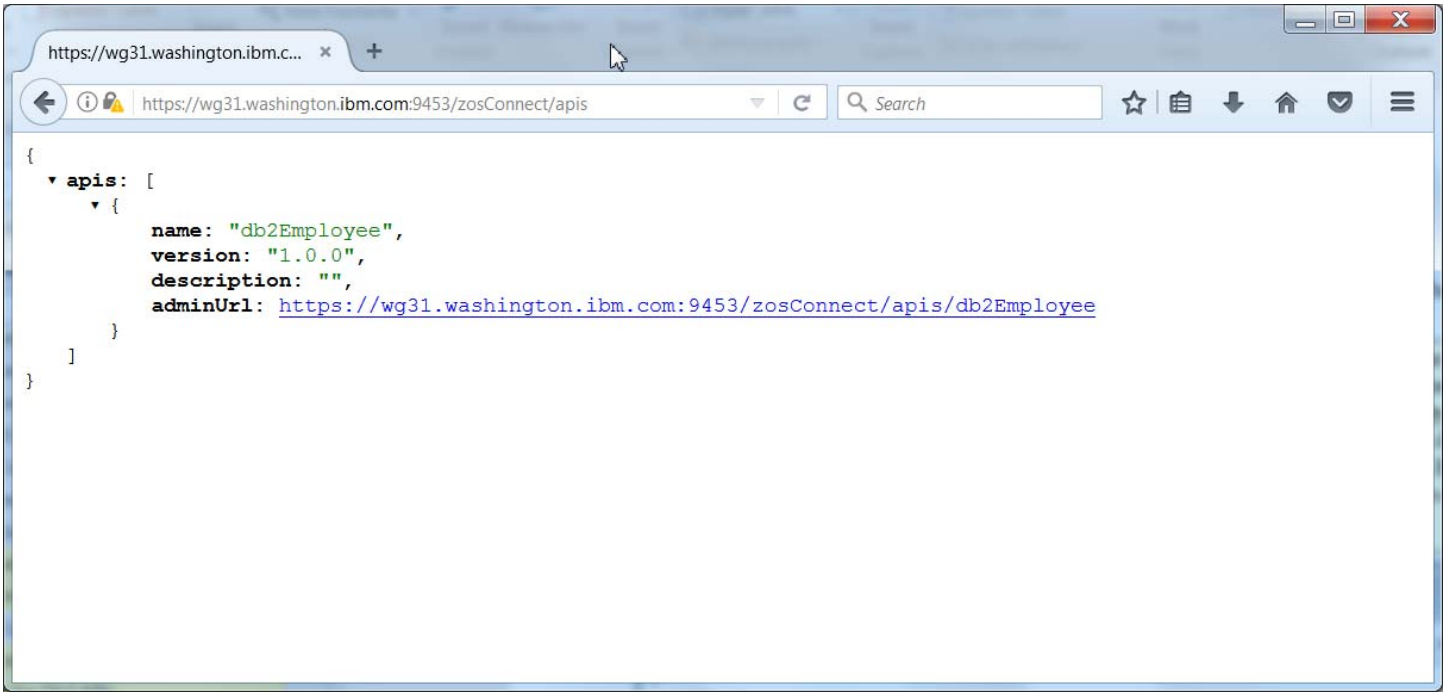


Mitch Johnson (mitchj@us.ibm.com)

Test the Db2 APIs

___1. Next enter URL ***https://wg31.washington.ibm.com:9453/zosConnect/apis*** in the Firefox browser and you should see the window below. The *Db2api* API now shows as being available.

> **Tech Tip:** You may be challenged by Firefox because the digital certificate used by the Liberty z/OS server is self-signed Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and password. If you do see the prompt, enter the username ***Fred*** and password **fredpwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.

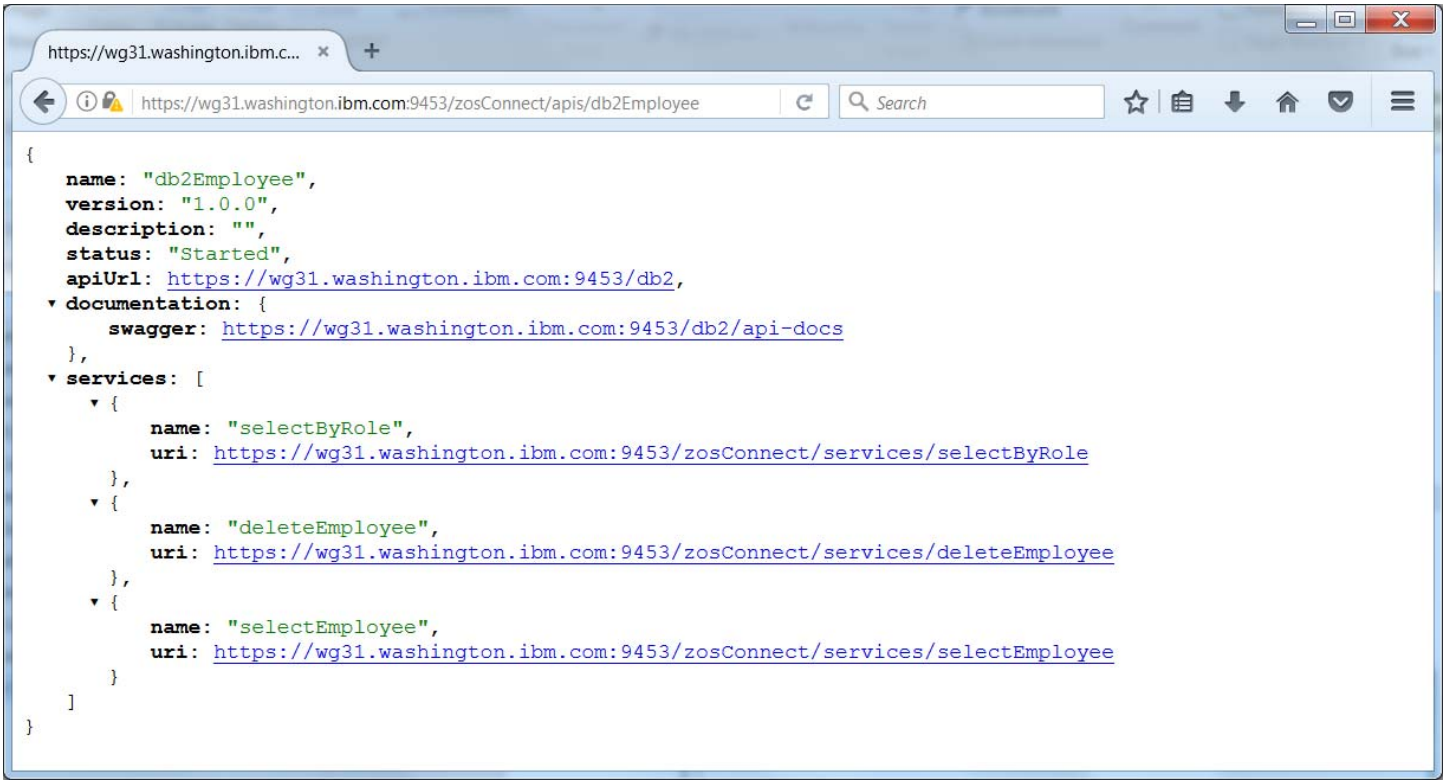


> **Tech-Tip:** It is very important to access the z/OS Connect server from a browser prior to any testing using the Swagger UI or Firefox RESTClient extension. Accessing a z/OS Connect URL from a browser starts an SSL handshake between the browser and the server. If this handshake has not performed prior to performing any test the test will fail with no message in the browser and no explanation. Ensuring this handshake has been performed is why you may be directed to access a z/OS Connect URL prior to using the Swagger UI or RESTClient extension during this exercise.

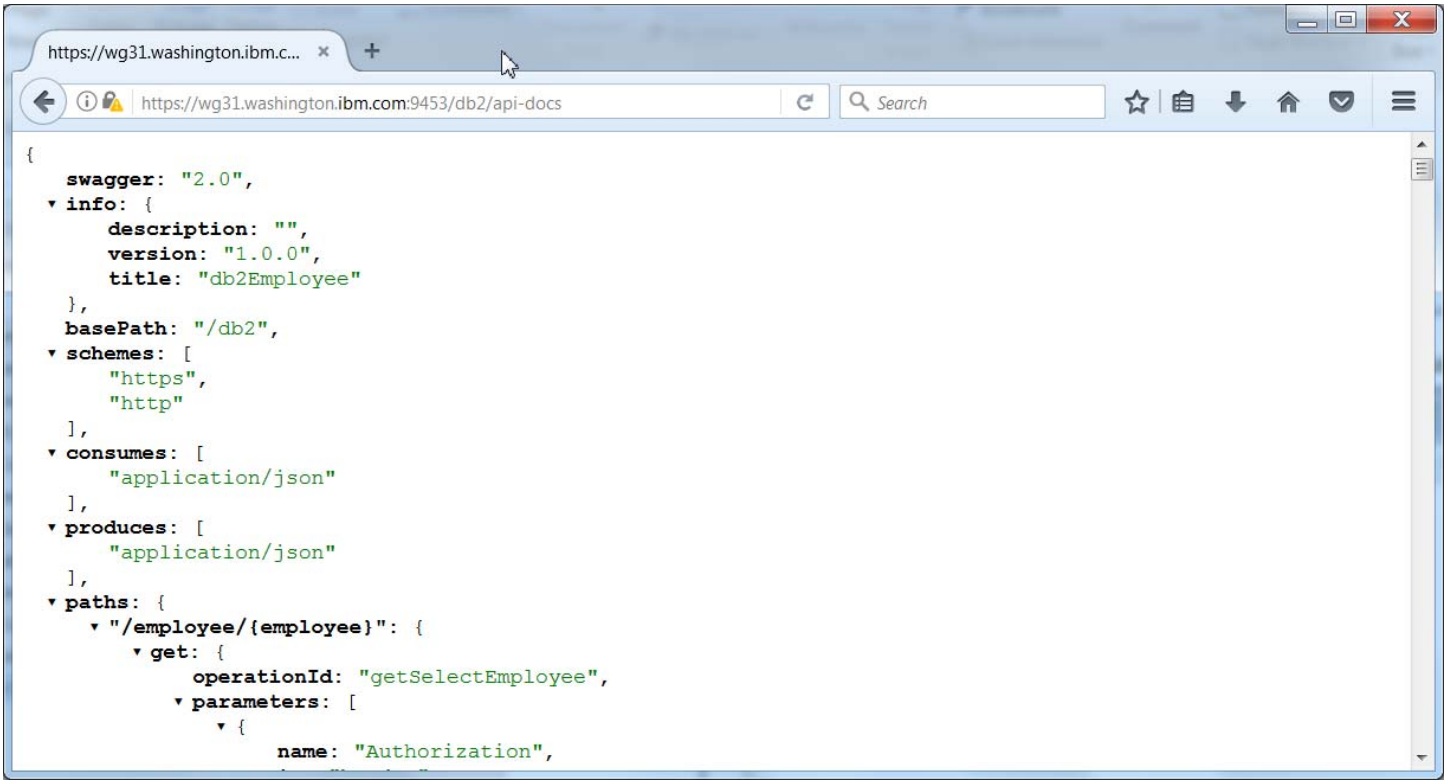
Mitch Johnson (mitchj@us.ibm.com)

Note that other APIs may also be displayed.

___2. If you click on *adminUrl* URL the window below should be displayed:

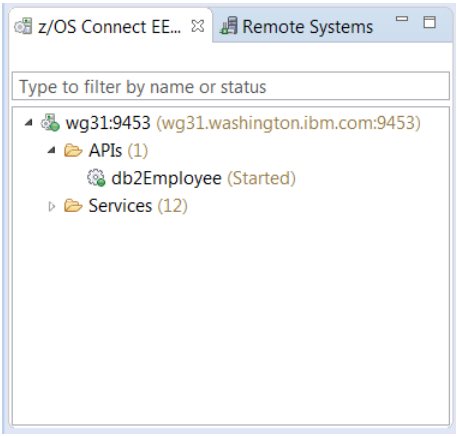


___3. Finally click on the *swagger* URL and you should see the Swagger document associated with this API.

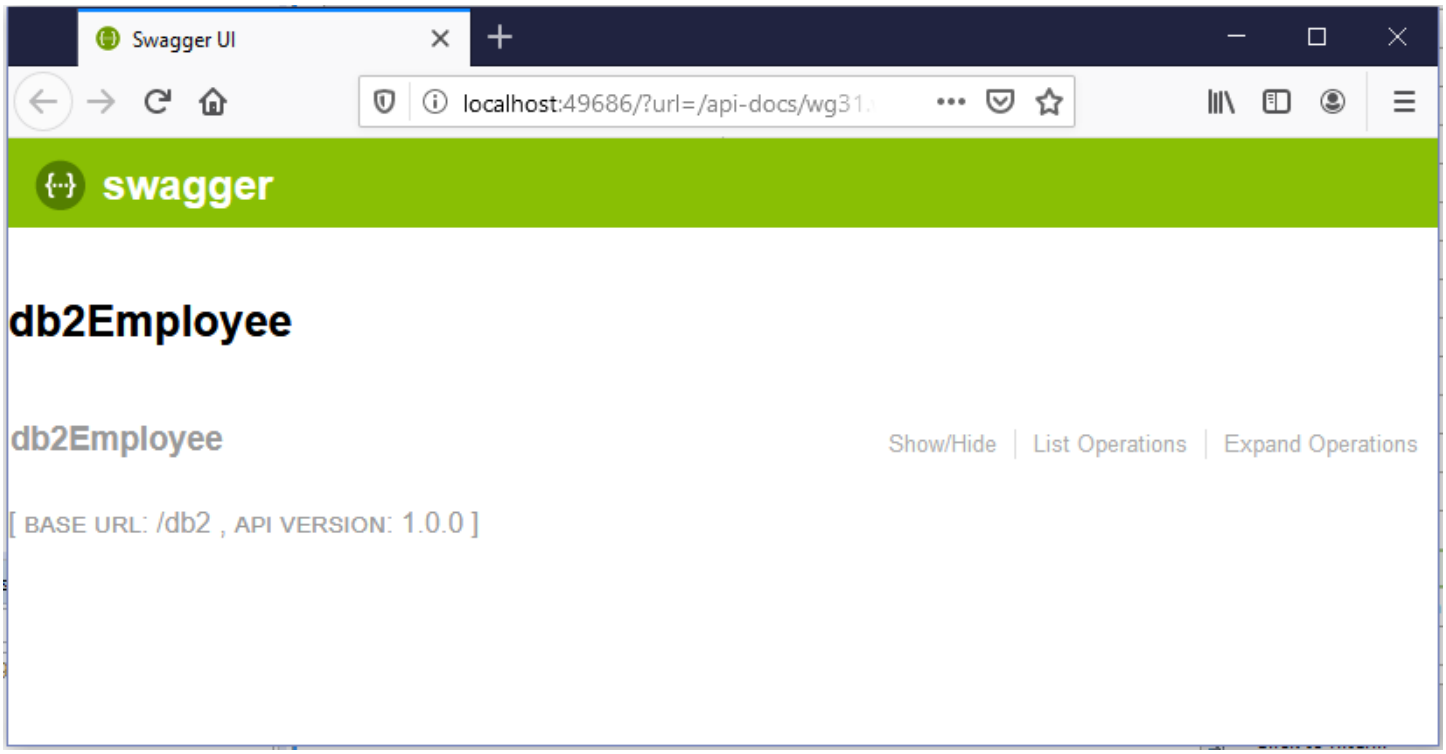


Explore this Swagger document and you will see the results of the request and response mapping performed earlier. This Swagger document can be used by a developer or other tooling to develop REST clients for this specific API.

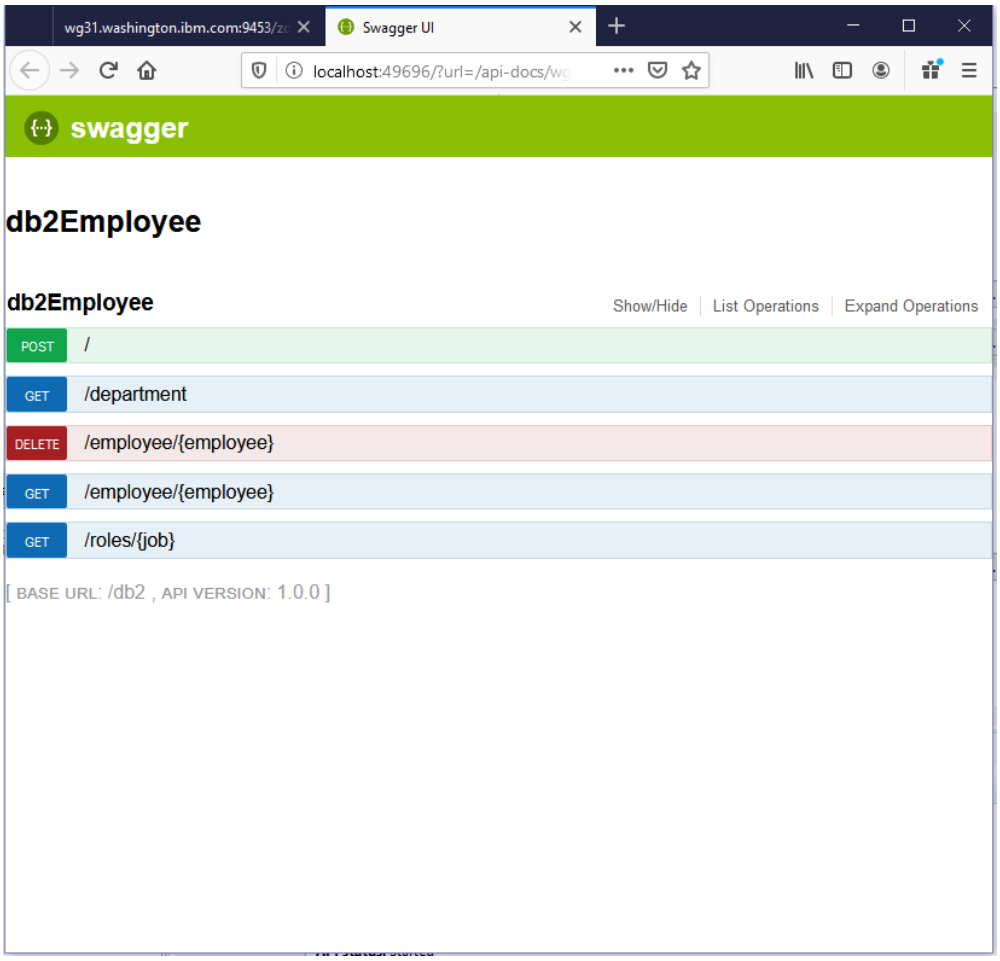
___4. In the lower left-hand side of the *z/OS Connect Explorer* perspective there is view entitled *z/OS Connect EE Servers*. Expand *wg31:9453* and the expand the *APIs* folder. the RESTClient icon. You should see a list of the APIs installed in the server.



___5. Right mouse button click on *db2Employee* and select *Open in Swagger UI.* Click OK if an informational prompt appears. This will open a new view showing a *Swagger* test client (see below).

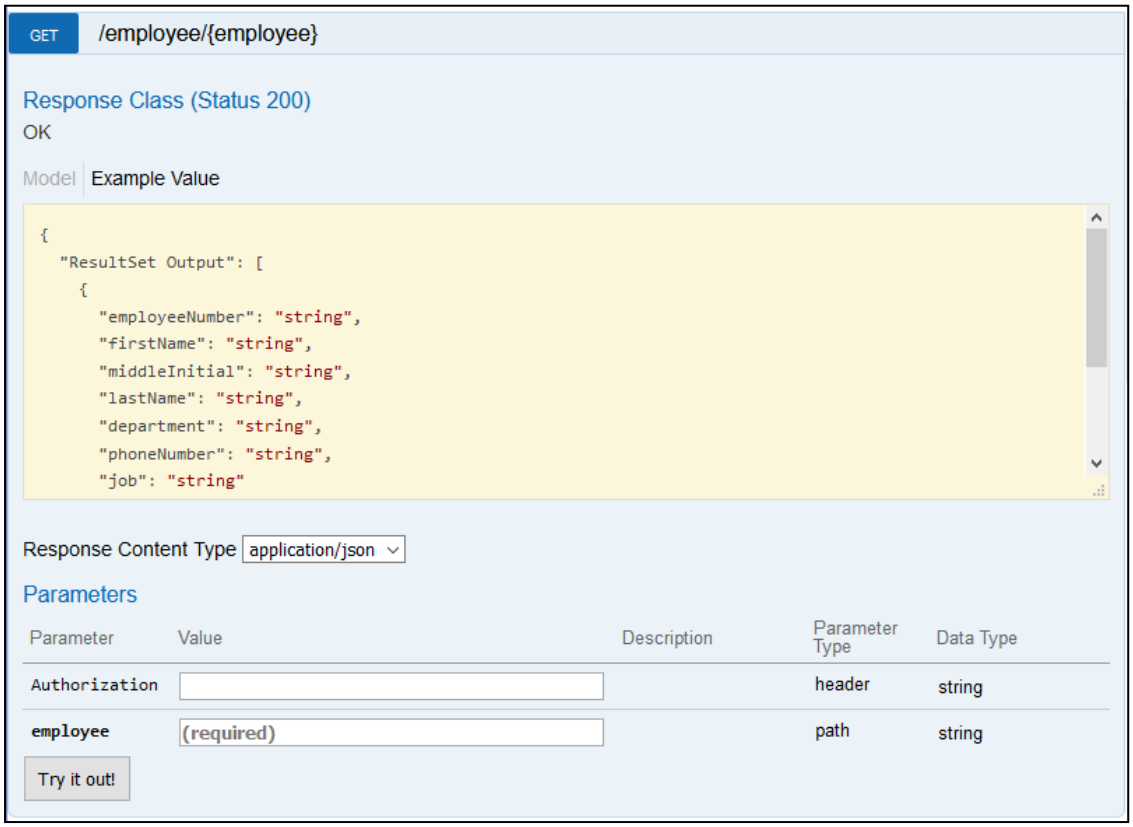


___6. Click on *List Operations* option in this view and this will display a list of available HTTP methods in this API.



Mitch Johnson (mitchj@us.ibm.com)

___7. Select the *GET* method for selecting an individual row from the table by clicking on the */employee/{employee}* URI string. Remember this was the *Path* specified for the *GET* method for the *selectEmployee* service when the API was defined. This action will expand this method in this view and provides a Swagger UI test client (you may have to use the slider bar and adjust the perspective to see the entire client).



___8. Enter ***000020*** in the box beside employee and ***Basic RnJlZDpmcmVkcHdk*** for *Authorization* and press the **Try it out!** button. You may see a Security Alert pop-up warning about the self-signed certificate being used by the z/OS Connect EE server. Click **Yes** on this pop-up.

___9. Scroll down the view and you should see the Request URL and Response Body which contains the results of the GET method (see below). Note that the columns removed from the interface in an earlier steps are not present.

Mitch Johnson (mitchj@us.ibm.com)

Request URL

```
https://wg31.washington.ibm.com:9453/db2/employee/000020
```

Request Headers

```
{
  "Accept": "application/json",
  "Authorization": "000020"
}
```

Response Body


```
{
  "StatusDescription": "Execution Successful",
  "ResultSet Output": [
    {
      "firstName": "MICHAEL",
      "lastName": "THOMPSON",
      "middleInitial": "L",
      "phoneNumber": "3476",
      "department": "B01",
      "job": "MANAGER ",
      "employeeNumber": "000020"
    }
  ],
  "StatusCode": 200
}
```


Response Code

```
200
```

___10. Repeat this process with the **DELETE** method and delete a row from the table. The *Response Body* should contain a JSON message like the one below:

Response Body


```
{
  "StatusDescription": "Execution Successful",
  "Update Count": 1,
  "StatusCode": 200
}
```

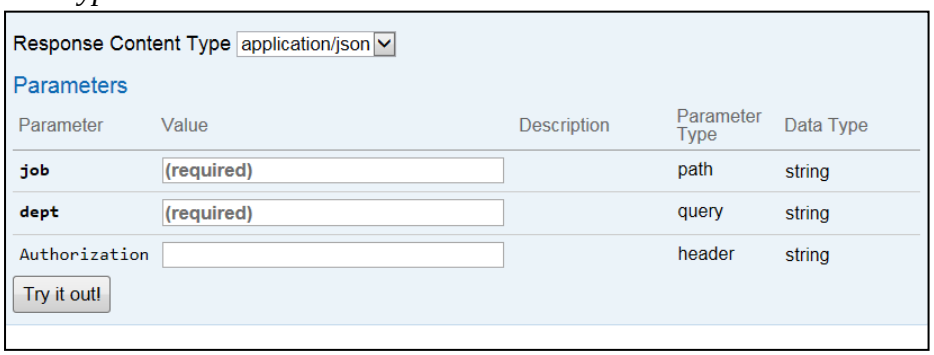
Repeat the **GET** and **DELETE** methods with other records (see table below) and verify the results are as expected.

EMPNO	FIRSTNME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	Birthdate	Salary	Bonus	COMM
000011	CHRISTINE	I	HAAS	A00	A1A1	1965-01-01	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
000110	VINCENZO	G	LUCCHESI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
000260	SYBIL	V	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

Note that the **GET** and **DELETE** methods do not required JSON in the *Request*. Only the path variable *employee* was required.

___11. Collapse the Swagger UI test areas for the **DELETE** and **GET** methods for */employee/{employee}* clicking on the URIs.

___12. Click on the URI for the **GET** method for */roles/{job}* to open its Swagger Test user interface and scroll down to the *Response Content Type* area.



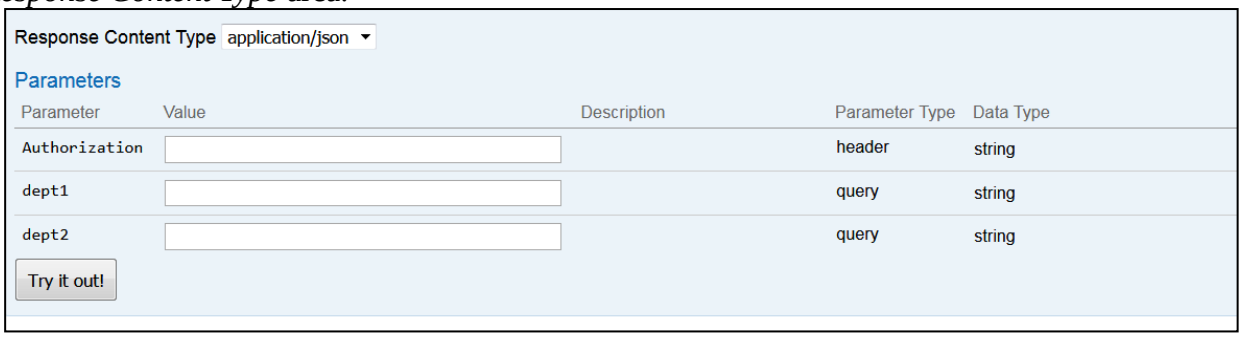
Note that this API requires two parameters, a path parameter *job* and a query parameter *dept*. These are present because the path /roles/{job}?dept was specified when the API was developed. Enter ***PRES*** for the j*ob* and ***A00*** as the *dept* and ***Basic RnJlZDpmcmVkcHdk*** for *Authorization* and press the **Try it Out**! button. Scroll down and you should see the following information in the *Response Body*.

Response Body


```
  "ResultSet Output": [
    {
      "firstName": "CHRISTINE",
      "lastName": "HAAS",
      "middleInitial": "I",
      "phoneNumber": "3978",
      "job": "PRES    ",
      "employeeNumber": "000010"
    },
    {
      "firstName": "CHRISTINE",
      "lastName": "HAAS",
      "middleInitial": "I",
      "phoneNumber": "A1A1",
      "job": "PRES    ",
      "employeeNumber": "000011"
    }
  ],
  "StatusCode": 200
}
```


Note that only the columns specified by the DBA in the Db2 native REST service appear

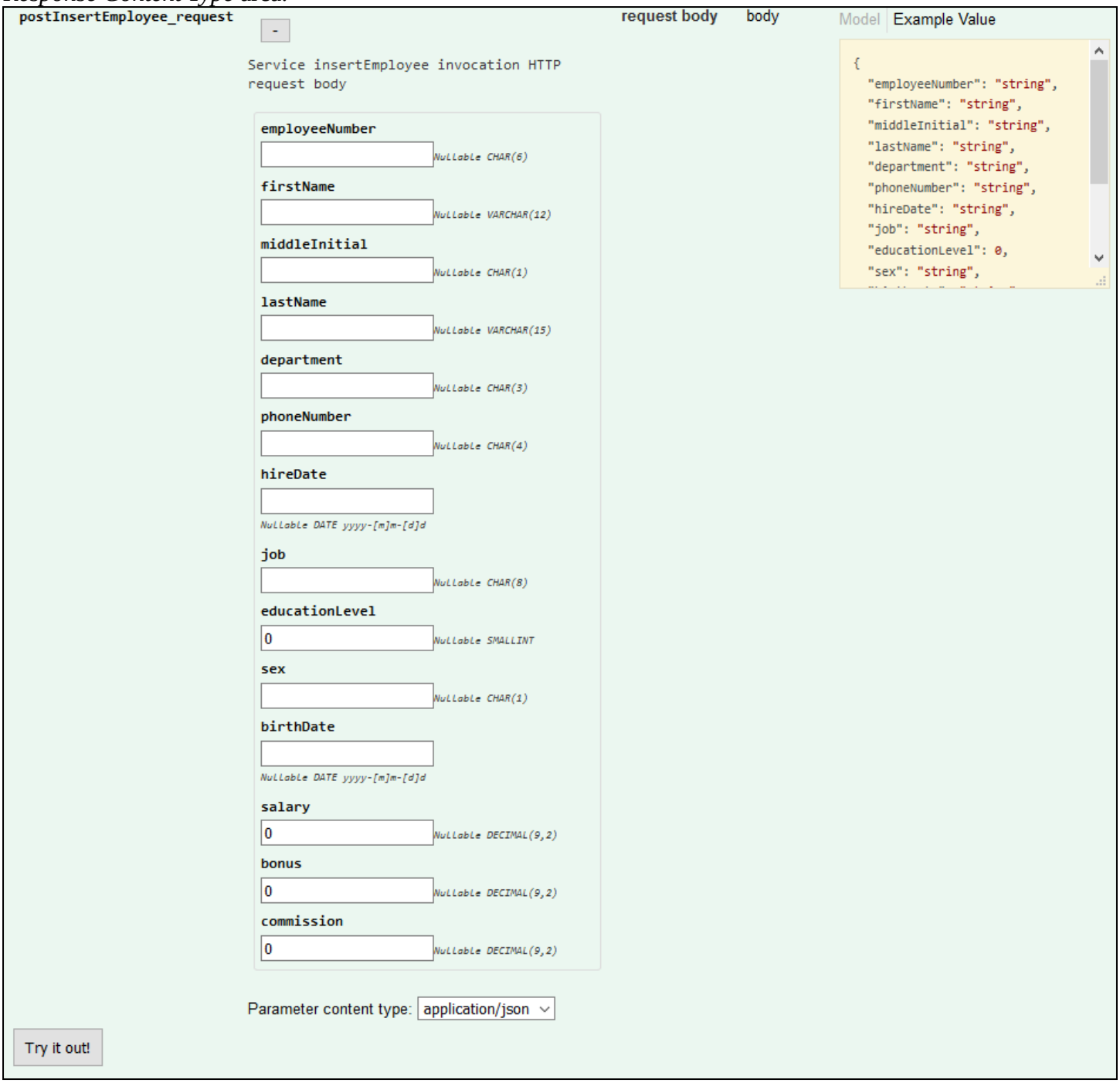
___13. Click on the URI for the **GET** method for */departments* to open its Swagger Test user interface and scroll down to the *Response Content Type* area.



Note that this API requires two query parameters, *dept1* and *dept2*. These are present because these parameters were added when the API was developed. Enter ***A01*** for the *dept1* and ***C01*** for *dept2* and ***Basic RnJlZDpmcmVkcHdk*** for *Authorization* and press the **Try it Out**! button. Scroll down and you should see the following information in the *Response Body*.

Try a few other combinations for *dept1* and *dept2* and compare the results with the above table.

___14. Click on the URI for the **POST** method for / to open its Swagger Test user interface and scroll down to the *Response Content Type* area.



Mitch Johnson (mitchj@us.ibm.com)

Enter ***Basic RnJlZDpmcmVkcHdk*** for *Authorization* and data for the other fieland press the **Try it Out**! button. Scroll down and you should see the following information in the *Response Body*.

Tech-Tip: Note that the fields *hireDate* and *birthDate* were derived from columns defined in the Db2 table as DATE types. The DATE type constraint was propagated in the JSON schema files for the JSON properties.

```
"hireDate": {
   "type": "string",
   "minLength": 8,
   "maxLength": 10,
   "pattern": "^(?![0]{4})([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|[1-2][0-
9]|3[0-1])$",
   "description": "Nullable DATE yyyy-[m]m-[d]d"
 },
```

This means that only strings matching the pattern that are valid calendar dates are valid for these fields.

Summary

You have verified the API. The API layer operates above the service layer you defined and tested earlier. The API layer provides a further level of abstraction and allows a more flexible use of HTTP verbs, and better mapping of data via the API editor function.

Mitch Johnson (mitchj@us.ibm.com)

Optional – Extend the API project by adding a PUT method.

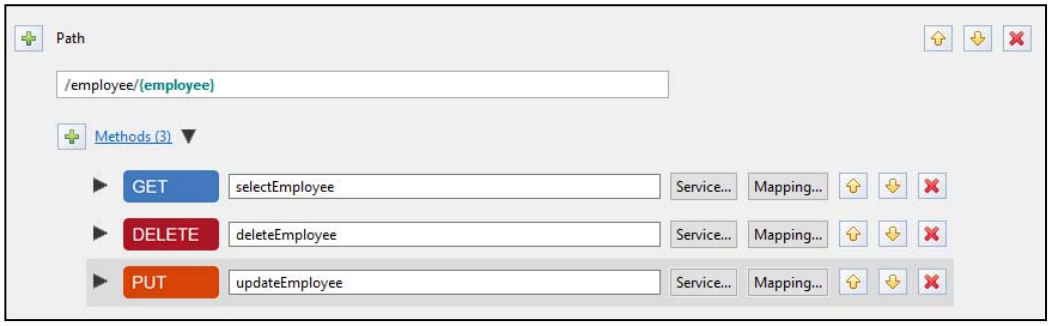
Using the previous instructions extend the API project to add support for updating a row in the USER1.EMPLOYEE table and then displaying the results.

____1. First submit job *DB2REST6* to create two new Db2 native services. Db2 native REST service *updateEmployee* updates the SALARY, BONUS and COMM columns in the Db2 table. Db2 native REST service *displayEmployee* will display all the columns of the table (remember Db2 native REST service *selectEmployee* only returns a subset of the columns).

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT  DD *
  UPDATE USER1.EMPLOYEE
         SET SALARY = :salary, BONUS  = :bonus, COMM   = :commission
    WHERE EMPNO = :employeeNumber
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE(SYSIBMSERVICE) -
NAME("updateEmployee") SQLENCODING(1047) -
DESCRIPTION('Insert an employee row into table USER1.EMPLOYEE')
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT  DD *
  SELECT * FROM USER1.EMPLOYEE WHERE EMPNO = :employeeNumber
//SYSTSIN  DD *
DSN SYSTEM(DSN2)
BIND SERVICE(SYSIBMSERVICE) -
NAME("displayEmployee") SQLENCODING(1047) -
DESCRIPTION('Display an employee row in table USER1.EMPLOYEE')
('Select an employee from table USER1.EMPLOYEE')
```

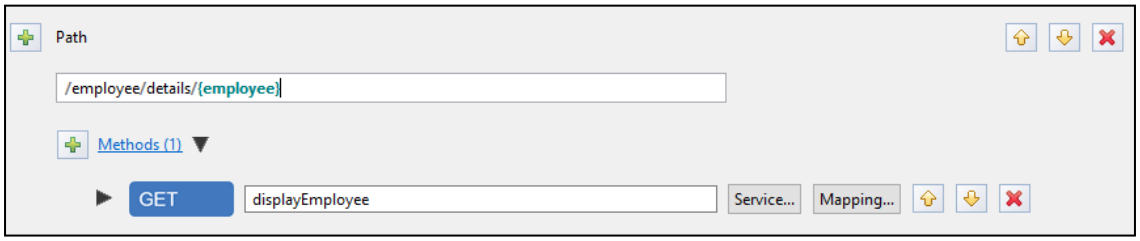
____2. Create two new z/OS Connect EE services, *updateEmployee* and *displayEmployee*, for these two new Db2 native REST services. Export these new services to the *Services* project and deploy them to the server.

____3. Add method a **PUT** method to path ***/employee/{employee}*** for service *updateEmployee* and performed the necessary mappings.

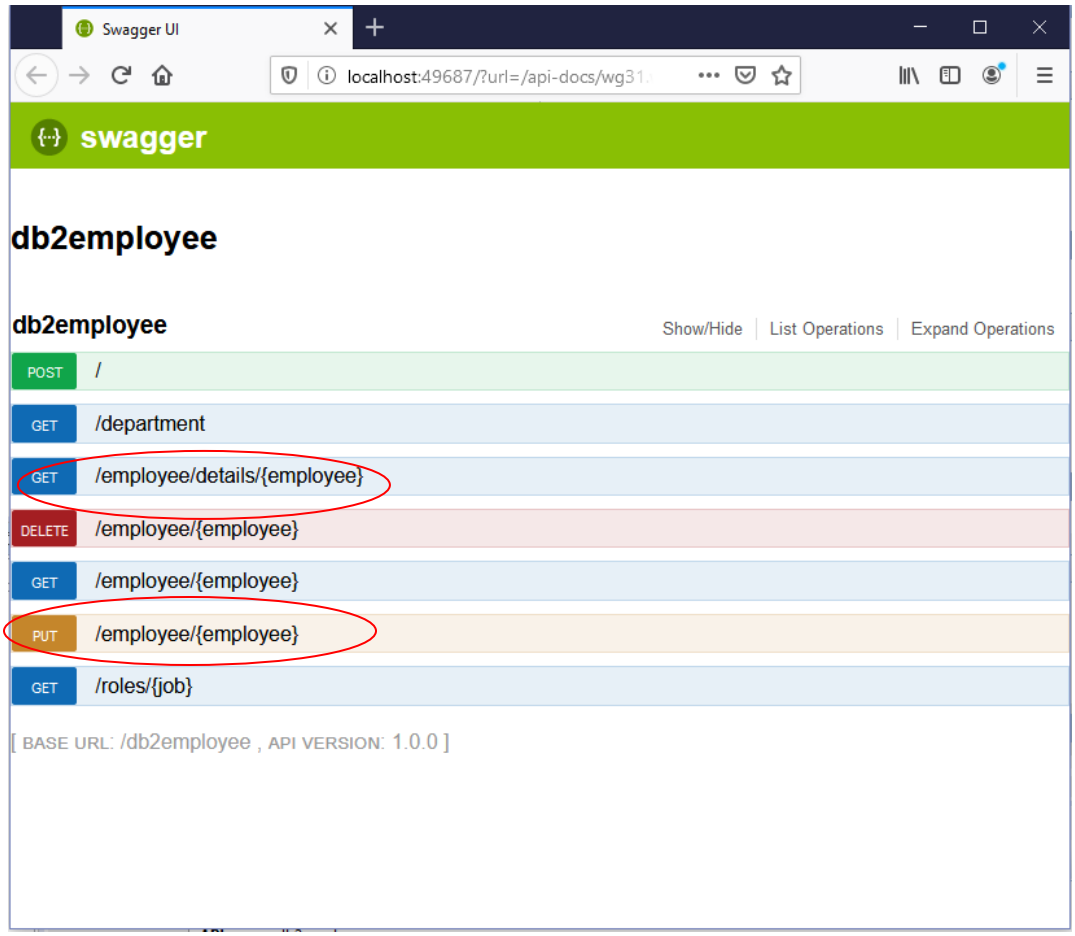


Mitch Johnson (mitchj@us.ibm.com)

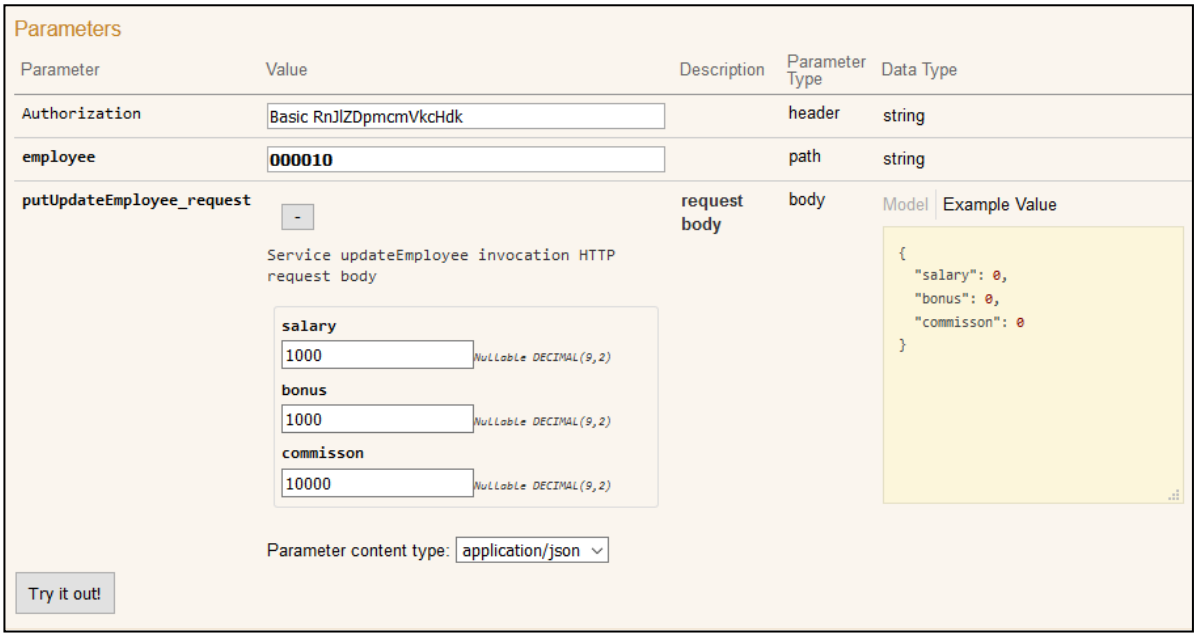
____4. Add a new path ***/employee/details/{employee}*** with a **GET** method for service *displayEmployee* and perform the necessary mappings.

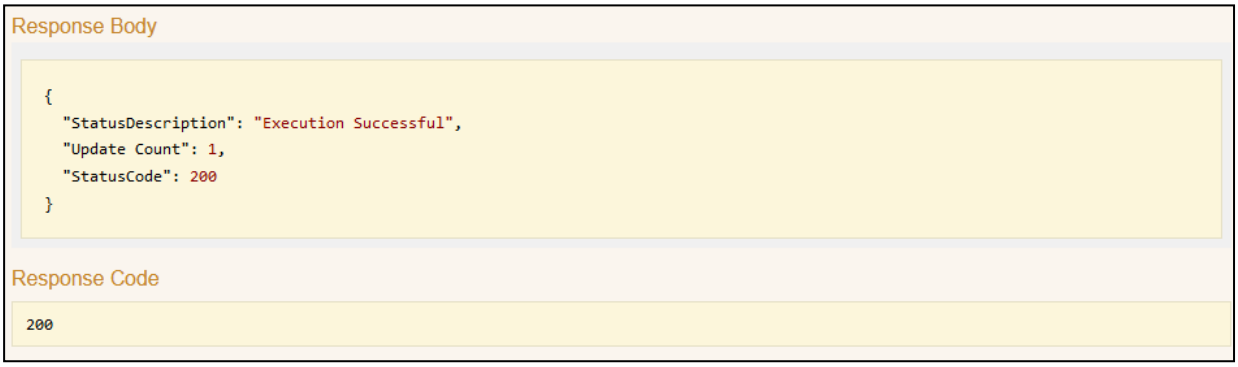


____5. Deploy the API project and test both new services using *Swagger UI* and test the new **PUT** and **GET** methods (see below).



The results for the **PUT** test should look similar to what is displayed below.





The results for the **GET** test for the *displayEmployee* API reqest should look like what is displayed below. Fields *SALARY*, *COMM*, and *BONUS* should be the values specified in the PUT request.

Response Body


```
{
  "StatusDescription": "Execution Successful",
  "ResultSet Output": [
    {
      "PHONENO": "3978",
      "EDLEVEL": 18,
      "SEX": "F",
      "FIRSTNME": "CHRISTINE",
      "MIDINIT": "I",
      "BIRTHDATE": "1933-08-14",
      "SALARY": 10000,
      "COMM": 10000,
      "LASTNAME": "HAAS",
      "WORKDEPT": "A00",
      "HIREDATE": "1965-01-01",
      "BONUS": 10000,
      "EMPNO": "000010",
      "JOB": "PRES    "
    }
```


Tech-Tip: The original SELECT statement did not use AS clauses for the column names so the original columns names were used for the JSON property names.

Congratulations, you have completed this exercise.