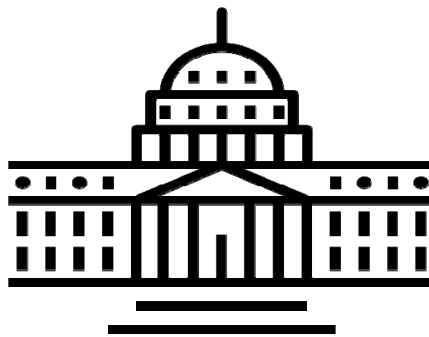


**IBM z/OS Connect (OpenAPI 3.0)**

# **Developing Native Server RESTful APIs for accessing Db2 REST Services**



**Washington  
Systems  
Center**

*Lab Version Date: August 8, 2023*

## Table of Contents

<b>Overview .....</b>	<b>3</b>
<b>Db2 REST services and z/OS Connect.....</b>	<b>4</b>
Db2 REST Services .....	4
<b>The z/OS Connect Designer Container environment .....</b>	<b>9</b>
The container configuration file .....	9
Considerations when deploying multiple APIs in a native server .....	10
Connecting to Db2 and the required server XML configuration .....	10
Basic security and the required server XML configuration .....	11
Accessing the z/OS Connect Designer log and trace files .....	13
<b>Developing a z/OS Connect APIs that accesses Db2 .....</b>	<b>15</b>
Configure the POST method for URI path /employees.....	17
Configure the GET method for URI path /employees/{employee}.....	34
<b>Testing the API's POST and GET methods .....</b>	<b>39</b>
<b>Complete the configuration of the API (Optional).....</b>	<b>43</b>
Configure the PUT method for URI path /employees/{employee} .....	43
Configure the GET method for URI path /employees/details/{employee} .....	47
Configure the DELETE method for URI path /employees/{employee} .....	52
Configure the GET method for URI path /roles/{job} .....	56
<b>Testing APIs deployed in a z/OS Connect Designer container .....</b>	<b>61</b>
<b>Deploying and installing APIs in a z/OS Connect Native Server .....</b>	<b>67</b>
Moving the API Web Archive file from the container to a z/OS OMVS directory .....	67
Updating the server xml .....	67
Defining the required RACF EJBRole resources.....	69
<b>Testing APIs deployed in a native z/OS server .....</b>	<b>70</b>
Using Postman.....	70
Using cURL .....	77
Using the API Explorer.....	79
<b>Additional information and samples.....</b>	<b>85</b>
JCL to define and load the Db2 table USER1.EMPLOYEE.....	86
Designer problem determination .....	87

**Important:** There is a folder on the Windows desktop named *CopyPaste Files*. This folder contains file with the commands and other text used in this workshop. Locate the file identified in the *General Exercise Information and Guidelines* section of this exercise and copy it to the desktop. Open the file and 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.

## Overview

The objective of these exercises is to gain experience with developing and deploying API using the *z/OS Connect Designer*. This exercise is offered in conjunction with a Washington Systems Center Wildfire workshop for z/OS Connect. For information about scheduling this workshop in your area contact your IBM representative.

**Important – You do not need any skills with Db2 to perform this exercise. Even if Db2 is not relevant to your current plans, performing the steps in this exercise will give additional experience using the *z/OS Connect Designer* to developing and administer APIs.**

### General Exercise Information and Guidelines

- ✓ This exercise requires using z/OS user identities *Fred*, *USER1* and *USER2*. The *Designer* passwords for these identities are *fredpwd*, *user1* and *user2* respectively and are case sensitive. The RACF password for these users are *FRED*, *USER1* and *USER2* respectively and are case insensitive.
- ✓ Any time you have any questions about the use of screens, features or tools do not hesitate to reach out for assistance.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *OpenAPI 3 development APIs CopyPaste* file.
- ✓ 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. For example, the text might reference host name *designer.ibm.com* when a screen shot shows the host as *designer.ibm.com* or even *localhost*. All these names resolve to the same IP address. Another example is that a section of a page has been expanded for display purposes. If a section or screen shot does not look exactly as what you are observing, consider maximizing or minimizing that section

## ***Db2 REST services and z/OS Connect***

Accessing a Db2 REST service from z/OS Connect differs from the ways in which z/OS Connect accesses the resources of other z/OS subsystems. Other subsystem's resources are accessed by using their normal subsystem interfaces (e.g., OTMA, IPIC, JMS, etc.).

A z/OS Connect Designer instance and server accesses Db2 not as a Db2 client using JDBC, but rather as a RESTful client accessing an existing Db2 REST service. This may raise the question as to what value-add does z/OS Connect provide if z/OS Connect can only access an existing Db2 REST service? 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. And finally (3) z/OS Connect provides security mechanism (e.g., OAUTH and JWT tokens) not available with Db2. If a full function RESTful API with support for the major HTTP methods (POST, PUT, GET and DELETE), or transforming JSON payloads and/or additional authentication methods are required, then z/OS Connect is the solution

### ***Db2 REST Services***

Db2 REST services 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). The Db2 REST services used in this exercise were created using the Db2 BIND command as shown in this section.

- Db2 REST service *selectEmployee* was defined using the BIND JCL below.

```
//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("zCEEService") -
  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, COMMISSION, etc. from the selection by omitting these columns from the SELECT statement. The DBA's use of the *AS* clause 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("zCEEService"."selectEmployee")

**Tech-Tip:** A minimum of EXECUTE authority on package zCEEService.selectEmployee would be required to have the ability to execute this service.

- Db2 REST service *deleteEmployee* was defined using the BIND command below.

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

The Db2 native REST Service named *deleteEmployee* deletes a row from table USER1.EMPLOYEE using a JSON request message like the one below.

```
{
  "employeeNumber": "000340"
}
```

You should see this result in the response area.

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

**Tech-Tip:** The update count, status code and description fields in a Db2 REST service response message will play an important part in determining if a request changed a Db2 resource.

- Db2 REST service *selectByRole* was defined using the BIND command below.

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

This service 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, COMMISSION, etc. from the selection by omitting these columns from the SELECT statement. The DBA's use of the *AS* clause will also ensure the assigning of meaningful JSON property names rather than the original Db2 column names to the JSON request and response messages.

- Db2 REST service insertEmployee was defined using the BIND command below

```
//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("zCEEService") -
NAME("insertEmployee") -
SQLENCODING(1047) -
DESCRIPTION('Insert an employee into table USER1.EMPLOYEE')
/*
```

This service 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.

- 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). These were defined by the BIND commands below.

```
//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("zCEEService") -
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("zCEEService") -
NAME("displayEmployee") SQLENCODING(1047) -
DESCRIPTION('Display an employee row in table USER1.EMPLOYEE')
('Select an employee from table USER1.EMPLOYEE')
```

There is a pattern in the response messages from the Db2 REST services. When a Db2 resource is added, updated, or deleted, the response message included an *Update Count* field. The field contains the number of Db2 resources affected by this invoking this service. In the same token, when Db2 resources were retrieved, the Db2 resources were returned in a list or array (e.g., *Resultset Output*) containing one or more list elements. These fields will be used in the *z/OS Connect Designer* to know if a specific REST method was successful or not.



## The z/OS Connect Designer Container environment

Before developing an API, it is useful to understand the configuration required for the z/OS Connect Designer the development environment.

### The container configuration file

Regardless of whether *Docker Engine*, *Docker Desktop*, *Podman* or some other container runtime product is being used, the container's environment required configuration.

First, the container requires that Db2 related environment variable be provided. These variables are used to customize the Db2 related server XML configuration elements. For this exercise, the container was configured with these environment variables set in the *docker-compose.yaml* file (in **bold**).

```
version: "3.2"
services:
  zosConnect:
    image: icr.io/zosconnect/ibm-zcon-designer:3.0.57
    user: root
    environment:
      - BASE_PATH=roster
      - CICS_USER=USER1
      - CICS_PASSWORD=USER1
      - CICS_HOST=wg31.washington.ibm.com
      - CICS_PORT=1491
      - DB2_USERNAME=USER1
      - DB2_PASSWORD=USER1
      - DB2_HOST=wg31.washington.ibm.com
      - DB2_PORT=2446
      - HTTP_PORT=9080
    ports:
      - "9449:9443"
      - "9086:9080"
    volumes:
      - ./project:/workspace/project
      - ./logs:/logs/
      - ./certs:/output/resources/security/
```

Connecting to a Db2 subsystem requires the addition of a *zosconnect\_db2Connection* configuration element to the container's Liberty configuration. And since this API has role-based security elements configured, additional configuration elements for a basic registry and authorization roles are also required. These Liberty configuration elements are described next.

**Tech-Tip:** The contents of this docker-compose.yaml file were based on the example found in the z/OS Connect product documentation at URL  
<https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=tutorials-creating-db2-zos-connect-api>

## Considerations when deploying multiple APIs in a native server

Deploying multiple APIs into a single native server requires that each API have a unique context root, otherwise there may be collisions with the URI paths of other APIs. The optimal time to ensure this context root is unique and/or to provide a unique context root is before the YAML document is imported into the z/OS Connect Designer.

Review the YAML document (`c:\z\openapi3\yaml\employees.yaml`) and locate the **Servers** section (see an example below). Each element in the *server* attribute will have an *url* attribute where a base path could be provided. The default base path is simply a slash (/). Again, if multiple APIs are to be deployed into a single native server, a unique value needs to be provided for each API. In this exercise, the base path has already been set in the YAML file to **/roster** (as shown below).

```
servers:
- url: /roster
```

Making this change in the z/OS Connect Designer requires the addition of a *webApplication* configuration element. This element will explicitly associate the WAR file developed for the API in this exercise with the value of **/roster** as the context root (see below).

To avoid hard coding a value for the context root, an additional environment variables **BASE\_PATH** was added to the *docker-compose.yaml* file and set a value of **roster**. The environment variable `${BASE_PATH}` will be used to provide values for the *name* and *contextRoot* attributes in the *webApplication* configuration element, see below.

```
<?xml version="1.0" encoding="UTF-8"?>
<server>

<webApplication id="myApi" name="${BASE_PATH}" contextRoot="/${BASE_PATH}"
  location="${server.config.dir}dropins/api.war" />

</server>
```

## Connecting to Db2 and the required server XML configuration

The *zosconnect\_db2Connection* element used to connect to a Db2 subsystem in this exercise looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="Db2 zosconnect_db2Connection ">
  <featureManager>
    <feature>zosconnect:db2-1.0</feature>
  </featureManager>

  <zosconnect_db2Connection id="db2Conn"
    host="${DB2_HOST}"
    port="${DB2_PORT}"
    credentialRef="commonCredentials" />

  <zosconnect_credential id="commonCredentials"
    user="${DB2_USERNAME}"
    password="${DB2_PASSWORD}" />

</server>
```

Notice the environment variables `${DB2_HOST}`, `${DB2_PORT}`, `${DB2_USERNAME}` and `${DB2_PASSWORD}` are set to the values provided in the *docker-compose.yaml* file.

**Basic security and the required server XML configuration**

The *basicRegistry* and *authorization-roles* elements used in this exercise looks like this:

```
<server description="basic security">

  <!-- Enable features -->
  <featureManager>
    <feature>appSecurity-2.0</feature>
    <feature>restConnector-2.0</feature>
  </featureManager>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <basicRegistry id="basic" realm="zosConnect">
    <user name="Fred" password="fredpwd" />
    <user name="user1" password="user1" />
    <user name="user2" password="user2" />
    <group name="Manager">
      <member name="Fred"/>
    </group>
    <group name="Staff">
      <member name="Fred"/>
      <member name="user1"/>
    </group>
  </basicRegistry>

  <administrator-role>
    <group>Manager</group>
  </administrator-role>

  <authorization-roles id="zCEERoles">
    <security-role name="Manager"><group name="Manager"/></security-role>
    <security-role name="Staff"><group name="Staff"/></security-role>
  </authorization-roles>
</server>
```

In the above configuration, identity *Fred* is a member of the *Manager* and *Staff* group. Identities *USER1* and is a member of the *Staff* group. Identity *USER2* is not a member of any role-based groups.

The role names *Manager* and *Staff* correspond to the values that appear in the API's specification document . In this example, a default role of *Manager* is defined in the root of the OpenAPI definition. Each of the GET operations defines a role of *Staff*. So only users in or with access to the *Staff* role all allowed to perform the GET methods. And only users in or with access to the *Manager* role all allowed to perform the POST, PUT and DELETE methods. A user with only *Staff* access with receive an HTTP 403 (Forbidden) response if they try to invoke one of these privileged methods.

```

openapi: 3.0.0
x-ibm-zcon-roles-allowed:
  - Manager
security:
  - BasicAuth: []
  - BearerAuth: []
paths:
  "/roles/{job}":
    get:
    -----
      x-ibm-zcon-roles-allowed:
        - Staff
    -----
  /employees:
    post:
    -----
  /employees/details/{employee}:
    get:
    -----
      x-ibm-zcon-roles-allowed:
        - Staff
    -----
  "/employees/{employee}":
    get:
    -----
      x-ibm-zcon-roles-allowed:
        - Staff
    -----
    delete:
    -----
    put:
    -----

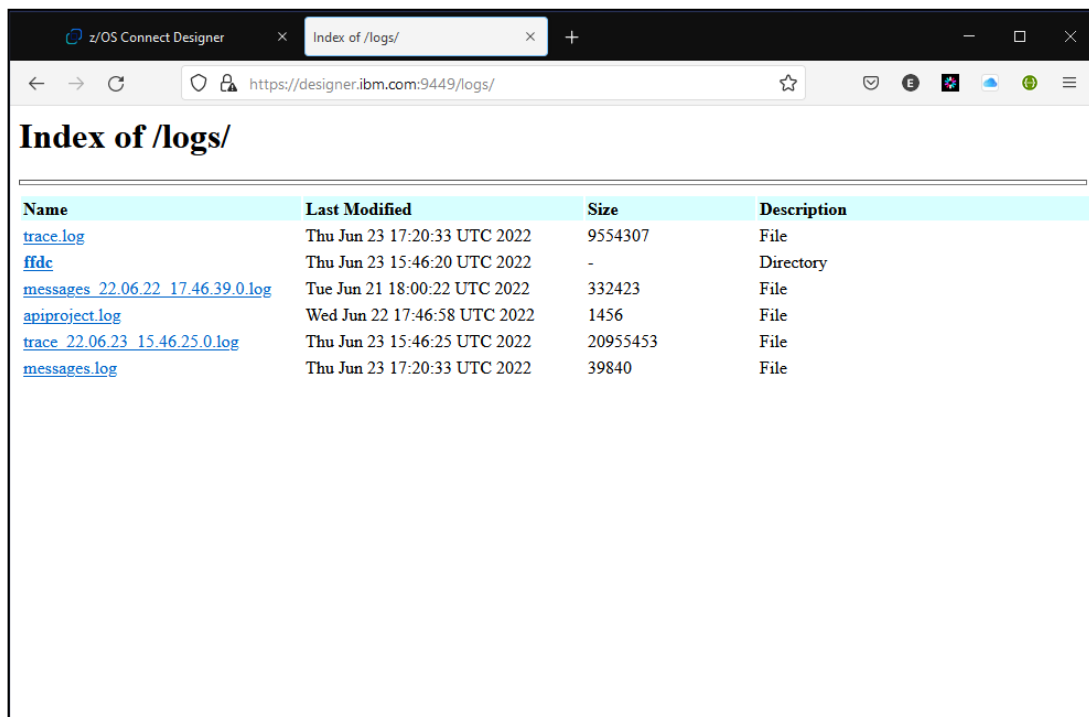
```

## Accessing the z/OS Connect Designer log and trace files

The Liberty server in which the z/OS Connect Designer has been further customized with the addition of the server XML configuration elements below. These XML configuration elements enables the Liberty server to become a file server.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="Default server">
  <webApplication id="resources-dropins" name="dropins"
    location="/opt/ibm/wlp/usr/servers/defaultServer/dropins">
    <web-ext context-root="dropins"
      enable-file-serving="true" enable-directory-browsing="true">
      <file-serving-attribute name="extendDocumentRoot"
        value="/opt/ibm/wlp/usr/servers/defaultServer/dropins" />
      </web-ext>
    </webApplication> >
  <webApplication id="resources-logs" name="logs"
    location="/logs">
    <web-ext context-root="logs"
      enable-file-serving="true" enable-directory-browsing="true">
      <file-serving-attribute name="extendDocumentRoot"
        value="/logs" />
      </web-ext>
    </webApplication> >
</server>
```

This is very useful because this allows the viewing of the server's log and trace file from a browser. This means an API developer using *z/OS Connect Designer* in one tab of browser will be able to monitor the messages and/or traces in other browser tabs as they are developing or testing their API. To access the server's logs directory, start with the same host and port as the *Designer* but with the URI path to */logs*. Double clicking on a file such as *trace.log* or *messages.log* allows the real time monitoring of trace messages or server messages by clicking the browser's refresh button.



Name	Last Modified	Size	Description
<a href="#">trace.log</a>	Thu Jun 23 17:20:33 UTC 2022	9554307	File
<a href="#">fdd</a>	Thu Jun 23 15:46:20 UTC 2022	-	Directory
<a href="#">messages 22.06.22 17.46.39.0.log</a>	Tue Jun 21 18:00:22 UTC 2022	332423	File
<a href="#">apiproject.log</a>	Wed Jun 22 17:46:58 UTC 2022	1456	File
<a href="#">trace 22.06.23 15.46.25.0.log</a>	Thu Jun 23 15:46:25 UTC 2022	20955453	File
<a href="#">messages.log</a>	Thu Jun 23 17:20:33 UTC 2022	39840	File

For example, using this technique the details of a SQL request and any SQL errors will appear in a *trace.log*. In this case this is information not returned in the response message but written to the trace by the service provider. This is very useful when the expected results are not returned.

```

org.apache.cxf.jaxrs.impl.ResponseImpl@dc04a5de
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > db2AssetWrapper Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@dc04a5de
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > constructHeadersObject Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@dc04a5de
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < constructHeadersObject Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < db2AssetWrapper Exit
{"headers":
{"connection":"close","Content-Language":"en-US","Content-Length":"228","content-type":"application/json; charset=UTF-8","Date":"Tue, 21 Jun 2022
14:20:08 GMT","Server":"DB2 DDF Native REST, DSN2LOC, DSNLJENG 10/02/19 UI65644","X-Correlation-ID":"C0A80067.H41C.DBB0633B32BF","X-Powered-
By":"DB2 for z/OS","Content-Type":"application/json; charset=UTF-8"},"body":{"statusCode":500,"statusDescription":"Service
zCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-180, SQLSTATE=22007, Message=THE DATE, TIME, OR TIMESTAMP VALUE *N IS
INVALID Error Location:DSNLJXUS:6"},"cookies":{},"statusCode":500}
[6/21/22 14:20:05:750 UTC] 000007c3 id=9223352b com.ibm.zosconnect.engine.impl.ResponseDataImpl > setAsset Entry
{"headers":
{"connection":"close","Content-Language":"en-US","Content-Length":"228","content-type":"application/json; charset=UTF-8","Date":"Tue, 21 Jun 2022
14:20:08 GMT","Server":"DB2 DDF Native REST, DSN2LOC, DSNLJENG 10/02/19 UI65644","X-Correlation-ID":"C0A80067.H41C.DBB0633B32BF","X-Powered-
By":"DB2 for z/OS","Content-Type":"application/json; charset=UTF-8"},"body":{"statusCode":500,"statusDescription":"Service
zCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-180, SQLSTATE=22007, Message=THE DATE, TIME, OR TIMESTAMP VALUE *N IS
INVALID Error Location:DSNLJXUS:6"},"cookies":{},"statusCode":500}
[6/21/22 14:20:05:750 UTC] 000007c3 id=9223352b com.ibm.zosconnect.engine.impl.ResponseDataImpl < setAsset Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < invoke Exit
[6/21/22 14:20:05:750 UTC] 000007c3 id=810e88ae com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl < invokeEndpoint Exit
com.ibm.zosconnect.engine.impl.ResponseDataImpl@9223352b
[6/21/22 14:20:05:750 UTC] 000007c3 id=810e88ae com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl > createResponseJson Entry

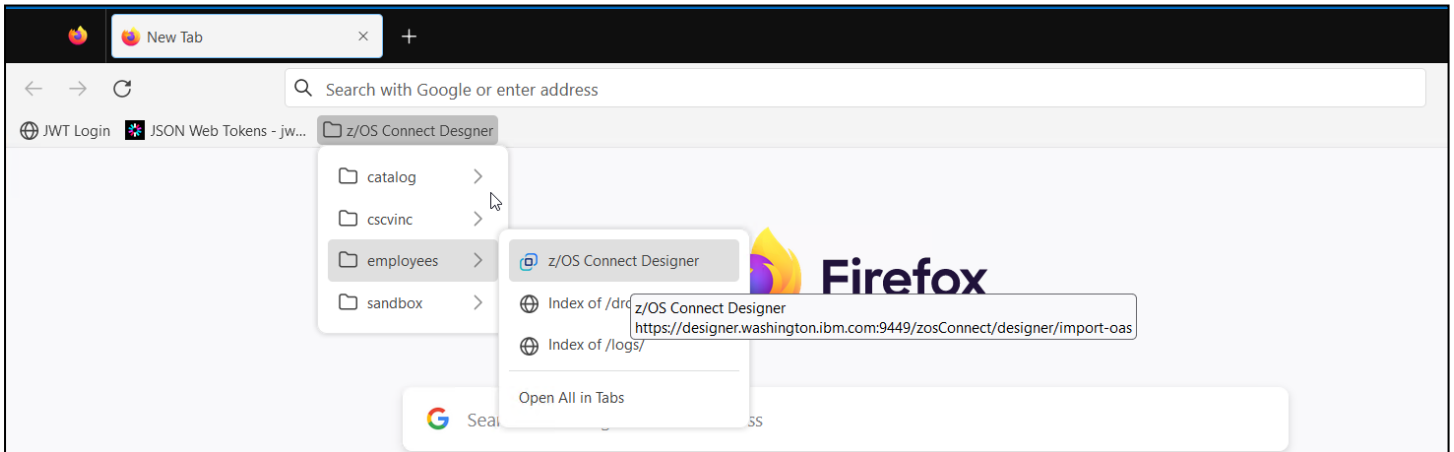
```

## Developing a z/OS Connect APIs that accesses Db2

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

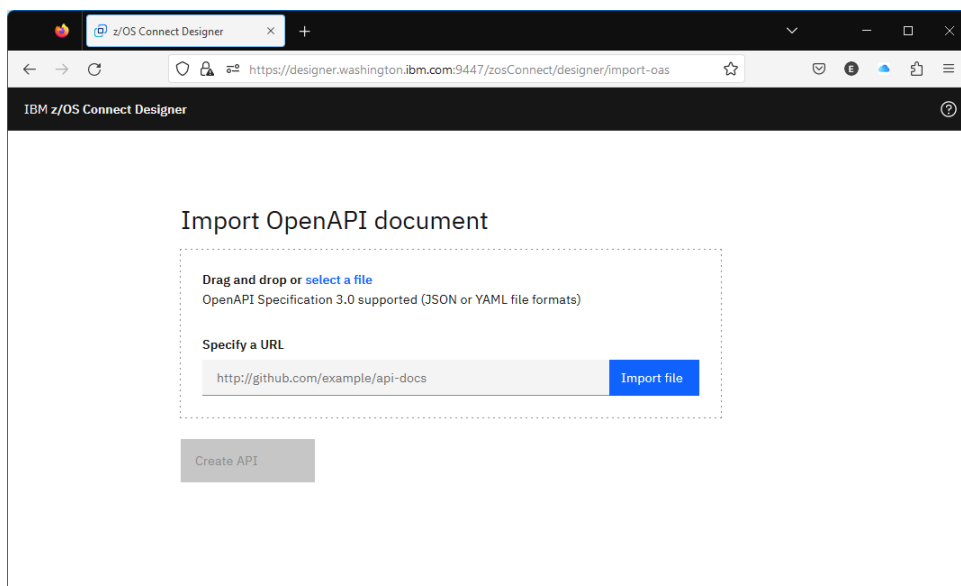
1. Start by opening the Firefox browser and going to URL  
<https://designer.washington.ibm.com:9449/zosConnect/designer/>

As an alternative, you could use the provided bookmark (see below) to access the Designer.

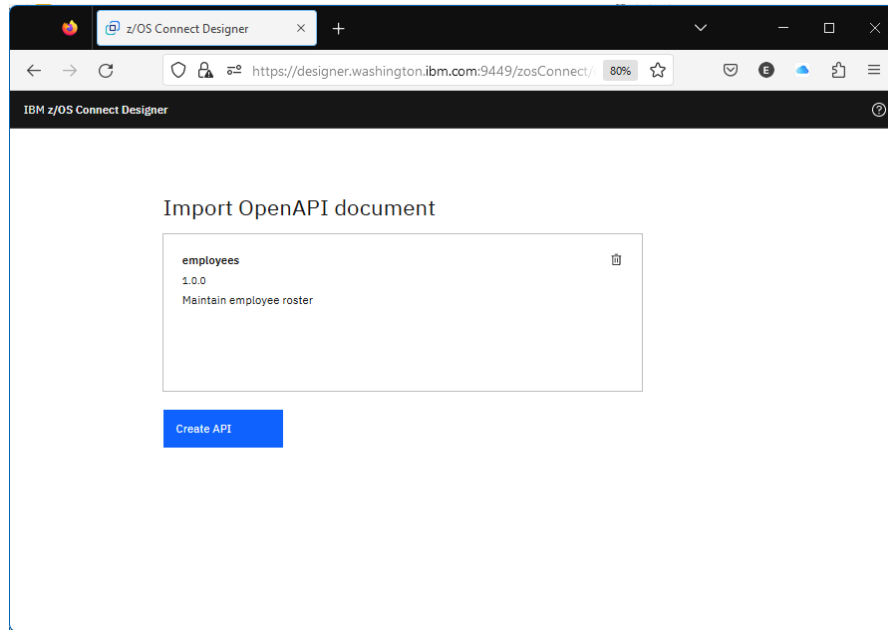


**Tech-Tip:** Be patient. It may take a while for the first page to fully load. Numerous background activities are being performed to fully initialize the application.

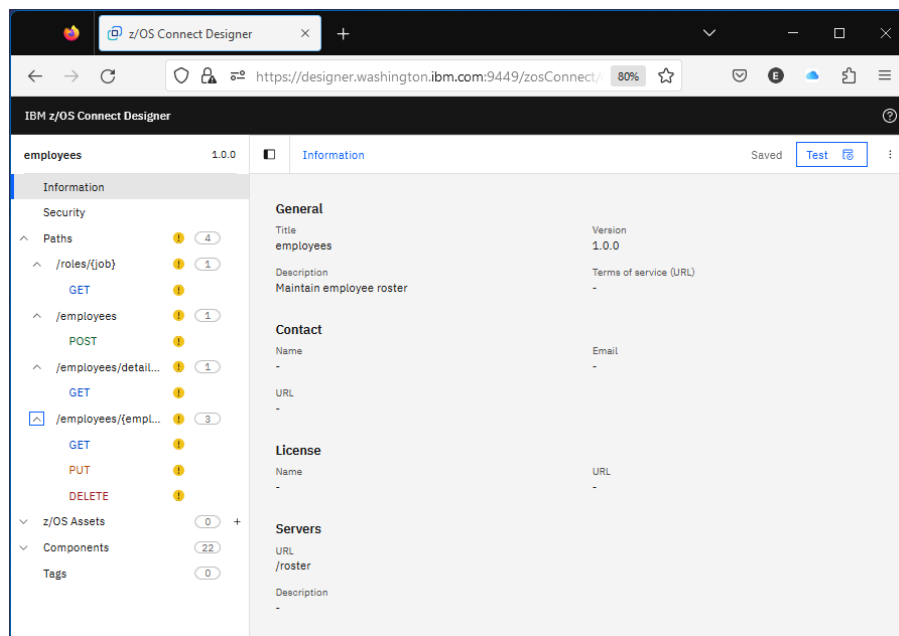
2. The first window you will see in a 'fresh' *Designer* environment gives you the opportunity to import an OpenAPI document. On the *Import OpenAPI document* window, click on [select a file](#) and traverse in the *File Upload* window to the directory where the specification document files are stored, e.g., *C:/z/openApi3/yaml*. Select file *employee.yaml* and click the **Open** button to continue.



3. On the next *Import OpenAPI document* window, click the **Create API** button to complete the importation of the specification document file into the *Designer*.



4. The next *Designer* page to be displayed will be the details of the API provided by the specification document. Expand the **Paths** on the left-hand side and you will see the URI paths of the API. Expand the URI paths will display the individual methods of each path. For example, expanding URI paths `/employees` and `/employees/{employee}` will display the *POST*, *GET*, *PUT* and *DELETE* methods associated with these URI paths (see below).



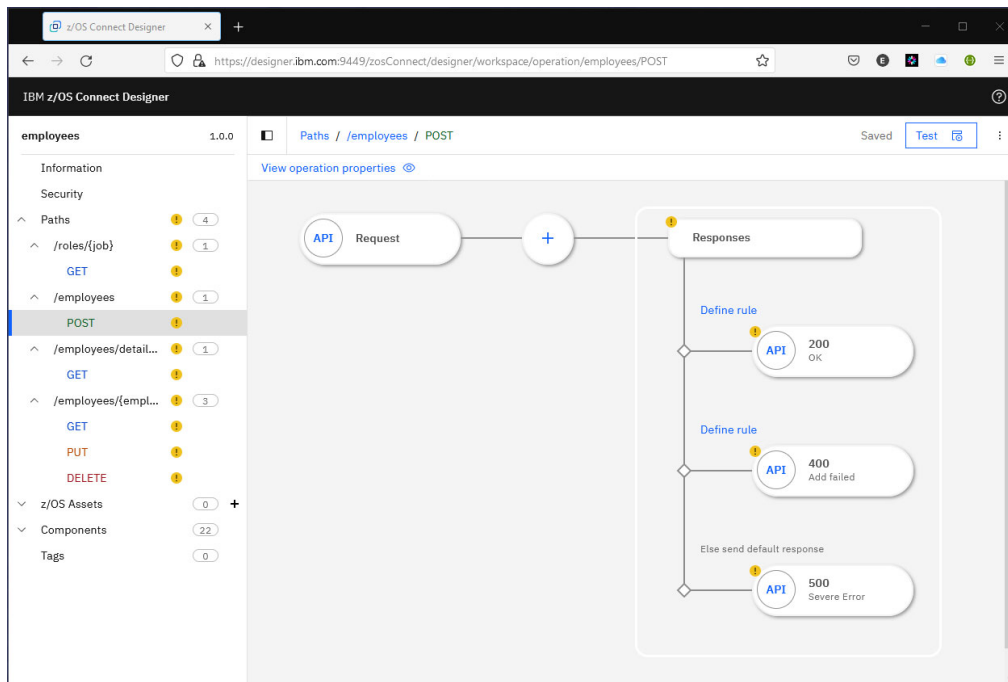


*Important: When using this tool, monitor the upper right-hand corner of the page. You will see either status of either **Saved** or **Saving**. It is suggested that you wait until changes are saved before continuing using the Designer.*

**Tech-Tip:** The yellow exclamation marks simply indicate the underlying configuration for this element is incomplete. As the exercise progresses, the exclamation marks will disappear.

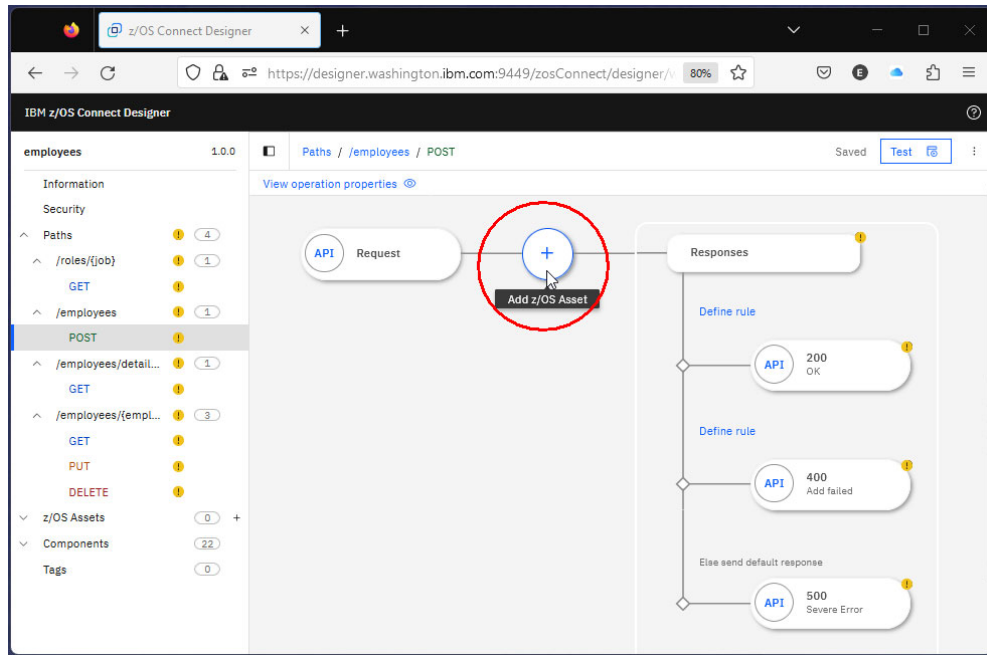
## Configure the POST method for URI path /employees

1. Selecting a method will display the operation properties of the method. Start with the *POST* method under */employees* and by selecting it, the view like the one below will appear.

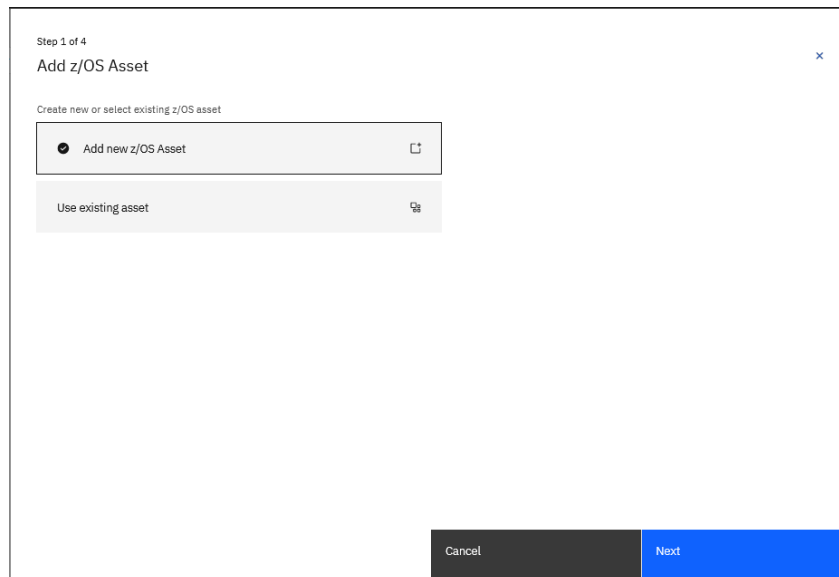


In the example above, we see that the specification document defined 3 responses for this method. One is a 200-status code which indicate the invocation of the method (an insert) was successful. A 400-status code which indicates, in this case, that the request to insert an employee record failed. And finally, a 500-status code which indicates a severe error has occurred while processing the request.

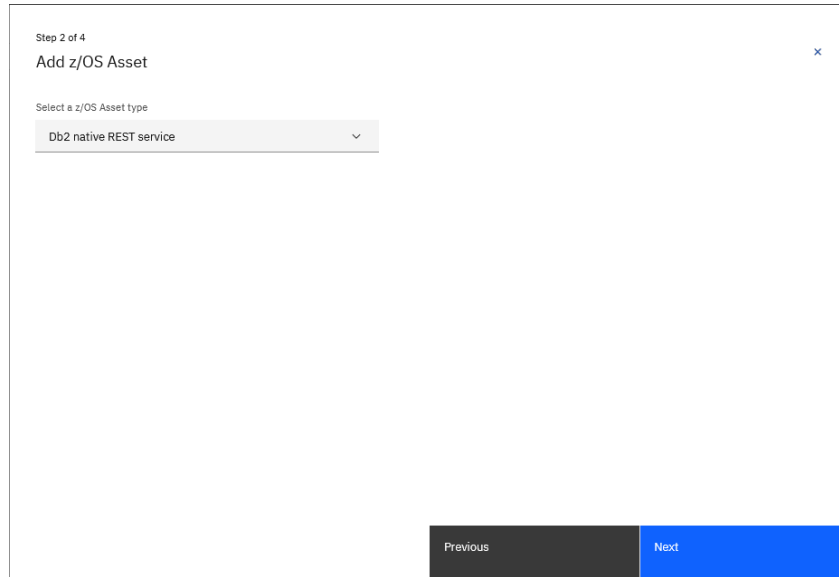
2. The first step in configuring this method for this URI path is to associate it with a z/OS asset or resource. Click the plus sign on the page to start the association of a z/OS asset with this URI path.



3. On the *Add z/OS Asset (Step 1 of 4)* page, select the *Add new z/OS Asset* and press **Next** to continue.



4. On the *Add z/OS Asset (Step 2 of 4)* page, use the pull-down arrow and select *Db2 native REST service* and press **Next** to continue.



Step 2 of 4

Add z/OS Asset

Select a z/OS Asset type

Db2 native REST service

Previous Next

5. On the *Add z/OS Asset (Step 3 of 4)* page, use the pull-down arrow and select the *db2conn* Db2 connection. This action will cause the full page to be displayed. Press **Import from Db2 service manager** to access Db2 and to list the available Db2 REST services. Note that the *collection ID*, *service name* and *version* can be used to filter the list.

Step 3 of 4
Add z/OS Asset

Select a Db2 connection

db2Conn

Import from Db2 service manager

Db2 native REST service collection ID

e.g. SYSIBMSERVICE

Db2 native REST service name

e.g. myService

Db2 native REST service version (optional)

e.g. V1

Import Db2 native REST service request schema

Drag and drop or [select a file](#)  
JSON schema specification draft 4 and 5 supported

Specify a URL

http://github.com/example/api-docs

Import file

Import Db2 native REST service response schema

Drag and drop or [select a file](#)  
JSON schema specification draft 4 and 5 supported

Specify a URL

http://github.com/example/api-docs

Import file

Previous
Next

**Tech-Tip:** The name *db2conn* is the name of the *zosconnect\_db2Connection* configuration element described earlier in this exercise.

6. A list of the available Db2 REST services will be displayed. In this case we want to associate the method with the Db2 REST service *insertEmployee* in the *zCEEService* collection. In this screen shot below, this service is on the first page. If it had not been, there would have been a need to use the page forward arrow at the bottom to go to page 2 of the list to display other pages of the list.

Add z/OS Asset / Import Db2 native REST service

Import Db2 native REST service

Select a Db2 connection

db2Conn

Filter table

12 Db2 native REST services found

	Service name	Version	Collection ID	Path	Description	Status
▼ ○	addEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Add the details of an ind...	Available
▼ ○	deleteEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Remove the details of a...	Available
▼ ○	getEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of a spec...	Available
▼ ○	getEmployees	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of all em...	Available
▼ ○	updateEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Update the details of an ...	Available
▼ ○	deleteEmployee	V1	zCEEService	/services/zCEEService/d...	Delete an employee fro...	Available
▼ ○	displayEmployee	V1	zCEEService	/services/zCEEService/d...	Display an employee in t...	Available
▼ ○	insertEmployee	V1	zCEEService	/services/zCEEService/i...	Insert an employee into ...	Available
▼ ○	selectByDepartments	V1	zCEEService	/services/zCEEService/s...	Select employees by de...	Available
▼ ○	selectByRole	V1	zCEEService	/services/zCEEService/s...	Select an employee bas...	Available

Items per page: 10 1-10 of 12 items

1 of 2 pages

Previous Import

7. Select the radio button beside the *insertEmployee* service under *Service Name* and in collection *zCEEService* see below. Press the **Import** button to have the Db2 native REST service information retrieved from Db2.

Add z/OS Asset / Import Db2 native REST service

Select a Db2 connection

db2Conn

Filter table

12 Db2 native REST services found

	Service name	Version	Collection ID	Path	Description	Status
<input type="radio"/>	addEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Add the details of an ind...	Available
<input type="radio"/>	deleteEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Remove the details of a...	Available
<input type="radio"/>	getEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of a spec...	Available
<input type="radio"/>	getEmployees	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Get the details of all em...	Available
<input type="radio"/>	updateEmployee	V1	SYSIBMSERVICE	/services/SYSIBMSERVI...	Update the details of an...	Available
<input type="radio"/>	deleteEmployee	V1	zCEEService	/services/zCEEService/...	Delete an employee fro...	Available
<input type="radio"/>	displayEmployee	V1	zCEEService	/services/zCEEService/...	Display an employee in ...	Available
<input checked="" type="radio"/>	insertEmployee	V1	zCEEService	/services/zCEEService/i...	Insert an employee into...	Available
<input type="radio"/>	selectByDepartments	V1	zCEEService	/services/zCEEService/s...	Select employees by de...	Available
<input type="radio"/>	selectByRole	V1	zCEEService	/services/zCEEService/s...	Select an employee bas...	Available

Previous Import

8. This will return you back to the *Add z/OS Asset (Step 3 of 4)* page with the details (request and response schema, etc.) populated from the Db2 service repository. Click **Next** to continue.

Step 3 of 4

Add z/OS Asset

Select a Db2 connection

db2Conn

Import from Db2 service manager

Db2 native REST service collection ID

zCEEService

Db2 native REST service name

insertEmployee

Db2 native REST service version (optional)

V1

Import Db2 native REST service request schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "employeeNumber": {
      "type": [
        "null",
        "string"
      ],
      "maxLength": 6,
      "description": "Nullable CHAR(6)",
      "firstName": {
        "type": [
          "null",
          "string"
        ],
        ...
      }
    }
  }
}
```

Import Db2 native REST service response schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "Update Count": {
      "type": "integer",
      "multipleOf": 1,
      "minimum": 0,
      "maximum": 32767,
      "description": "Update Count",
      "StatusDescription": {
        "type": "...
      }
    }
  }
}
```

Previous Next

9. On the *Add z/OS Asset (Step 4 of 4)* page, we are given an opportunity to rename or update the description of the z/OS asset. In this exercise, simply press **Add z/OS Asset** to continue. Eventually you see a brief message that the asset has been added successfully and the operation properties page will reflect the z/OS asset request mapping details (see below). On this page we are seeing the request properties from the Db2 request schema. What needs to be done is to map these fields to the request schema properties of the API as defined in the specification document that described the entire API.

Paths / /employees / POST

View operation properties

API Request

DB2 zCEEService-insertEmployee...  
Insert an employee into table U...

Responses

Define rule

API 200 OK

Define rule

400

zCEEService-insertEmployee-V1

z/OS Asset options

Request Response z/OS Asset details

Edit mapping View structure

Map fields from the API request into the z/OS Asset request.

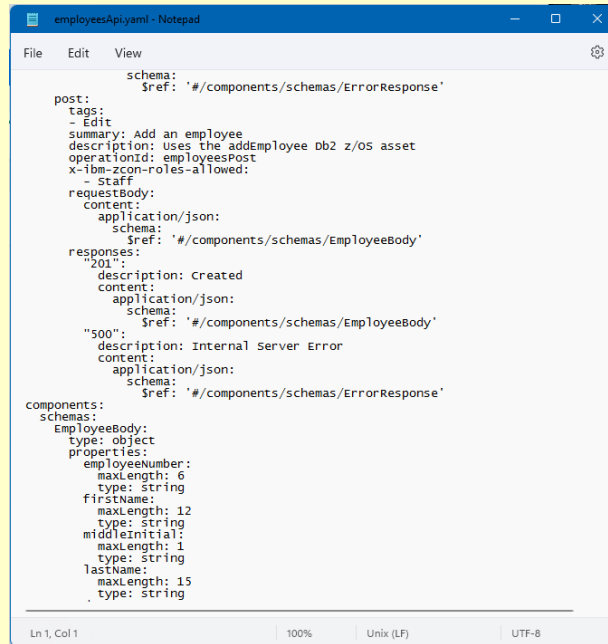
Body

* employeeNumber	abc		
* firstName	abc		
* middleInitial	abc		
* lastName	abc		
* department	abc		
* phoneNumber	abc		

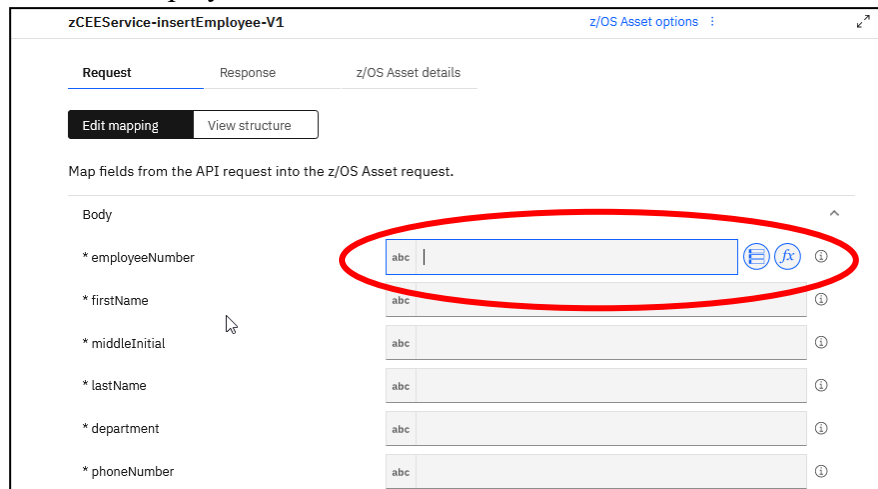
**Tech-Tip:** The names on the left-hand side are derived from the host-variables names specified in the VALUES clause of the INSERT statement (on in the “AS” attribute value in a SELECT statement).

```
INSERT INTO USER1.EMPLOYEE
(EMPNO, FIRSTNAME, MIDDLEINITIAL, 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)
```

While request message properties names on the right-hand side are derived from the YAML document description of the request message.



**Note:** It is very important that when working with mapping fields that the field has been properly selected. A properly selection field will be displayed in a blue box as shown below.

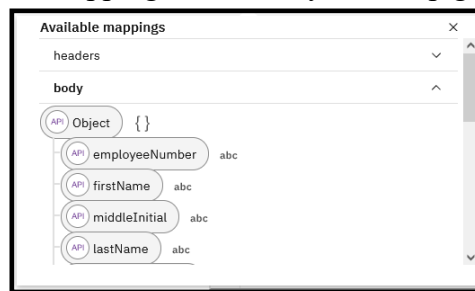




10. Now map the Db2 request message fields with the corresponding API request message fields. But first become familiar with the fields in the API request message for this method. To display the API's request message fields, select any container field, and then select the *Insert a mapping* tool (see below).

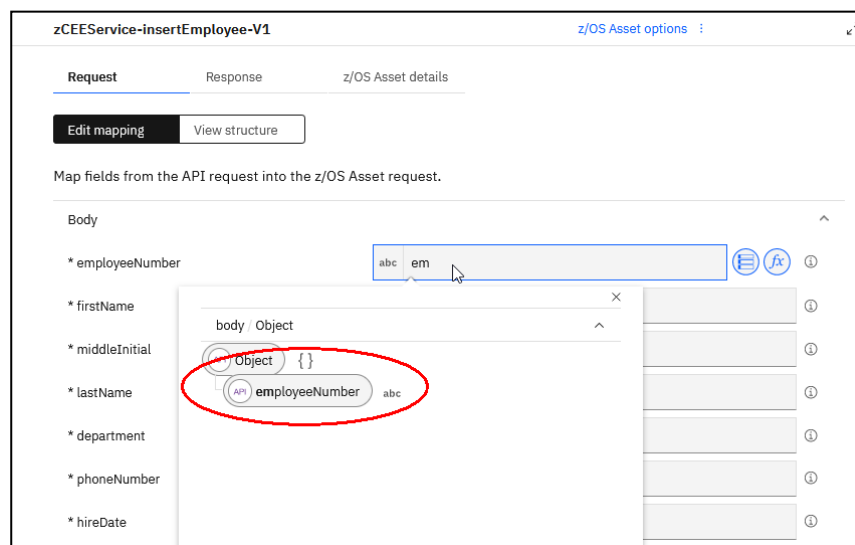


11. This will display the header and body mappings available for this method of the API. Become familiar with the mappings available in the body of the request message (you will have to use the scroll bar to see all the available mappings). Knowledge of the mappings in the body will help greatly in the next few steps.



*There are two ways to map the Db2 request schema with the corresponding specification document API request fields. Both will be demonstrated in this section of the exercise. Use which ever method you prefer when mapping fields later in this exercise.*

12. Start by selecting an empty Db2 request message field and start typing the corresponding API request message field name. For example, entering the string **em** in the area beside *employeeNumber* will eventually match a field in the API request message whose name includes the same characters, and that schema field will be displayed in the drop-down list (see below).



13. Select the field and the area beside the Db2 request schema property name this will cause the API's request message field name to populate the area and complete the mapping.

zCEEService-insertEmployee-V1 z/OS Asset options


Request Response z/OS Asset details

**Edit mapping** View structure

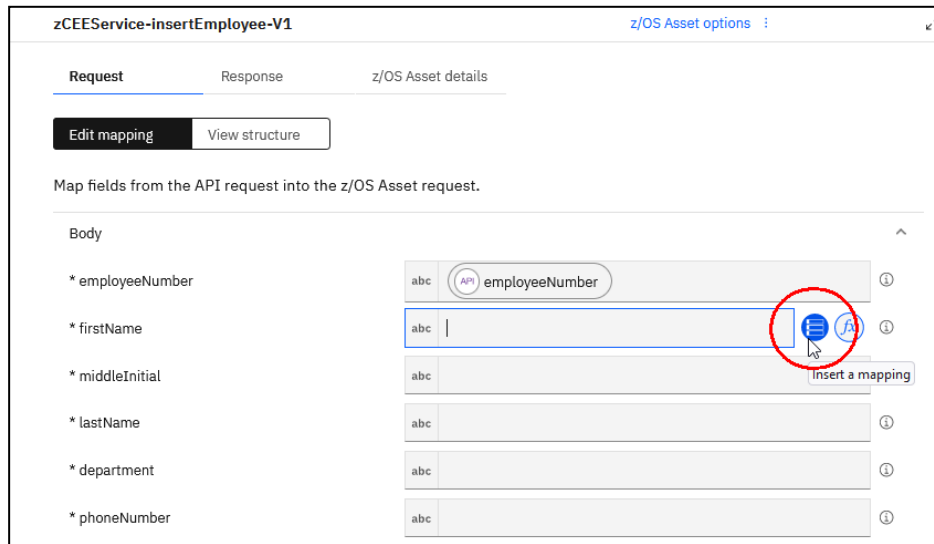
Map fields from the API request into the z/OS Asset request.

Body

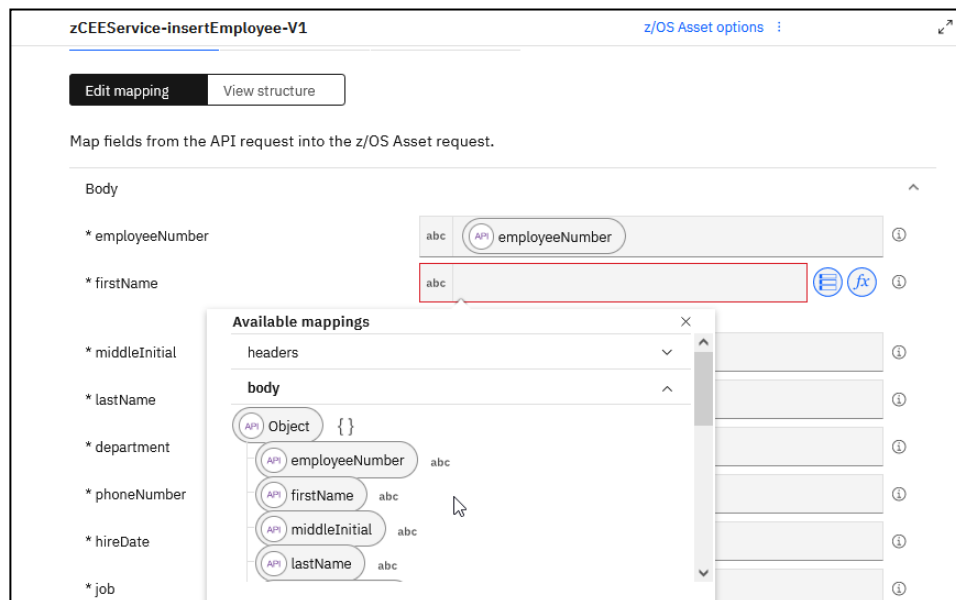
* employeeNumber	abc	<span>API</span> employeeNumber	①
* firstName	abc		①
* middleInitial	abc		①
* lastName	abc		①
* department	abc		①
* phoneNumber	abc		①

**Tech-Tip:** The icon  can be used to maximize or reset this area of the page.

14. The alternate mapping method is to select the *Insert a mapping* tool (see below).



15. This will display a list of available mapping fields. Since this is a request message and the fields are in the request *body*. Scroll up or down and choose appropriate field from the fields from the *body*, not a query parameter, nor a path parameter. In this case, the field to select is the *firstName* field.



16. Use either technique to complete the mappings. When completed, the results should look something like this the page below.

zCEEService-insertEmployee-V1
z/OS Asset options

Request
Response
z/OS Asset details

Edit mapping
View structure

Map fields from the API request into the z/OS Asset request.

Body			
* employeeNumber	abc	API employeeNumber	①
* firstName	abc	API firstName	①
* middleInitial	abc	API middleInitial	①
* lastName	abc	API lastName	①
* department	abc	API departmentCode	①
* phoneNumber	abc	API phoneNumber	①
* hireDate	abc	API dateOfHire	①
* job	abc	API job	①
* educationLevel	123	API educationLevel	①
* sex	abc	API sex	①
* birthDate	abc	API dateOfBirth	①
* salary	123	API salary	①
* bonus	123	API lastBonus	①
* commission	123	API lastCommission	①

17. The next step is to provide the configuration to evaluate the responses that come back in the Db2 REST service response message. Select the *Responses* box in the *view operation properties* page.

The screenshot displays the 'View operation properties' page in the IBM z/OS Connect API Designer. The top navigation bar shows the path `Paths / /employees / POST` and includes 'Saved' and 'Test' buttons. The main workflow area shows a sequence of steps: 'API Request', 'DB2 zCEEService-insertEmploy...' (with the description 'Insert an employee into table U...'), and a 'Responses' step which is highlighted with a red oval. Below the workflow, the 'Responses' configuration panel is open, showing instructions: 'Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.'

Two response rules are listed:

- 200 - OK**: If .. equals .. then send 200 response. The rule combination is 'All the following are true'. The configuration table below shows a single condition:
 

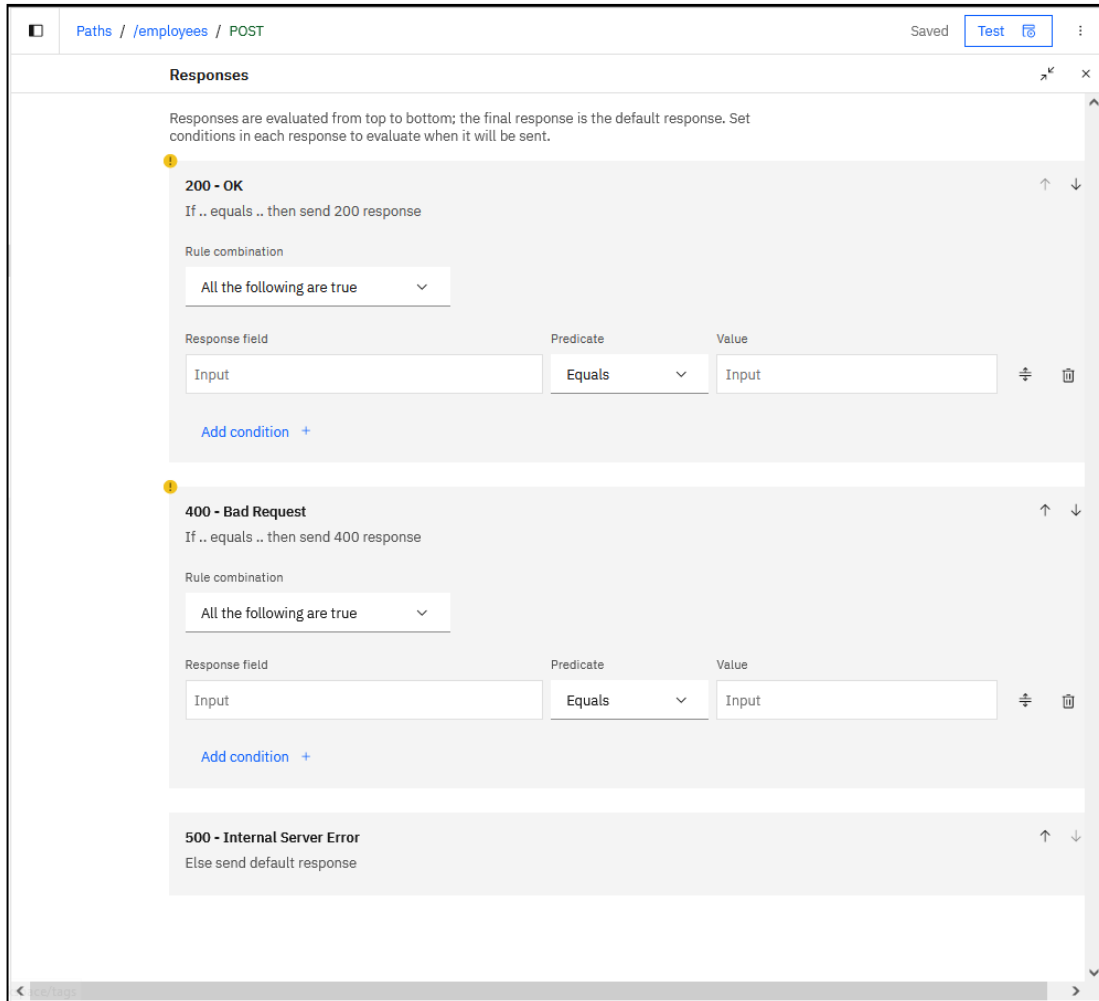
Response field	Predicate	Value
Input	Equals	Input
- 400 - Bad Request**: If .. equals .. then send 400 response. The rule combination field is currently empty.

At the bottom left, the 'space/overview' tab is visible.

18. Maximize the *Responses* area of the browser's page (see below).

Responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record was inserted successfully. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was inserted or not. So, we need another way to determine whether a row was really inserted. Fortunately, a Db2 REST service returns another response field, *Update Count*, which we can check the value to see how many rows were affected by this request.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) the value of *Update Count* is set to either 1 or zero.



19. Under the **200 – OK** response, Enter the string **Stat** in the **Input** area under **Response field**. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the **StatusCode** field. Leave the **Predicate** as **Equals** and enter **200** in the **Input** field for **Value**.

200 - OK  
If .. equals .. then send 200 response

Rule combination  
All the following are true

Response field: Stat  
Predicate: Equals  
Value: Input

zosAssetResponse / Object

- Object {}
- statusCode 123
- body {}
- StatusDescription abc
- StatusCode 123

20. Next add a condition check for the value of **Update Count** by clicking on **Add condition** in the **200 – OK** evaluation.

Paths /employees / POST

Responses

Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.

200 - OK  
If StatusCode equals 200 then send 200 response

Rule combination  
All the following are true

Response field: StatusCode  
Predicate: Equals  
Value: 200

Add condition +

21. Use the same technique described above to add a check for response field **Update Count**. Leave the **Predicate** as **Equals** and set the value to **1** below:

Paths /employees / POST

Responses

Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.

200 - OK  
If StatusCode equals 200 then send 200 response

Rule combination  
All the following are true

Response field: StatusCode  
Predicate: Equals  
Value: 200

Response field: Update Count  
Predicate: Equals  
Value: 1

Add condition +

22. For the *400 – Bad Request* check, add a check for **Status Code** equaling **200** and a check for Update Count equaling **0**

**400 - Bad Request**  
If StatusCode equals 200 and "Update Count" equals 0 then send 400 response

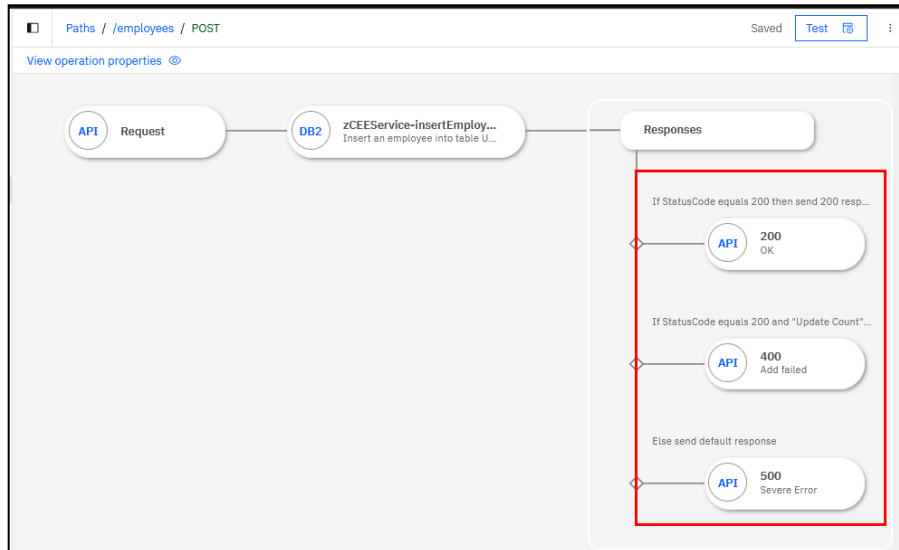
Rule combination  
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Equals	0

Add condition +

23. If neither of these connections are met, simply return with a HTTP 500 status code.

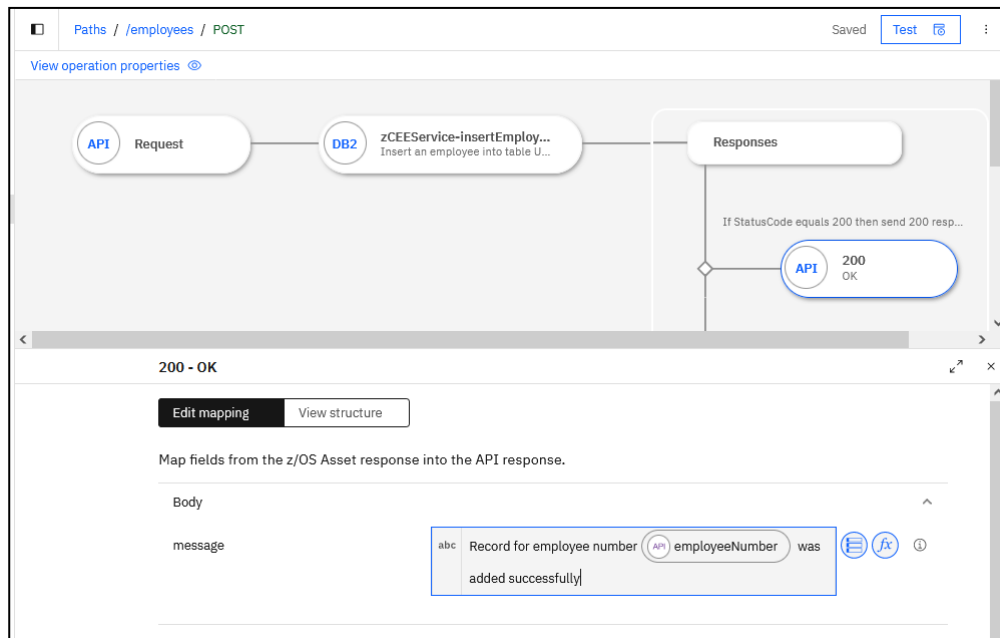
24. Next the API response messages need to be configured for each of these potential status codes.





25. Select the response for *200 OK* paste the text below in the *message* area.

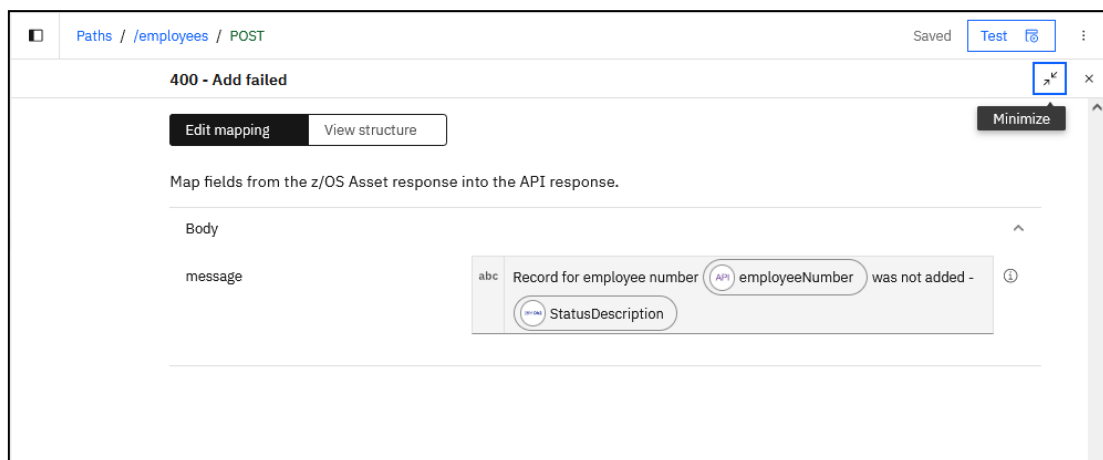
**Record for employee number {{\$apiRequest.body.employeeNumber}} was added successfully**



The same techniques used to map API response with the Db2 REST request message can be used to insert Db2 REST response files into text like this message which is then subsequently mapped to a field in the API response message. There is flexibility in building complex text strings based on the fields in the Db2 REST response message.

26. Select the response for *400 Add failed* response mapping and paste the text below in the *message* area.

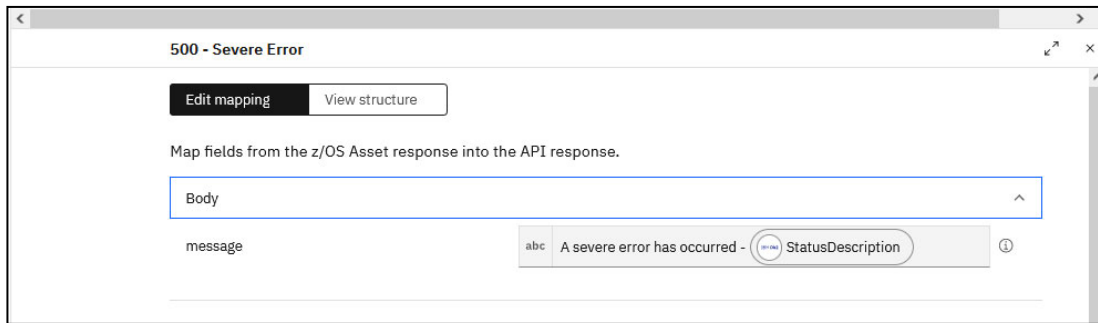
**Record for employee number {{\$apiRequest.body.employeeNumber}} was not added - {{\$zosAssetResponse.body.StatusDescription}}**



**Tech-Tip:** Db2 REST response property field *StatusDescription* provides more information regarding the issue that caused the insert to fail.

27. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

***A severe error has occurred - {{\$zosAssetResponse.body.StatusDescription}}***

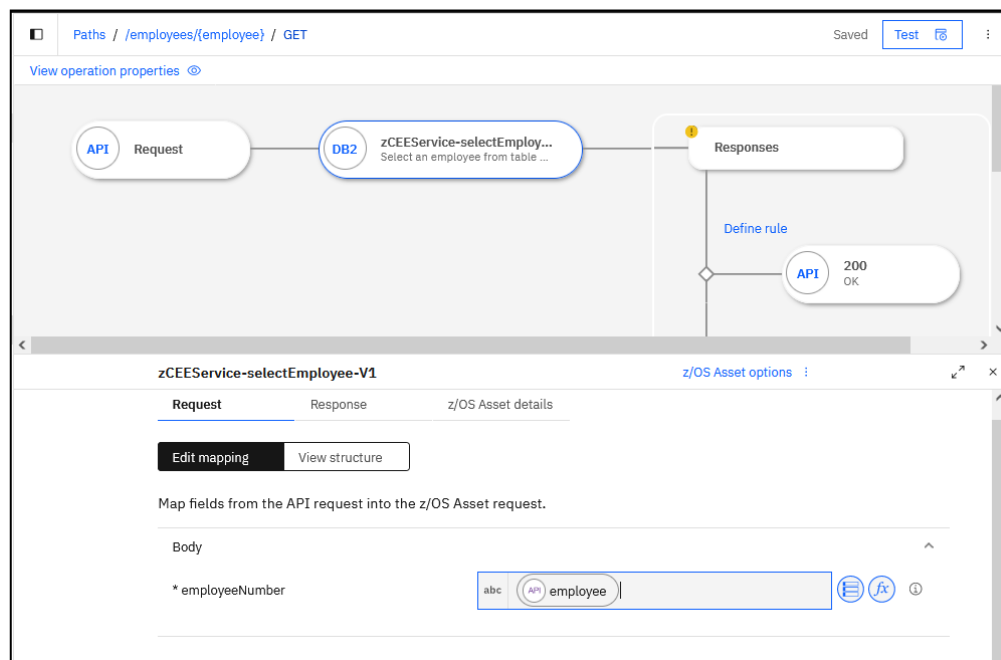


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

### ***Configure the GET method for URI path /employees/{employee}***

Now let's repeat the process and complete the configuration for the *GET* method of URI Path */employees/{employee}*

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-selectEmployee* to this method by using the same steps as performed before. Map the path parameter *employee* to the Db2 REST service request message field *employeeNumber* as shown below.



## 2. Maximize the *Responses* area of the browser's page (see below).

Again, responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record a row or rows were returned as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was selected or not. So, we need another way to determine whether a row was really selected. In this case a Db2 REST service will return the rows selected in a list or array. We are going to take advantage of function that will return the number of elements in the list or array, e.g., \$count. If the result of invoking the function against a list returns a non-zero values, the list or array contains elements. If the result is zero, no elements are in the list and therefore no rows were selected.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and the value of invoking the \$count function against the list of returned rows.

The screenshot shows the 'Responses' tab in the API Designer. The breadcrumb path is 'Paths / /employees/{employee}/ GET'. There are 'Saved' and 'Test' buttons. The 'Responses' section contains three rules:

- 200 - OK**: If .. equals .. then send 200 response. Rule combination: All the following are true. Response field: Input. Predicate: Equals. Value: Input.
- 404 - Not Found**: If .. equals .. then send 404 response. Rule combination: All the following are true. Response field: Input. Predicate: Equals. Value: Input.
- 500 - Internal Server Error**: Else send default response.

Under the **200 – OK** response, Enter the string **Stat** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter **200** in the *Input* field for *Value*.

Next add a condition check for the value of invoking the function \$count against the array of rows returned by the Db2 REST service *Count* by clicking on Add condition in the **200 – OK** evaluation and entering the string below in the area for the new check of a Response field.

***\$count(\$zosAssetResponse.body."ResultSet Output")***

And set the *Predicate* to *Is greater than or equal to* a *Value* of **1**.

**200 - OK**  
If StatusCode equals 200 and \$count("ResultSet Output") is greater than or equal to 1 then send 200 response

Rule combination  
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
\$count(\$zosAssetResponse.body."ResultSet Output")	Is greater th...	1

Add condition +

3. For the **404 – Not Found** check, add a check for **StatusCode** equaling **200** and a check for the count equaling zero.

**404 - Not Found**  
If StatusCode equals 200 then send 404 response

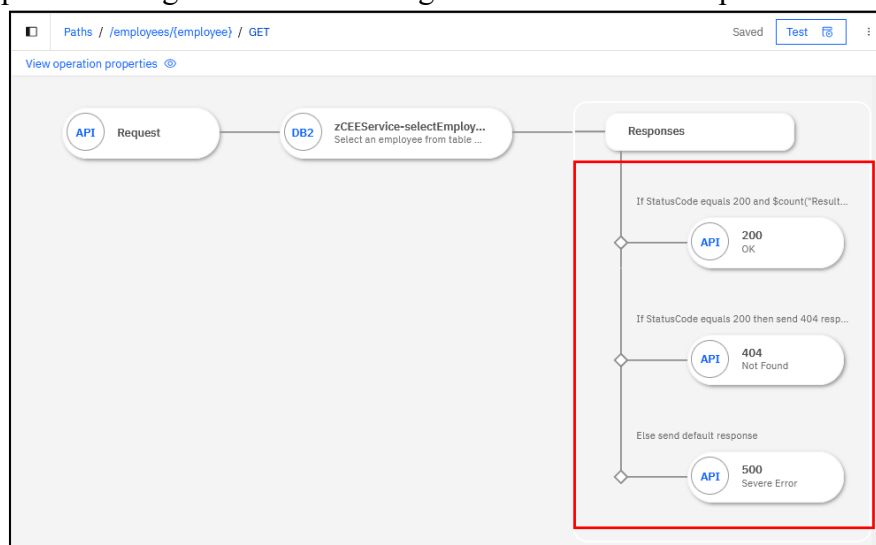
Rule combination  
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
count(\$zosAssetResponse.body."ResultSet Output")	Equals	0

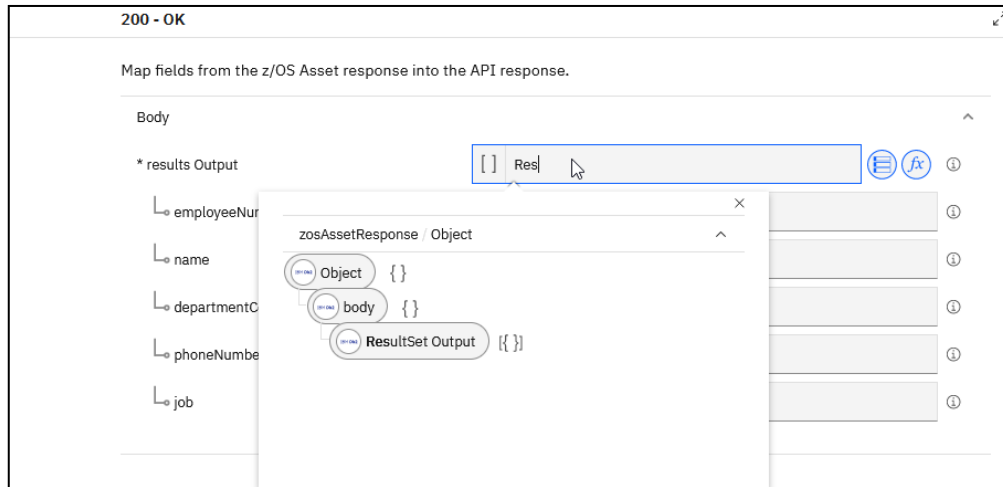
Add condition +

4. If neither of these connections are met, simply return with a HTTP 500 status code.

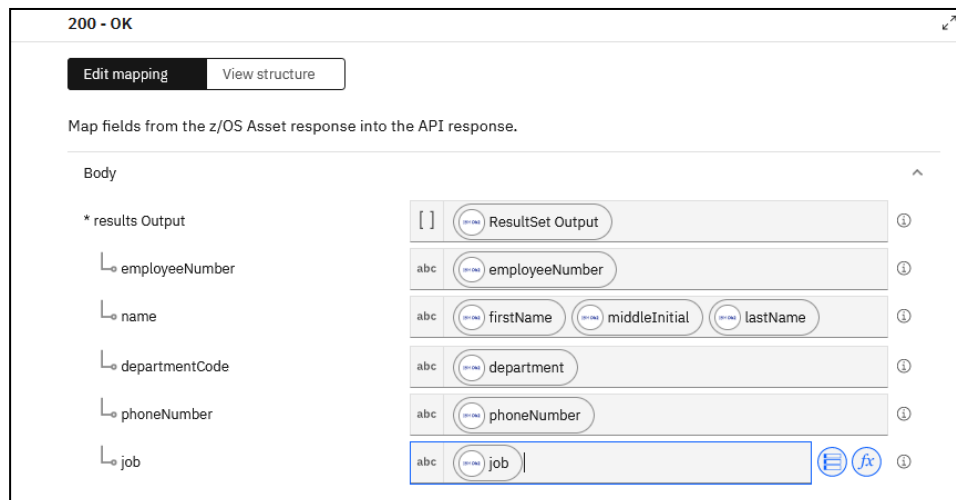
5. Next the API response messages need to be configured for each of these potential status codes.



6. Select the response for *200 OK* and map the fields from the Db2 REST response message. Start by mapping the *ResultSet Output* field from the Db2 REST response message to the API response field *results Output*. This must be done first to be able to access the elements in the array.

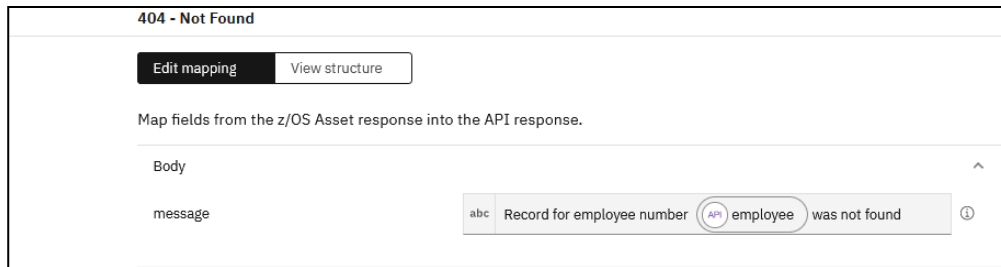


7. Be careful at this point to ensure you are selecting fields in the *ResultSet output* array in the *body* of the *zosAssetResponse*. The same property name may appear in another one of the available mappings, e.g., *apiRequest*, *ResultSet Row item*, etc. and if a property is selected from one these mappings, the results will be unpredictable.
8. Complete the mapping for the other properties. Notice the mapping of the Db2 REST response properties *firstName*, *middleInitial* and *lastName* into the single API response property *name*.



9. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

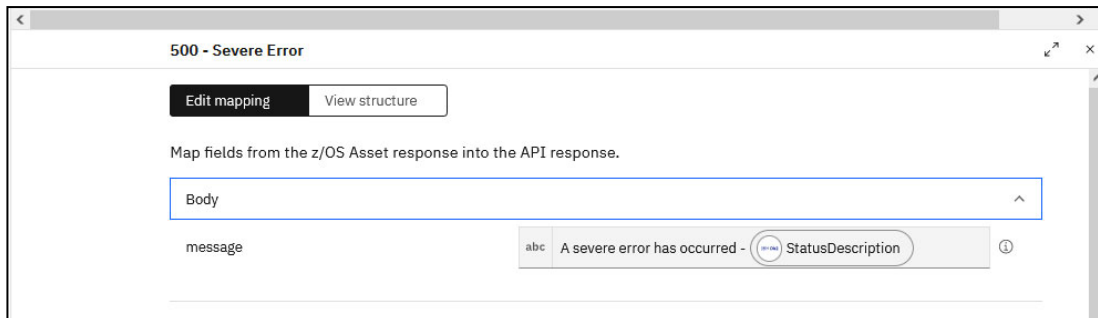
***Record for employee number {{ \$apiRequest.pathParameters.employee }} was not found***



Notice that the mapping for the property in the message was from the API request message and not the Db2 REST response message.

10. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

***A severe error has occurred - {{ \$zosAssetResponse.body.StatusDescription }}***



The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

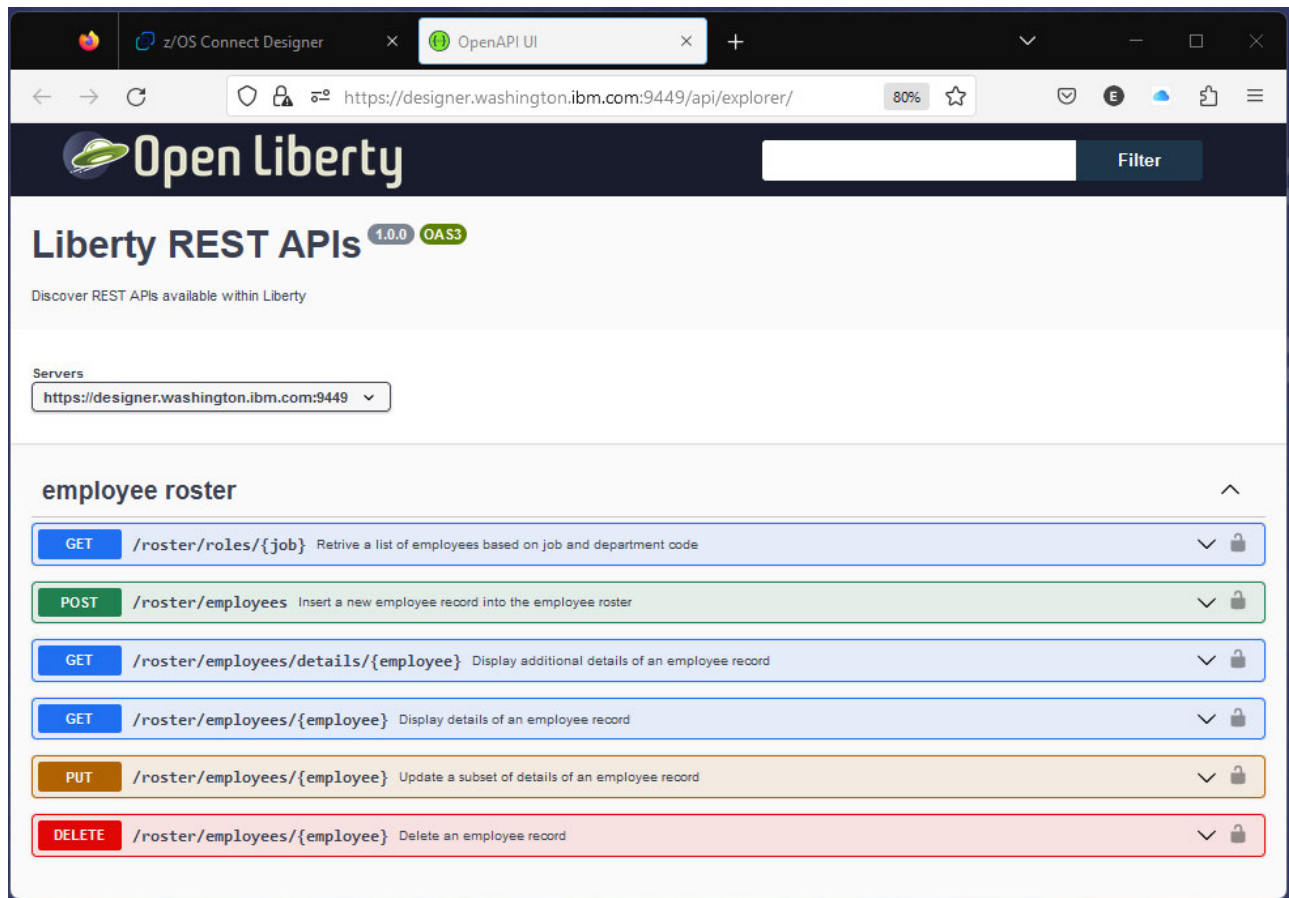
**Tech-Tip:** Note that the exclamation mark has disappeared from *zCEEService-insertEmployee* on the operation page.

## Testing the API's POST and GET methods

As the API was being developed, the changes have been saved and a Web Archive (WAR) file was generated with each change. If the upper right-hand corner of the browser page there will be a **Test** button. Clicking this button will open an API Explorer page. All the URI paths and methods in the original OpenAPI 3 specification document will be displayed, but only the *POST* for */employees* and the *GET* for */employees/{employee}* have been created. Executing one of the other methods will return an HTTP 404 because the components required to execute these methods cannot be found in the WAR.

Let's test what has been developed so far.

1. Click the **Test** button to open the API Explorer.



**Tech Tip:** You may be challenged by browser because the digital certificate used by the *Designer* 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.

2. Use the pull-down arrow in the **Servers** box at the top of the page and select

<https://designer.washington.ibm.com:9449>

3. Click on **Post /roster/employees** URI path to display the request body view of the URI path.

**employee roster**

**GET** /roster/roles/{job} Retrieve a list of employees based on job and department code

**POST** /roster/employees Insert a new employee record into the employee roster

Insert a new employee record into the employee roster (insertEmployee Db2 z/OS REST service)

**Parameters** Try it out

Name	Description
Authorization	Authorization

string (header)

**Request body** *required* application/json

request body

Example Value | Schema

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "str",
  "phoneNumber": "stri",
  "dateOfHire": "stringst",
  "job": "string",
  "educationLevel": 32767,
  "sex": "s",
  "dateOfBirth": "stringst",
  "salary": 9999999.99,
  "lastBonus": 9999999.99,
  "lastCommission": 9999999.99
}
```

4. Next press the **Try it out** button to enable the entry of an authorization string and a request message body

**Parameters** Cancel

Name	Description
Authorization	Authorization

string (header)

**Request body** *required* application/json

request body

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "str",
  "phoneNumber": "stri",
  "dateOfHire": "stringst",
  "job": "string",
  "educationLevel": 32767,
  "sex": "s",
  "dateOfBirth": "stringst",
  "salary": 9999999.99,
  "lastBonus": 9999999.99,
  "lastCommission": 9999999.99
}
```

**Servers**

These path-level options override the global server options.

/

**Execute**



5. Enter the JSON request message below in the *Request body* section and press the **Execute** button.

```
{
  "employeeNumber": "948478",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

6. Security was enabled in the original specification document, so you will be required to sign in with one of the identities defined in the basicSecurity.xml file explored earlier. Use **Fred** for the *Username* and **fredpwd** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

7. Scroll down the view and you should see the *Response body* with the expected successful message.

The screenshot displays an API testing tool interface with the following sections:

- Responses**: A header for the response section.
- Curl**: A text area containing the curl command for a POST request to `https://designer.ibm.com:9449/roster/employees` with a JSON body.
- Request URL**: A text field showing the URL `https://designer.ibm.com:9449/roster/employees`.
- Server response**: A section showing the response details.
  - Code**: 200
  - Details**: A tab for viewing the response body.
  - Response body**: A text area showing the JSON response: `{ "message": "Record for employee number 948478 was added successfully" }`. There are icons for copying and downloading the response body.
  - Response headers**: A section for viewing response headers, currently empty.

8. Press the **Execute** button again and observe the results. A row for this employee number already existed in the employee roster (a Db2 tables) so the request failed with an HTTP 500.

Responses

Curl

```
curl -X 'POST' \
  'https://designer.ibm.com:9449/roster/employees' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "employeeNumber": "948478",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
  }'
```

Request URL

https://designer.ibm.com:9449/roster/employees

Server response

Code	Details
500	Error: Internal Server Error

Response body

```
{
  "message": "A severe error has occurred - Service zCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-803, SQLSTATE=23505, Message=AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE EMP1LYCA CONSTRAINS COLUMNS OF THE TABLE EMPLOYEE. ONLY ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS. RID OF EXISTING ROW IS X'0000000225'. Error Location:DSNL3XUS:
}
```

Response headers

9. Scroll down and click on *GET /roster/employees/{employees}* URI path to display the request body view of the URI path for this method. Next click on the **Try it out** button to enable the entry of data for this method. Enter **948478** as the employee identity and press the **Execute** button to retrieve a subset of data for this employee.

Responses

Curl

```
curl -X 'GET' \
  'https://designer.ibm.com:9449/roster/employees/948478' \
  -H 'accept: application/json'
```

Request URL

https://designer.ibm.com:9449/roster/employees/948478

Server response

Code	Details
200	

Response body

```
{
  "results Output": [
    {
      "employeeNumber": "948478",
      "name": "Matt T Johnson",
      "departmentCode": "C00",
      "phoneNumber": "0065",
      "job": "Staff"
    }
  ]
}
```

Download

10. Try this again using number **121212** and observe the results. You see the message that the employee was not found.

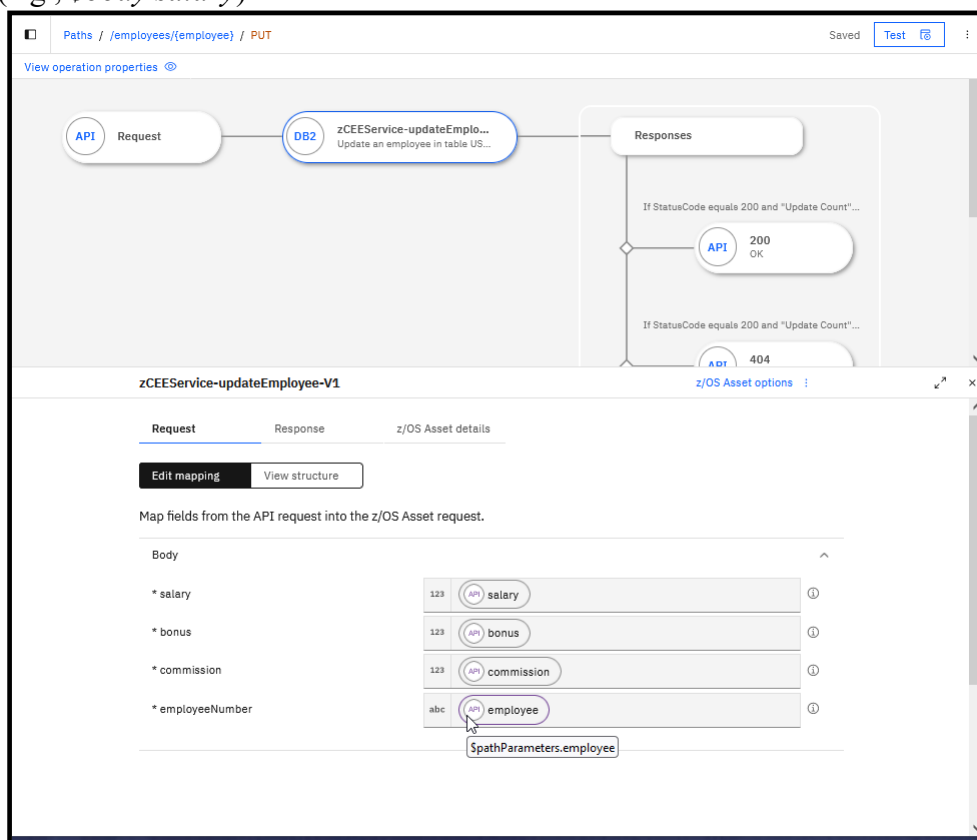
## Compete the configuration of the API (Optional)

To be able to fully test all the URI paths and methods the other methods need to be configured. Otherwise, you may advance to the section *Deploying and Installing APIs in a z/OS Connect Native Server*.

### Configure the PUT method for URI path /employees/{employee}

Now add support for updated a subset of the details of an employee record by completing the configuration for the *PUT* method of URI Path /employees/{employee}

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-updatetEmployee* to this method. Map the API request path parameter field *employee* (*\$pathParameters.employee*) to the DB2 REST request message field *employeeNumber* as shown below. Map *salary*, *bonus*, and *commission* fields from the body of the API request message (e.g., *\$body.salary*)



**Tech Tip:** Hover each of the properties and you see that all the properties except *employee* are mapped to the Db2 REST service from the *body* of the API request message. Property *employee* is mapped to the Db2 REST service from the path parameter.

## 2. Maximize the *Responses* area of the browser's page (see below).

Responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record was updated as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was updated or not. So, we need another way to determine whether a row was really updated. Again, we will use the *Update Count* response field to check the value to see how many rows were affected by this request.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of *Update Count*.

The screenshot shows the 'Responses' configuration area for the API endpoint `/employees/{employee}` with the `PUT` method. The interface includes a 'Saved' button and a 'Test' button. Below the header, a note states: 'Responses are evaluated from top to bottom; the final response is the default response. Set conditions in each response to evaluate when it will be sent.'

Three response rules are listed:

- 200 - OK**: If .. equals .. then send 200 response. Rule combination: 'All the following are true'. Response field: 'Input'. Predicate: 'Equals'. Value: 'Input'.
- 404 - Not Found**: If .. equals .. then send 404 response. Rule combination: 'All the following are true'. Response field: 'Input'. Predicate: 'Equals'. Value: 'Input'.
- 500 - Internal Server Error**: Else send default response.

Each rule has an 'Add condition +' button and up/down arrows for reordering. The bottom of the window shows the file path: `kspace/assets/zCEEService-deleteEmployee-V1`.

3. Under the **200 – OK** response, Enter the string **Stat** in the **Input** area under **Response field**. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, as long as the string entered matches any portion of the field name, that field will be displayed). In this case, select the **StatusCode** field. Leave the **Predicate** as **Equals** and enter **200** in the **Input** field for **Value**.

Next add a condition check for the value of the **Update Count** Db2 REST response property by clicking on the **Add condition** in the **200 – OK** evaluation and entering the string below in the area for the new check of a **Response field**. Enter property **Update Count** for the **Response field** name. Set the **Predicate** to **Is greater than or equal to** and a value of **1**.

**200 - OK**

If StatusCode equals 200 and "Update Count" is greater than or equal to 1 then send 200 response

Rule combination

All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Is greater th...	1

Add condition +

4. For the **404 – Not Found** check, add a check for **StatusCode** equaling **200** and a check for **Update Count** equaling zero.

**404 - Not Found**

If StatusCode equals 200 and "Update Count" equals 0 then send 404 response

Rule combination

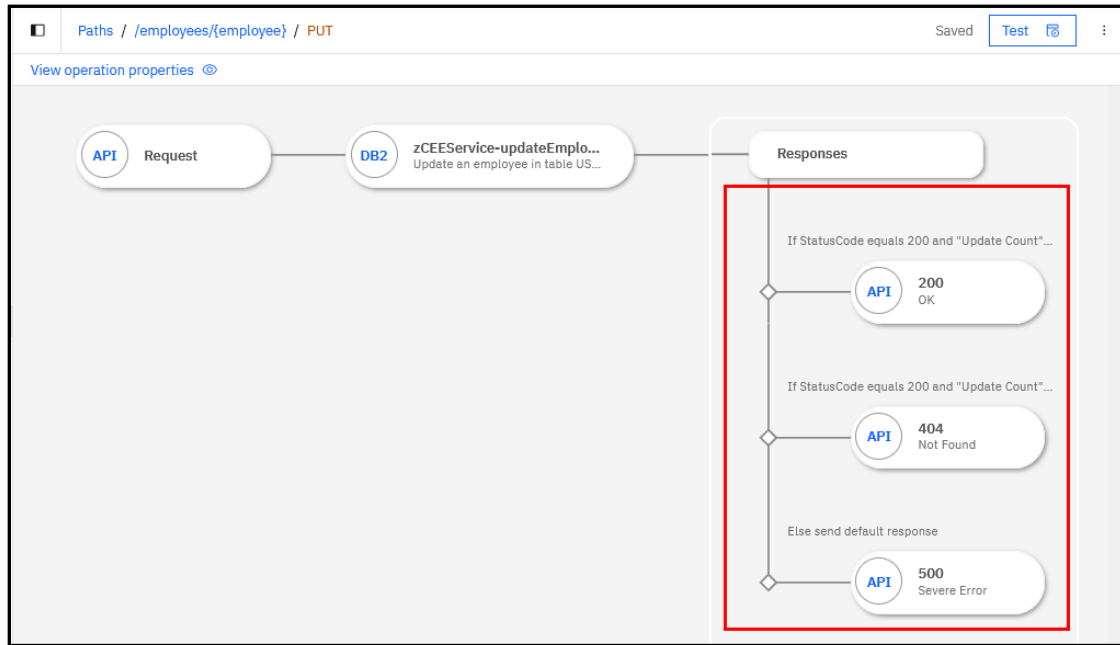
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Equals	0

Add condition +

5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



7. Select the response for *200 OK* paste the text below in the *message* area.

***Record for employee {{\$apiRequest.pathParameters.employee}} successfully updated***

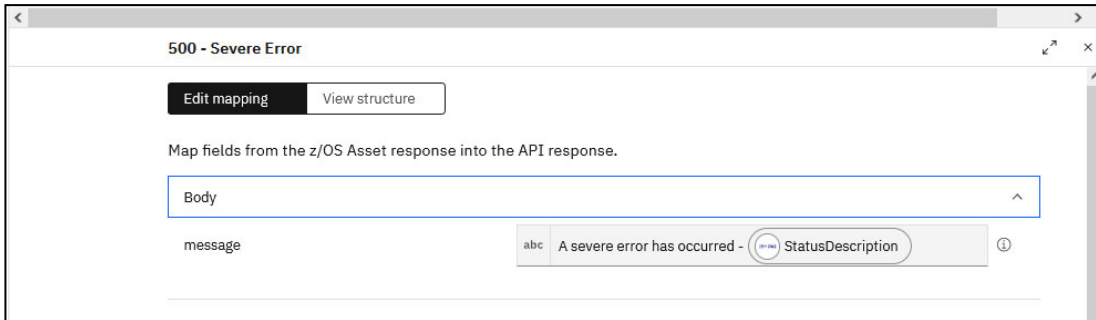
8. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

***Record for employee number {{\$apiRequest.pathParameters.employee}} was not found***

Notice that the mapping for the property in the message was from the API request message and not the Db2 REST response message.

9. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

***A severe error has occurred - {{\$zos.AssetResponse.body.StatusDescription}}***

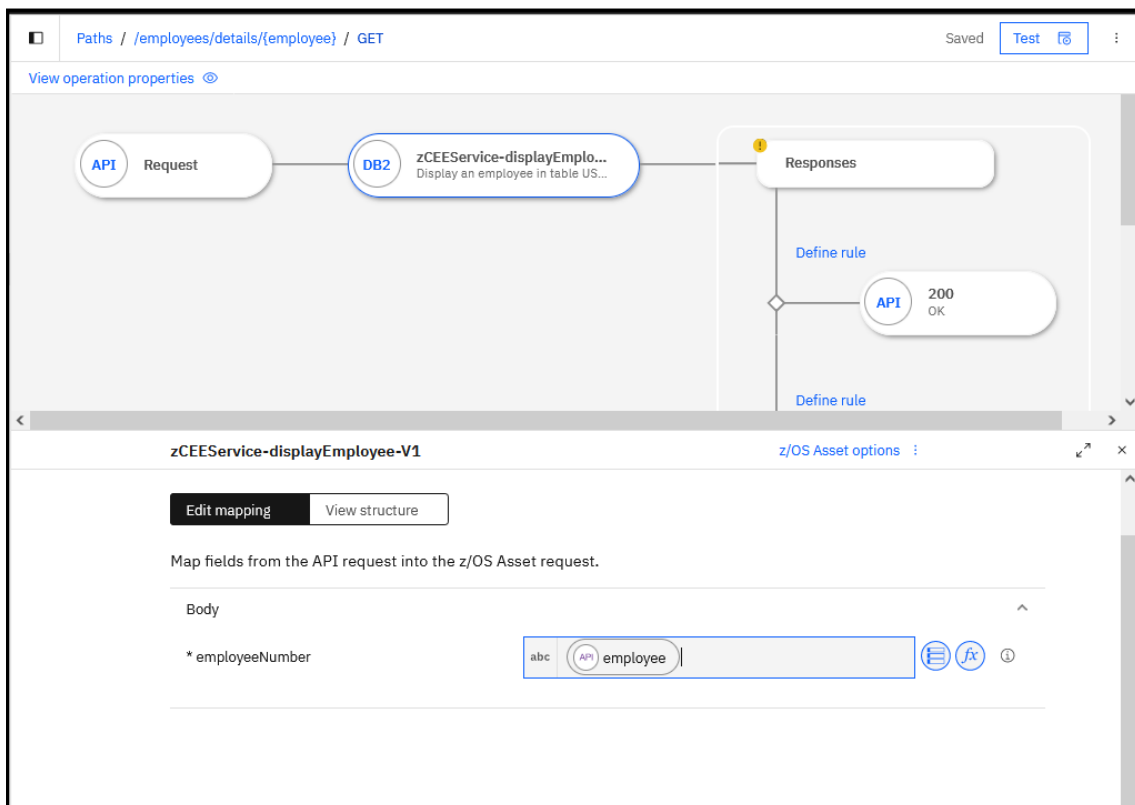


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

### ***Configure the GET method for URI path /employees/details/{employee}***

Now let's repeat the process and complete the configuration for the *GET* method of URI Path */employees/details/{employee}*

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-displayEmployee* to this method. Map the API request field *employee* to the DB2 REST request message field *employeeNumber* as shown below.



2. Maximize the *Responses* area of the browser's page (see below).

Again, responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record a row or rows were returned as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was selected or not. So, we need another way to determine whether a row was really selected. A Db2 REST service will return the rows selected in a list or array. We are going to take advantage of function that will return the number of elements in the list or array, e.g., \$count. If the result of invoking the function returns a non-zero values, the list or array contains elements. If the result is zero, no rows were selected.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of invoking the \$count function against the array of returned rows.

3. Under the *200 – OK* response, Enter the string *Stat* in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter *200* in the *Input* field for *Value*.

Next add a condition check for the value of invoking the function \$count against the array of rows returned by the Db2 REST service *Count* by clicking on Add condition in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field.

***\$count(\$zosAssetResponse.body."ResultSet Output")***

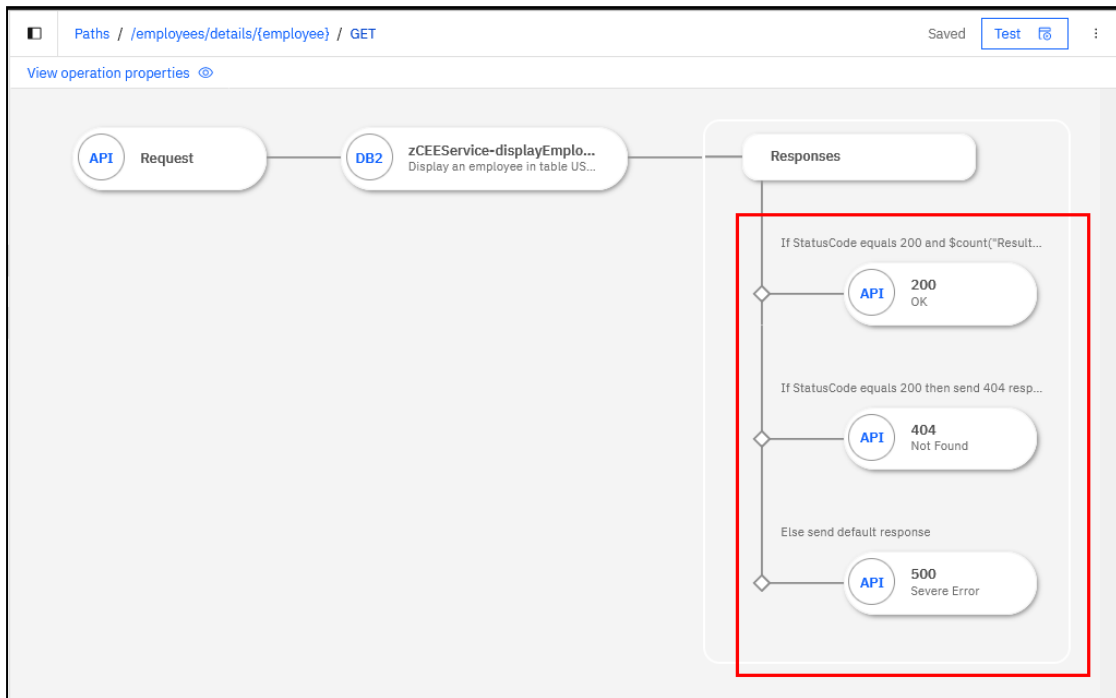
And set the *Predicate* to *Is greater than or equal to a Value of 1*.

4. For the *404 – Not Found* check, add a check for *StatusCode* equaling *200* and a check for the count equaling zero.

5. If neither of these connections are met, simply return with a HTTP 500 status code.



6. Next the API response messages need to be configured for each of these potential status codes.



7. Select the response for *200 OK* and map the fields from the Db2 REST response message. Start by mapping the *ResultSet Output* field from the Db2 REST response message to the API response field *results Output*. This must be done first to access the elements in the array.

The screenshot shows the '200 - OK' response configuration. The 'Body' section is expanded, showing a table mapping fields from the z/OS Asset response to the API response. The table has two columns: 'z/OS Asset response' and 'API response'.

z/OS Asset response	API response
* retrievedResults Output	[ ] <b>ResultSet Output</b>
employeeID	abc
name	abc
departmentCode	abc
phoneNumber	abc
dateOfHire	abc
job	abc
educationLevel	123
sex	abc
dateOfBirth	abc
annualSalary	123
lastBonus	123
lastCommision	123

8. Be careful at this point to ensure you are selecting fields in the *ResultSet output* array in the *body* of the *zosAssetResponse*. The same property name may appear in another one of the available mappings, e.g., *apiRequest*, *ResultSet Row item*, etc. and if a property is selected from one these mappings, the results will be invalid.
9. Complete the mapping for the other properties. Notice the mapping of the Db2 REST response properties *firstName*, *middleInitial* and *lastName* into the API response property *name*.

The screenshot shows the IBM z/OS Connect configuration interface for the path `/employees/details/{employee}` with a `GET` method. The status is `200 - OK`. The interface includes buttons for `Edit mapping` and `View structure`. Below these, a message states: "Map fields from the z/OS Asset response into the API response."

The `Body` section shows the mapping of fields from the z/OS Asset response into the API response. The API response structure is shown on the left, and the z/OS Asset response structure is shown on the right. The mapping is as follows:

API Response Property	z/OS Asset Response Property
* retrievedResults Output	ResultSet Output
employeeID	EMPNO
name	FIRSTNME, MIDINIT, LASTNAME
departmentCode	WORKDEPT
phoneNumber	PHONENO
dateOfHire	HIREDATE
job	JOB
educationLevel	EDLEVEL
sex	SEX
dateOfBirth	BIRTHDATE
annualSalary	SALARY
lastBonus	BONUS
lastCommision	COMM

10. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

***Record for employee number {{\$apiRequest.pathParameters.employee}} was not found***

The screenshot shows the configuration for a 404 - Not Found response. At the top, there are buttons for 'Edit mapping' and 'View structure'. Below, a text area contains the message: 'Record for employee number [API employee] was not found'. The 'API' label is circled, indicating the source of the mapped property 'employee'.

Notice that the mapping for the property in the message was from the API request message and not the Db2 REST service response message.

11. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

***A severe error has occurred - {{\$zosAssetResponse.body.StatusDescription}}***

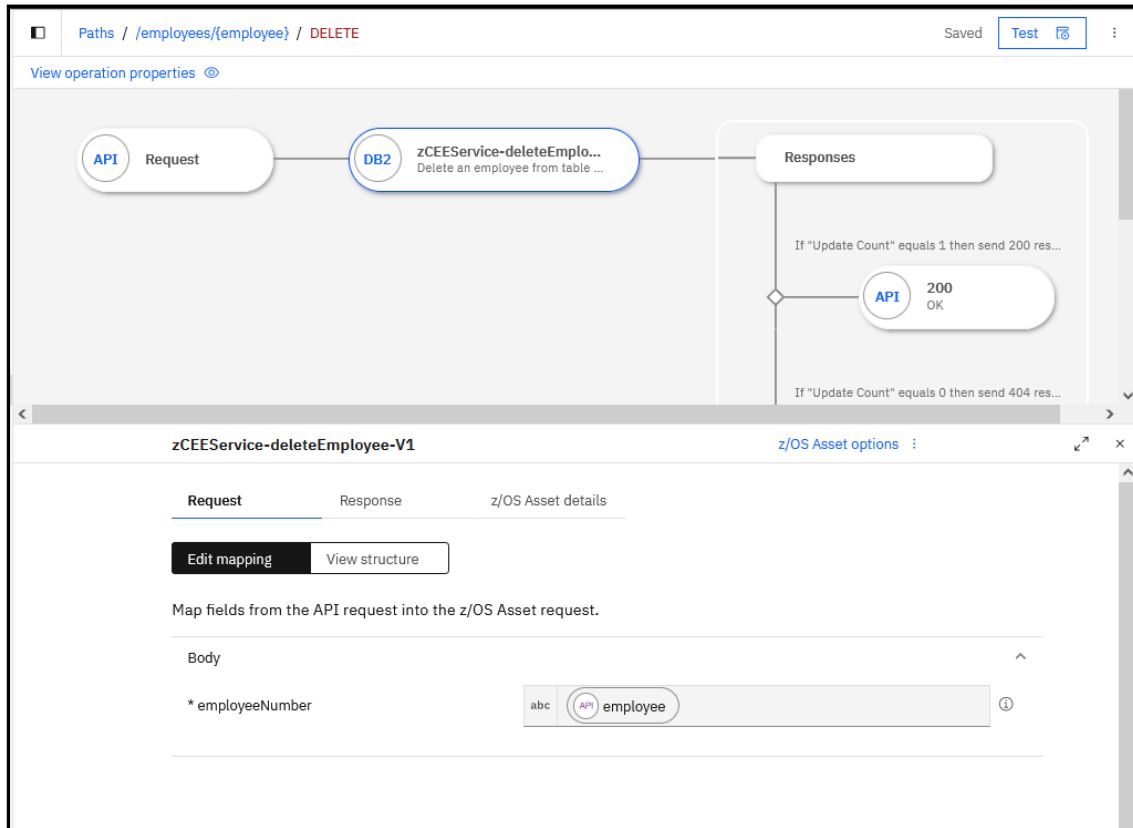
The screenshot shows the configuration for a 500 - Severe Error response. At the top, there are buttons for 'Edit mapping' and 'View structure'. Below, a text area contains the message: 'A severe error has occurred - [zosAssetResponse StatusDescription]'. The 'zosAssetResponse' label is circled, indicating the source of the mapped property 'StatusDescription'.

The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

## Configure the DELETE method for URI path /employees/{employee}

Now let's repeat the process and complete the configuration for the *DELETE* method of URI Path /employees/{employee}

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-deleteEmployee* to this method. Map the API request field *employee* to the DB2 REST request message field *employeeNumber* as shown below.



2. Maximize the *Responses* area of the browser's page (see below).

Responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record was updated as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was updated or not. So, we need another indication whether a row was really updated. Again, we will use the *Update Count* response field to check the value to see how many rows were affected by this request.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of *Update Count*.

3. Under the *200 – OK* response, Enter the string **Stat** in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter **200** in the *Input* field for *Value*.

Next add a condition check for the value of the *Update Count* Db2 REST response property by clicking on the *Add condition* in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field. Enter property **Update Count** for the Response field name. Set the *Predicate* to *Is greater than or equal to* and a value of **1**.

**200 - OK**

If StatusCode equals 200 and "Update Count" is greater than or equal to 1 then send 200 response

Rule combination

All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Is greater th...	1

Add condition +

4. For the *404 – Not Found* check, add a check for **Status Code** equaling **200** and a check for an update count equaling zero.

**404 - Not Found**

If StatusCode equals 200 and "Update Count" equals 0 then send 404 response

Rule combination

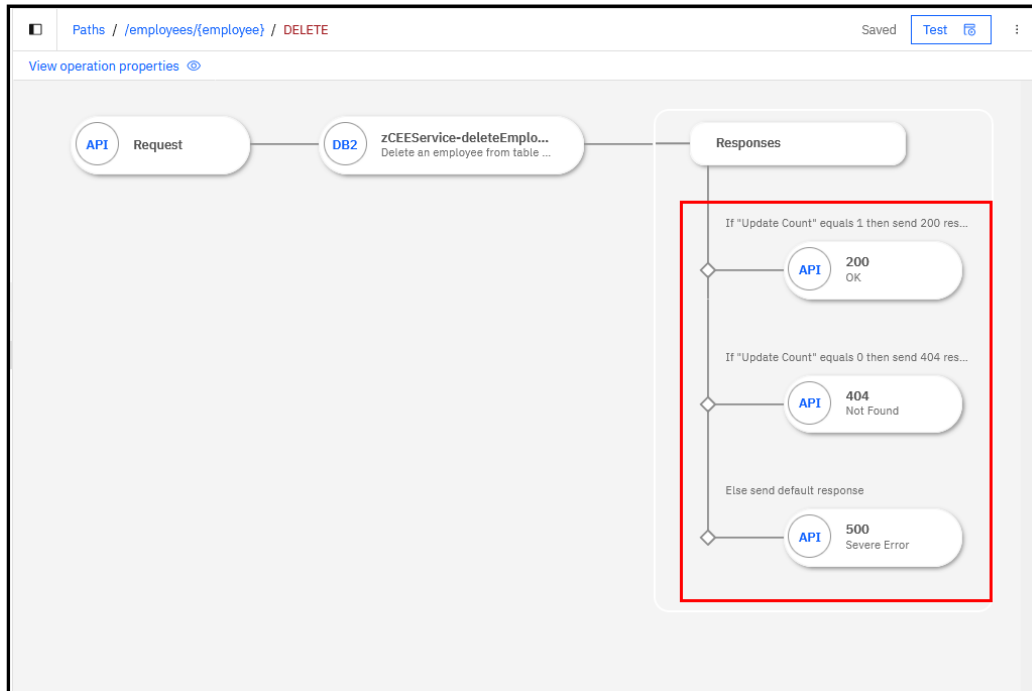
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Equals	0

Add condition +

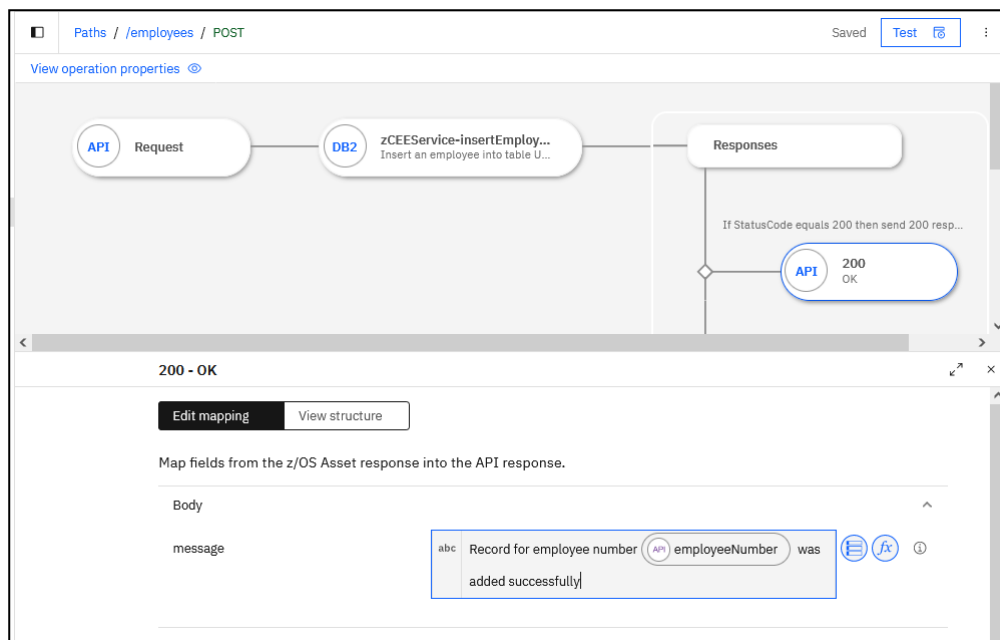
5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



28. Select the response for 200 OK paste the text below in the *message* area.

**Record for employee number {{\$apiRequest.body.employeeNumber}} was deleted successfully**



12. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

***Record for employee number {{\$apiRequest.pathParameters.employee}} was not found***

404 - Not Found

Edit mapping View structure

Map fields from the z/OS Asset response into the API response.

Body

message abc Record for employee number [API employee] was not found ⓘ

Notice that the mapping for the property in the message was from the API request message and not the Db2 REST response message.

13. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

***A severe error has occurred - {{\$zosAssetResponse.body.StatusDescription}}***

500 - Severe Error

Edit mapping View structure

Map fields from the z/OS Asset response into the API response.

Body

message abc A severe error has occurred - [StatusDescription] ⓘ

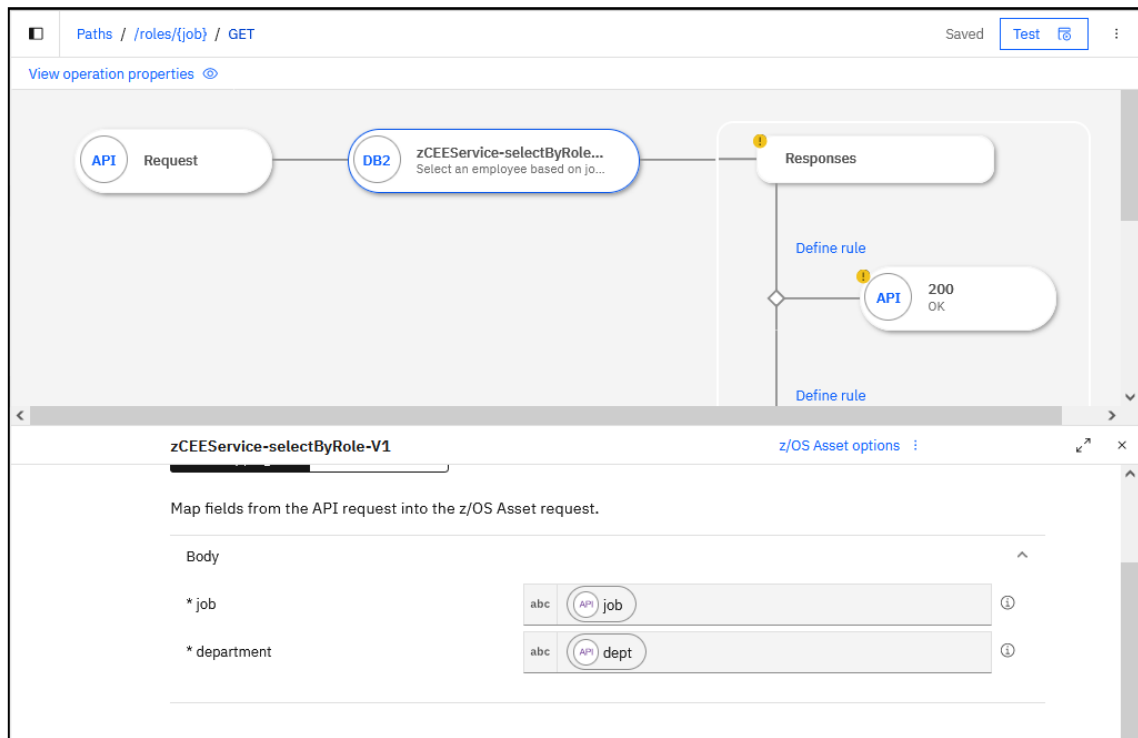
**Tech-Tip:** Db2 REST response property field *StatusDescription* provides more information regarding the issue that caused the insert to fail.

The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

## Configure the GET method for URI path /roles/{job}

Now complete the configuration for the *GET* method of URI Path */roles/{job}*. This method includes both a path parameter and a query parameter.

1. Start by adding a new z/OS Asset for Db2 REST service *zCEEService-selectByRole* to this method. Map the API request path parameter *job* to the DB2 REST server request message field *job* and the API query parameter *department* to the Db2 REST service request message field *dept*.





## 2. Maximize the *Responses* area of the browser's page (see below).

Again, responses from the Db2 REST service are evaluated in the order shown in the sequence shown. The first check is to see if the record a row or rows were returned as intended. Db2 REST services will return an HTTP status code of 200 if the Db2 REST service was able to complete regardless of whether a row was selected or not. So, we need another indication whether a row was really selected. A Db2 REST service will return the rows selected in a list or array. We are going to take advantage of function that will return the number of elements in the list or array, e.g., \$count. If the result of invoking the function returns a non-zero values, the list or array contains elements. If the result is zero, no rows were selected.

So, we are going to check the response fields to (1) confirm the HTTP status code from Db2 is 200 and (2) and for the value of invoking the \$count function against the array of returned rows.

## 3. Under the *200 – OK* response, Enter the string *Stat* in the *Input* area under *Response field*. This will display all the fields in the Db2 REST response which match this string (position of the string in the field name does not matter, if the entered string matches any portion of the field name, that field will be displayed). In this case, select the *StatusCode* field. Leave the *Predicate* as *Equals* and enter *200* in the *Input* field for *Value*.

Next add a condition check for the value of invoking the function \$count against the array of rows returned by the Db2 REST service *Count* by clicking on Add condition in the *200 – OK* evaluation and entering the string below in the area for the new check of a Response field.

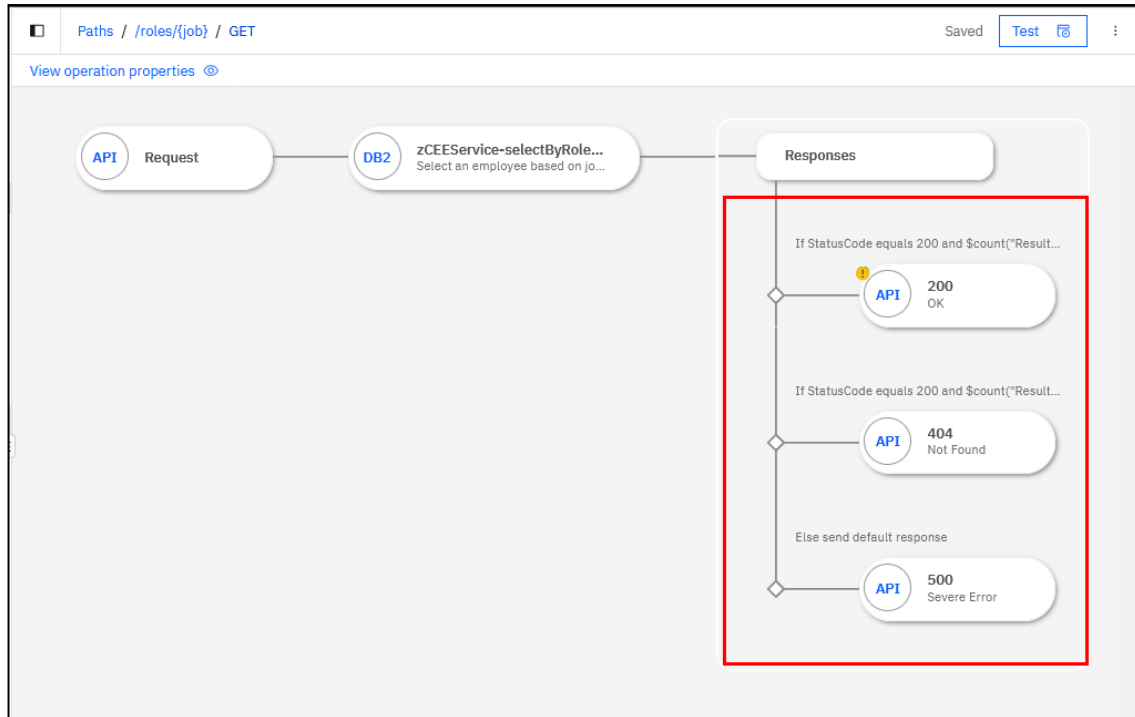
***\$count(\$zosAssetResponse.body."ResultSet Output")***

And set the *Predicate* to *Is greater than or equal to a Value of 1*.

## 4. For the *404 – Not Found* check, add a check for *StatusCode* equaling *200* and a check for count equaling zero.

## 5. If neither of these connections are met, simply return with a HTTP 500 status code.

6. Next the API response messages need to be configured for each of these potential status codes.



7. Select the response for *200 OK* and map the fields from the Db2 REST response message. Start by mapping the *ResultSet Output* field from the Db2 REST response message to the API response field *results Output*. This must be done first to be able to access the elements in the array.

**200 - OK**

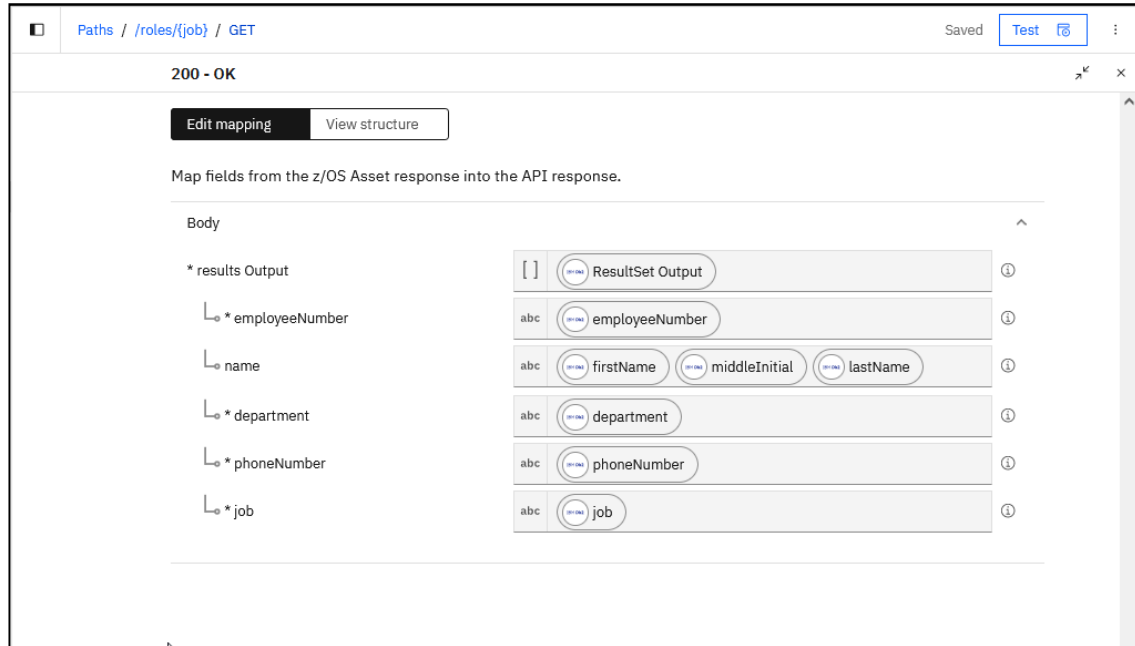
[Edit mapping](#) [View structure](#)

Map fields from the z/OS Asset response into the API response.

**Body**

Field	Value
* results Output	[ ] <a href="#">ResultSet Output</a>
↳ * employeeNumber	abc
↳ * name	abc
↳ * department	abc
↳ * phoneNumber	abc
↳ * job	abc

8. Be careful at this point to ensure you are selecting fields in the *ResultSet output* array in the *body* of the *zosAssetResponse*. The same property name may appear in another one of the available mappings, e.g., *apiRequest*, *ResultSet Row item*, etc. and if a property is selected from one these mappings, the results will be invalid.
9. Complete the mapping for the other properties. Notice the mapping of the Db2 REST response properties *firstName*, *middleInitial* and *lastName* into the API response property *name*.



10. Select the response for *404 Not found* response mapping and paste the text below in the *message* area.

***No records were found***

The screenshot shows the configuration interface for a 404 - Not Found response. At the top, there are two buttons: "Edit mapping" (highlighted) and "View structure". Below these buttons, a text label reads "Map fields from the z/OS Asset response into the API response." Underneath, there is a section labeled "Body" with a small upward arrow icon. In the "message" field, the text "No record were not found" is entered, preceded by a small "abc" icon. A small circular icon with an exclamation mark is visible to the right of the message field.

11. Finally in the 500 – Severe Error response mapping paste the following in the area for the message property

***A severe error has occurred - {{\$zosAssetResponse.body.StatusDescription}}***

The screenshot shows the configuration interface for a 500 - Severe Error response. At the top, there are two buttons: "Edit mapping" (highlighted) and "View structure". Below these buttons, a text label reads "Map fields from the z/OS Asset response into the API response." Underneath, there is a section labeled "Body" with a small upward arrow icon. In the "message" field, the text "A severe error has occurred - StatusDescription" is entered, preceded by a small "abc" icon. A small circular icon with an exclamation mark is visible to the right of the message field.

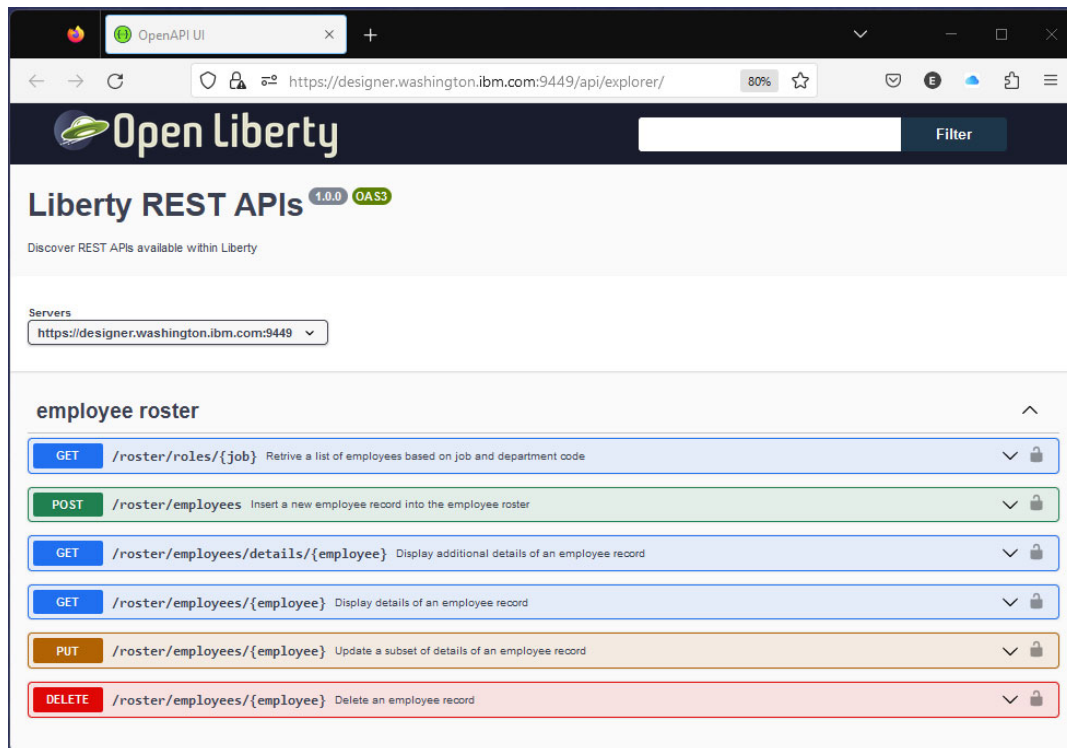
The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

## Testing APIs deployed in a z/OS Connect Designer container

The deployed APIs are accessible when the *Designer* container is active, even when the *Designer* is not opened in a browser. In fact, there are advantages in this behavior when testing security roles outside of the *Designer* since security tokens are cached by the browser.

This section will demonstrate using common HTTP clients to test APIs specifically for when security enabled.

We know the URI paths of the API from the initial page of the *API Explorer* displayed when testing in the *Designer*. From this page the first part of the URL can be determined, e.g., <https://designer.washington.ibm.com:9449>. This along with the URI path of each methods provides the URL we need to use to invoke a method. For example, to invoke the GET to display the additional details of an employee record in any client, the URL will be `https://designer.washington.ibm.com:9449/roster/employees/{employee}`



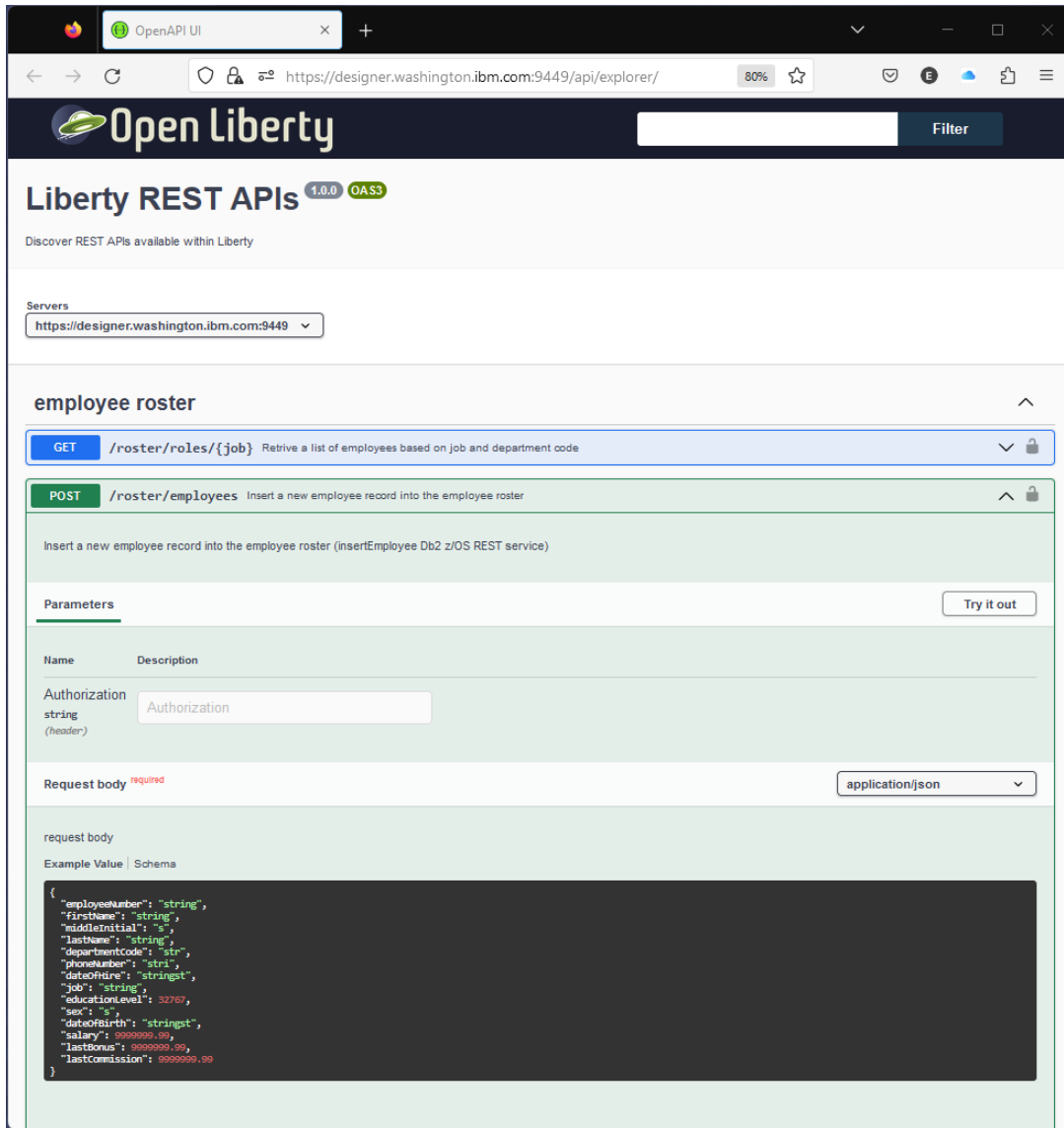
From this display, the methods and URLs required to access the API deployed in this container are:

- GET `https://designer.washington.ibm.com:9449/roster/roles/{job}`
- POST `https://designer.washington.ibm.com:9449/roster/employees`
- GET `https://designer.washington.ibm.com:9449/roster/employees/details/{employee}`
- GET `https://designer.washington.ibm.com:9449/roster/employees/{employee}`
- PUT `https://designer.washington.ibm.com:9449/roster/employees/{employee}`
- DELETE `https://designer.washington.ibm.com:9449/roster/employees/{employee}`

This section primarily covers the testing of the optional *DELETE*, *PUT*, and *GET details* methods. The tests for the POST and GET methods have already been covered in section *Testing the API's POST and GET methods* on page 39 and can be repeated using the *API Explorer* as described in this section.

1. Using the Firefox browser, go to URL <https://designer.washington.ibm.com:9449/api/explorer> to start the API Explorer.

2. Click on *Post /roster/employees* URI path to display the request body view of the URI path.



**Tech Tip:** You may be challenged by browser because the digital certificate used by the *Designer* 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.

2. Next press the **Try it out** button to enable the entry of a request message body

POST /roster/employees Insert a new employee record into the employee roster

Insert a new employee record into the employee roster (insertEmployee Db2 z/OS REST service)

Parameters Cancel

Name	Description
Authorization string (header)	Authorization

Request body required application/json

request body

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "str",
  "phoneNumber": "str",
  "dateOfHire": "stringst",
  "job": "string",
  "educationLevel": 32767,
  "sex": "s",
  "dateOfBirth": "stringst",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

3. Enter the JSON request message below in the *Request body* section and press the **Execute** button.

```
{
  "employeeNumber": "948498",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

4. Security was enabled in the original specification document, so you will be required to sign in with one of the identities defined in the basicSecurity.xml file explored earlier. Use **Fred** for the *Username* and **fredpwd** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

5. Scroll down the view and you should see the *Response body* with the expected successful message.

Responses

Curl

```
curl -X 'POST' \
  'https://designer.ibm.com:9449/roster/employees' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "employeeNumber": "948498",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
  }'
```

Request URL

https://designer.ibm.com:9449/roster/employees

Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Record for employee number 948498 was added successfully" }</pre> <p><a href="#">Download</a></p>

6. Press the **Execute** button again and observe the results. A row for this employee number already existed in the employee roster (a Db2 tables) so the request failed with an HTTP 500.

Responses

Curl

```
curl -X 'POST' \
  'https://designer.ibm.com:9449/roster/employees' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "employeeNumber": "948498",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0065",
    "dateOfHire": "10/15/1980",
    "job": "Staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
  }'
```

Request URL

https://designer.ibm.com:9449/roster/employees

Server response

Code	Details
500	<p>Error: Internal Server Error</p> <p>Response body</p> <pre>{   "message": "A severe error has occurred - Service zCEEService.insertEmployee.(V1) execution failed due to SQL error, SQLCODE=-803, SQLSTATE=23505, Message=AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE EMPL1YCA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS. RID OF EXISTING ROW IS X'0000000227'. Error Location:DSNLJXUS." }</pre> <p><a href="#">Download</a></p>



7. Scroll down and click on *GET /roster/employees/{employees}* URI path to display the request body view of the URI path for this method. Next click on the **Try it out** button to enable the entry of data for this method. Enter **948498** as the employee identity and press the **Execute** button to retrieve a subset of data for this employee.

The screenshot displays the 'Responses' section of an API client. It shows the 'Curl' command used to make the request, the 'Request URL' as 'https://designer.ibm.com:9449/roster/employees/948498', and the 'Server response' with a status code of 200. The 'Response body' is a JSON object containing employee details for ID 948498.

```
curl -X 'GET' \
  'https://designer.ibm.com:9449/roster/employees/948498' \
  -H 'accept: application/json'
```

Request URL

https://designer.ibm.com:9449/roster/employees/948498

Server response

Code	Details
200	<p>Response body</p> <pre>{   "results Output": [     {       "employeeNumber": "948498",       "name": "Matt T. Johnson",       "departmentCode": "C00",       "phoneNumber": "0065",       "job": "Staff"     }   ] }</pre>

8. Try this again using number **121212** and observe the results. You see the message that the employee was not found.
9. Expand the *PUT* method and enter press the **Try it out** button.
10. Enter **948498** in the *employee* field and paste the JSON below in the request body area.

```
{
  "salary": 5000.00,
  "bonus": 500.00,
  "commission": 400.00
}
```

11. Press the **Execute** button.

The screenshot displays the 'Responses' section of an API client for the PUT method. It shows the 'Curl' command with the JSON body from the previous step, the 'Request URL' as 'https://designer.ibm.com:9449/roster/employees/948498', and the 'Server response' with a status code of 200. The 'Response body' is a JSON object indicating a successful update.

```
curl -X 'PUT' \
  'https://designer.ibm.com:9449/roster/employees/948498' \
  -H 'accept: application/json' \
  -H 'Content-type: application/json' \
  -d '{
    "salary": 5000.00,
    "bonus": 500.00,
    "commission": 400.00
  }'
```

Request URL

https://designer.ibm.com:9449/roster/employees/948498

Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Record for employee 948498 successfully updated" }</pre>

12. Expand the *GET* method for URI path */roster/employees/details/{employee}* and enter press the **Try it out** button.
13. Enter **948498** in the *employee* field and press the **Execute** button. Observe that the updates values have been applied.

The screenshot shows the 'Responses' tab of an API client. The 'Curl' section contains the command: `curl -X 'GET' \ 'https://designer.ibm.com:9449/roster/employees/details/948498' \ -H 'accept: application/json'`. The 'Request URL' is `https://designer.ibm.com:9449/roster/employees/details/948498`. The 'Server response' section shows a status code of 200. The 'Response body' is a JSON object:

```
{
  "retrievedResults output": [
    {
      "employeeID": "948498",
      "name": "Matt T Johnson",
      "departmentCode": "C00",
      "phoneNumber": "0065",
      "dateOfHire": "1980-10-15",
      "job": "Staff",
      "educationLevel": 21,
      "sex": "M",
      "dateOfBirth": "1960-06-18",
      "annualSalary": 5000,
      "lastBonus": 500,
      "lastCommission": 400
    }
  ]
}
```

A 'Download' button is visible at the bottom right of the response body.

14. Expand the *DELETE* method for URI path */roster/employees/{employee}* and enter 948498 as the employee number press the **Try it out** button. Observe the record has been deleted.

The screenshot shows the 'Server response' section of the API client. The status code is 200. The 'Response body' is a JSON object:

```
{
  "message": "record deleted"
}
```

A 'Download' button is visible at the bottom right of the response body.

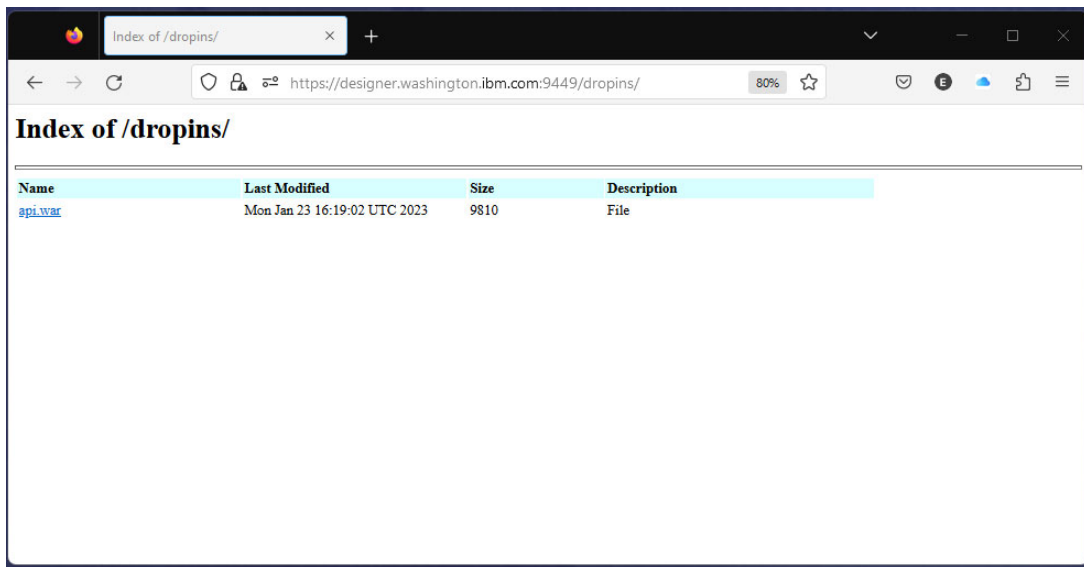
15. Repeat either of the two *GET* method request for employee **948498** and you see a message that the record could not be found.
16. Try other methods using other rows in the table. The initial contents of the Db2 table are shown below. See the table on page 85 for the contents of the Db2 table.

## Deploying and installing APIs in a z/OS Connect Native Server

As the *z/OS Connect Designer* is being used to develop the API from specification file, a Web Archive (WAR) file is being constantly regenerated and being automatically deployed to the z/OS Connect server embedded in the *Designer*. This section of the exercises provides details on how this WAR file can be extracted from the *Designer* container, moved to a zOS OMVS directory, and then added to a native z/OS Connect server.

### Moving the API Web Archive file from the container to a z/OS OMVS directory

1. The first step is to use the file serving capability added the Liberty server's configuration. Use a web browser to access URL <https://designer.washington.ibm.com:9449/dropins/>.



Double click the *api.war* file and use the save as option to save the file in local directory, e.g., *c:\z\openApi3\wars*. Specify a *File* name of ***employees.war***.

2. Open a DOS command prompt and use the change directory command to go to directory *C:\z\openApi3\wars*, e.g., ***cd C:\z\openApi3\wars***

3. Start a file transfer session with the WG31 host using the *psftp* command, e.g., ***psftp user1@wg31***

4. Enter USER1's password and then use the *cd* command to change to directory to directory */var/zcee/openApi3/apps*, e.g., ***cd /var/zcee/openApi3/apps***

5. Use the *put* command to move the file the remote directory, e.g., ***put employees.war***

6. When the transfer has completed enter the **quit** command.

```
c:\z\openApi3\wars>psftp user1@wg31
Using username "user1".
user1@wg31's password:
Remote working directory is /var/zcee/openApi3/apps
psftp> cd /var/zcee/openApi3/apps
Remote directory is now /S0W1/var/ats/zosconnect/servers/openApi3/apps
psftp> put employees.war
local:cscvinc.war => remote:/var/zcee/openApi3/apps/employees.war
psftp> quit
```

These steps have moved the WAR file from the Designer container to the OMVS directory accessible by the z/OS Connect native server.

## Updating the server xml

The next step is to add a *webApplication* server XML configuraton element for the API to the OpenAPI 3 server's configuration.

1. Edit OMVS file **server.xml** in directory */var/zcee/openApi3* and add this configuration element.

```
<webApplication id="db2" contextRoot="/roster" name="db2API"
location="${server.config.dir}apps/employees.war"/>
```

The addition of this configuration adds the web application found in the *employees.war* file to the server's configuration. The context root of */db2* is prepended is to the URI paths of each URI path found in the web application to ensure the uniqueness of this API's URI paths versus the URI paths of other APIs installed in the server.

2. Use MVS modify command **F ZCEEAPI3,REFRESH,CONFIG** to have the server XML changes installed.

**Tech-Tip:** To refresh an application using the MVS modify command **F ZCEEAPI3,REFRESH,APPS**

This completes the installation of the API's web application.

## Defining the required RACF EJBRole resources

The API has been installed but the required RACF EJBRoles have not been defined and access permitted. This section describes the steps required to complete the RACF configuration required to execute the API.

Remember the specification file defined two roles for invoking the methods of this API, *Manager* and *Staff*. In the *basicSecurity.xml* configuration file we saw how we configured these roles and granted access to the roles in a Liberty internal registry. On z/OS we want to use RACF. The names of the required RACF EJBRoles are derived by combining information from 3 sources. The first is the *profilePrefix* attribute of the server XML *safCredentials* configuration element. In our case, the value of *profilePrefix* is **ATSZDFLT**. The next source is the name of the web application. The name of the web application is either derived from information in the specification or the *name* attribute provided on the *webApplication* configuration element. In our case, this value should be **db2API**. The final source is the role name provided in the specification document, **Manager** or **Staff**. So, two EJBRoles need to be defined, **ATSZDFLT.db2API.Manager** and **ATSZDFLT.db2API.Staff**.

1. Use the RACF RDEFINE command to define EJBROLE **ATSZDFLT.db2API.Manager**.

```
rdefine ejbrole ATSZDFLT.db2API.Manager uacc(none)
```

2. Use the RACF RDEFINE command to define EJBROLE **ATSZDFLT.db2API.Staff**.

```
rdefine ejbrole ATSZDFLT.db2API.Staff uacc(none)
```

3. Use the RACF PERMIT command to permit identity FRED READ access to EJBROLE ATSZDFLT.db2API.Manager.

```
permit ATSZDFLT.db2API.Manager class(ejbrole) id(fred) acc(read)
```

4. Use the RACF PERMIT command to permit identity FRED READ access to EJBROLE ATSZDFLT.Db2API.STAFF.

```
permit ATSZDFLT.db2API.Staff class(ejbrole) id(fred) acc(read)
```

5. Use the RACF PERMIT command to permit identity USER1 READ access to EJBROLE ATSZDFLT.Db2API.STAFF.

```
permit ATSZDFLT.db2API.Staff class(ejbrole) id(user1) acc(read)
```

6. Use the RACF SETROPTS command to refresh the EJBRole instorage profiles.

```
setropts raclist(ejbrole) refresh
```

Now we are ready to test the invoking of the methods of this API.

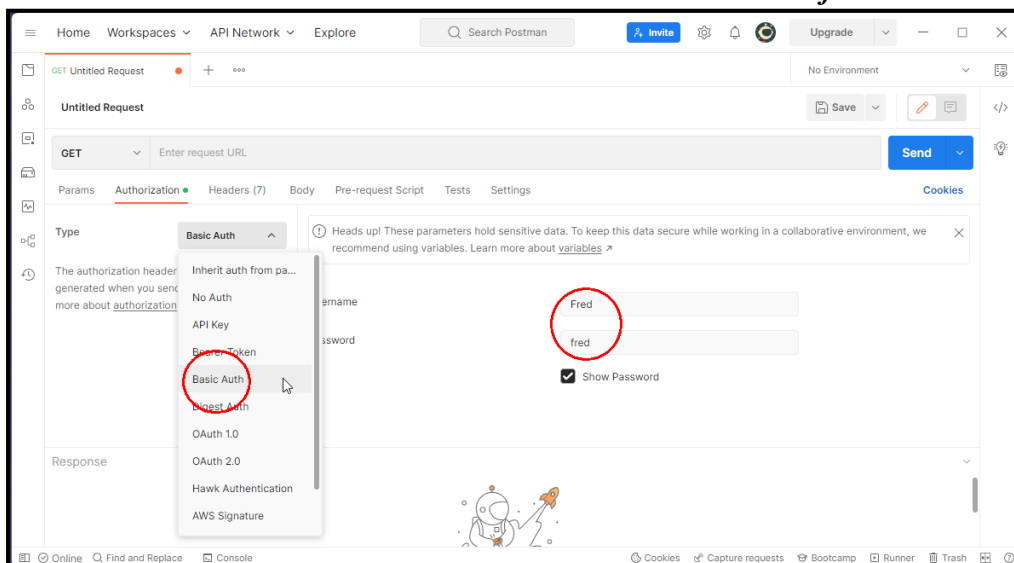
## Testing APIs deployed in a native z/OS server

This section assumes the optional configuration steps in section *Complete the configuration of the API (Optional)* on page 43 have completed. If this section was skipped, then some of the test in this section cannot be performed.

### Using Postman

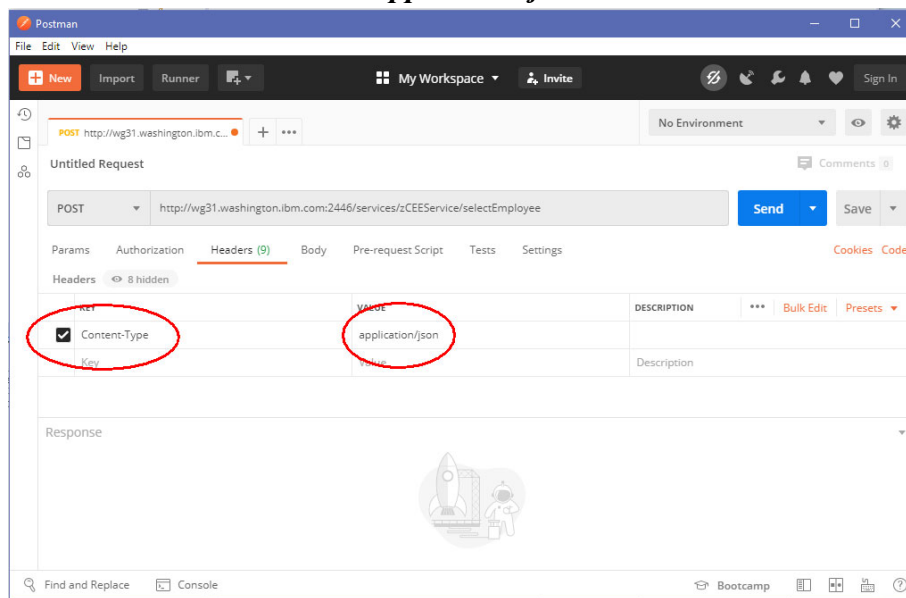
Start a Postman session using the Postman icon on the desktop.

1. Open the *Postman* tool icon on the desktop and if necessary reply to any prompts and close any welcome messages and select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter **Fred** as the *Username* and **fred** as the *Password*.



**Tech-Tip:** If the above Postman view is not displayed select *File* on the toolbar and then choose *New Tab* on the pull down. Alternatively, if the *Launchpad* view is displayed, click on the *Create a request* option.

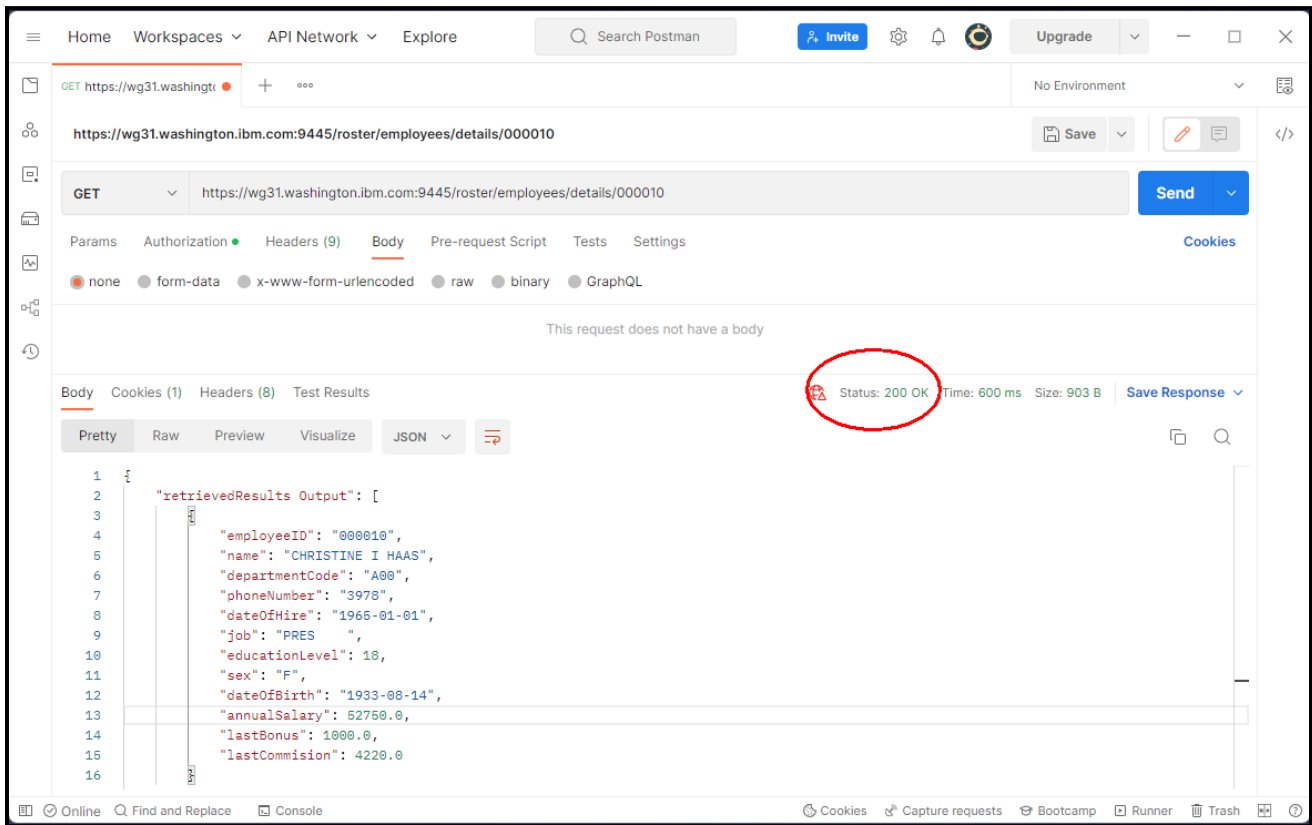
- Next select the *Headers* tab and under *KEY* use the code assist feature to enter ***Content-Type*** and under *VALUE* use the code assist feature to enter ***application/json***.



**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.

2. Next select the *Body* tab and select the *raw* radio button. Then use the down arrow in the *Body* tab to select **GET** and enter **<https://wg31.washington.ibm.com:9445/roster/employees/details/000010>** in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

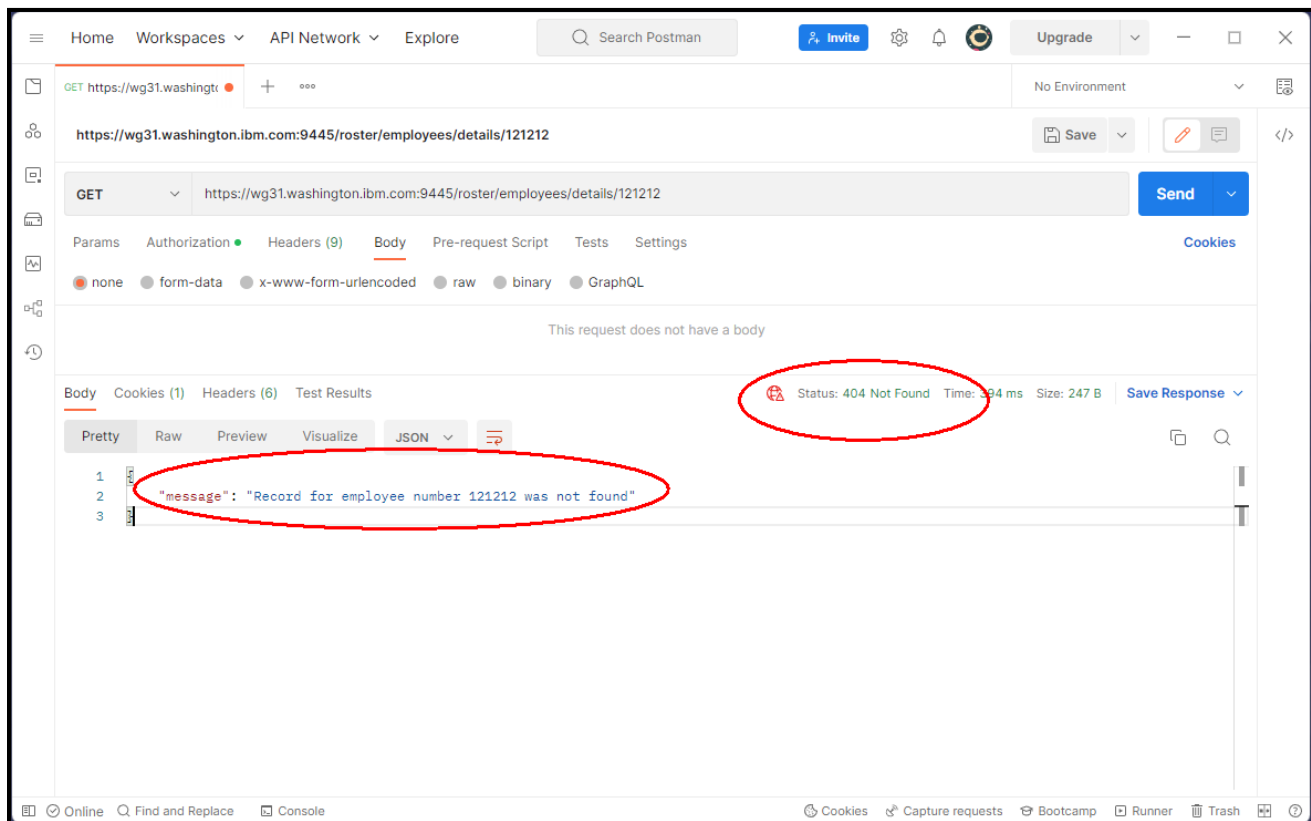
## IBM z/OS Connect (OpenAPI 3.0)



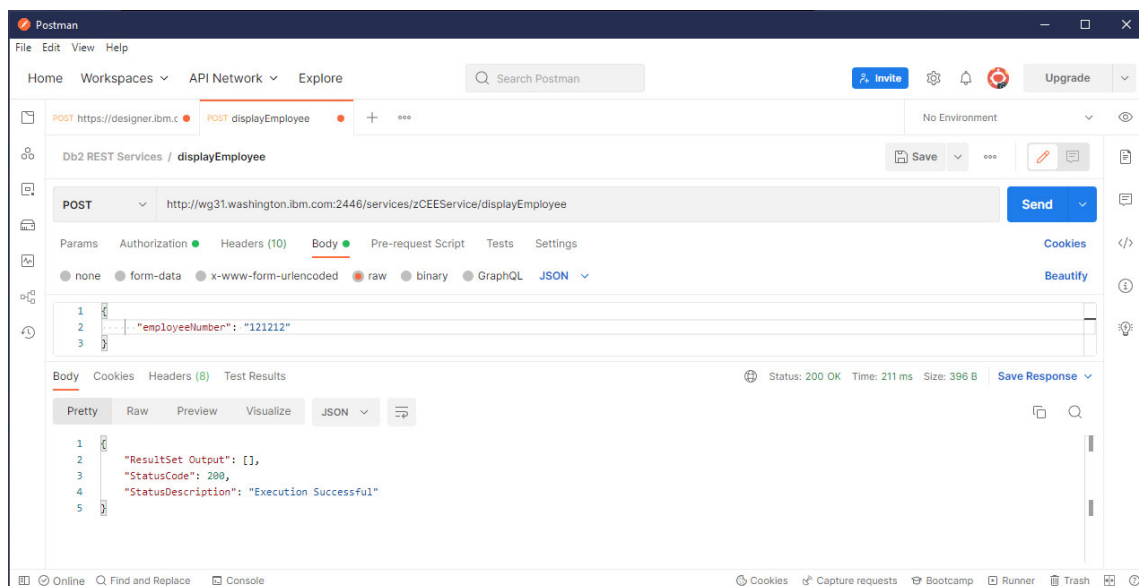
Notice what is different from the earlier testing when the API was deployed in the *Designer* container. First the credentials were for the RACF credentials (*fred*) with access to the RACF EJBRole.



3. Next enter an invalid employee number such as 121212, <https://wg31.washington.ibm.com:9445/roster/employees/details/121212> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.



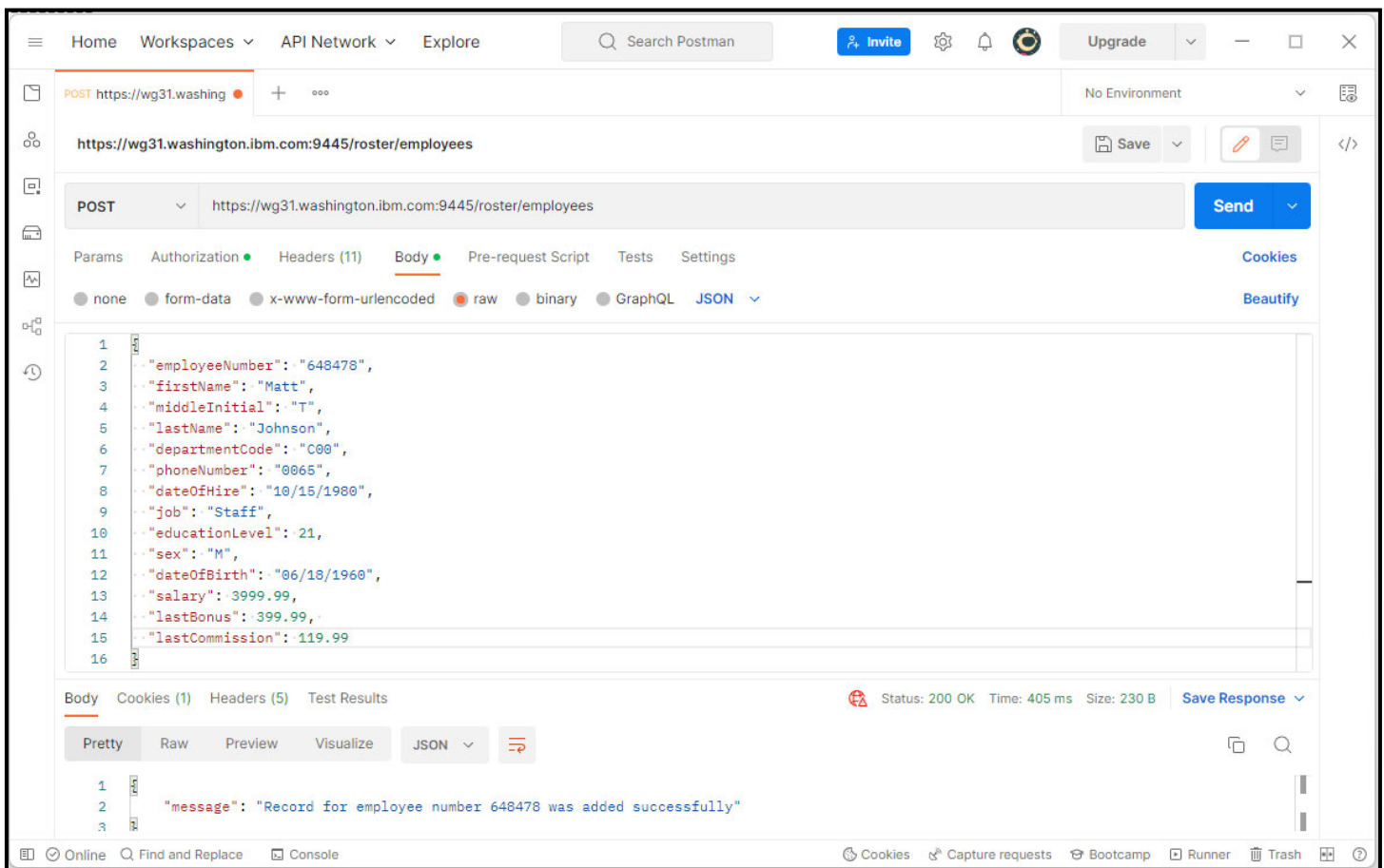
If you invoked the underlying Db2 REST service (`/services/zCEEService/displayEmployee`) you would receive an HTTP 200 with an empty *ResultSet Output* array.



The API created in the *Designer* essentially intercepted the response from the Db2 service and was to determine no results were found and returned an HTTP 404 (not found) to the client rather than an HTTP 200 (OK).

Optional, experiment using *Postman* to invoke other methods of the API. For example, if you want to invoke a *POST* with URI path */roster/employees*, use the JSON below for the request message.

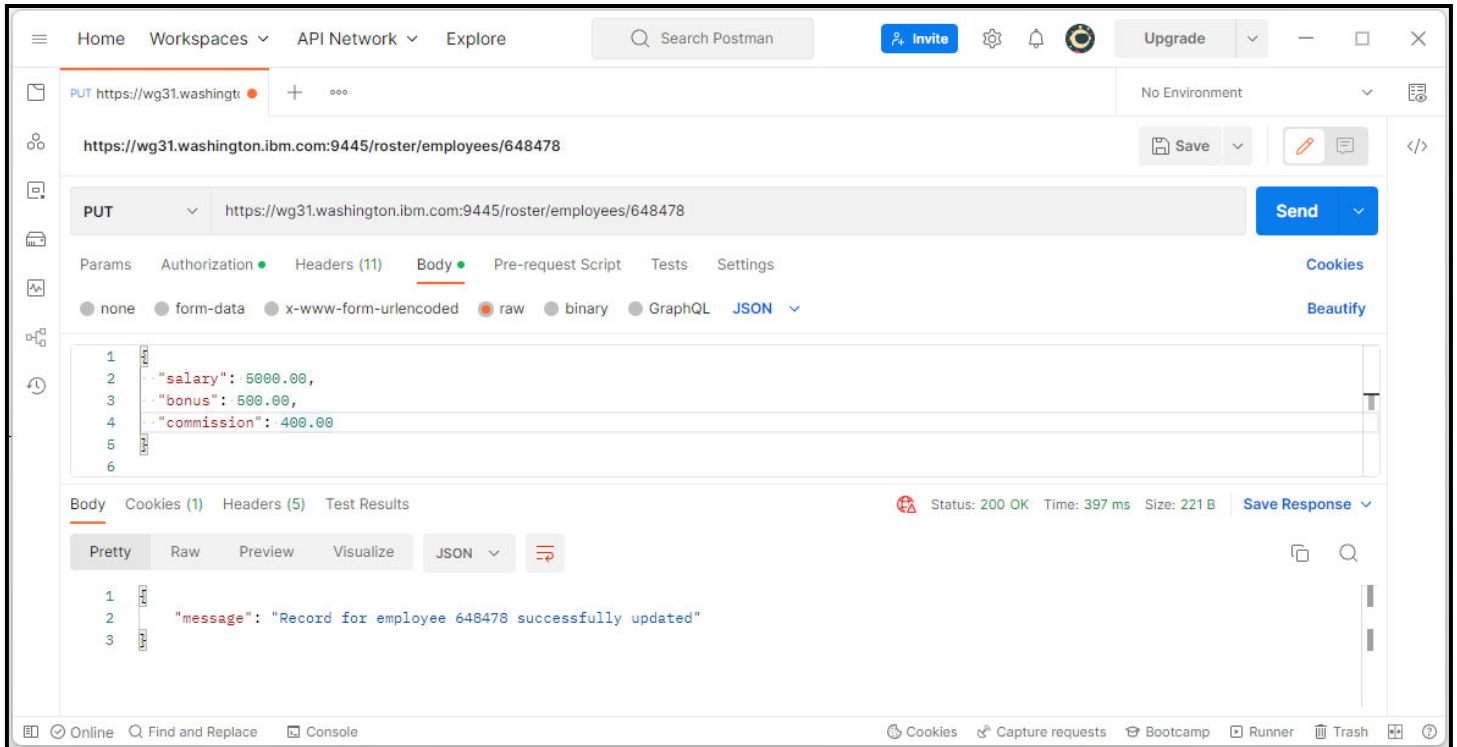
```
{
  "employeeNumber": "648478",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```



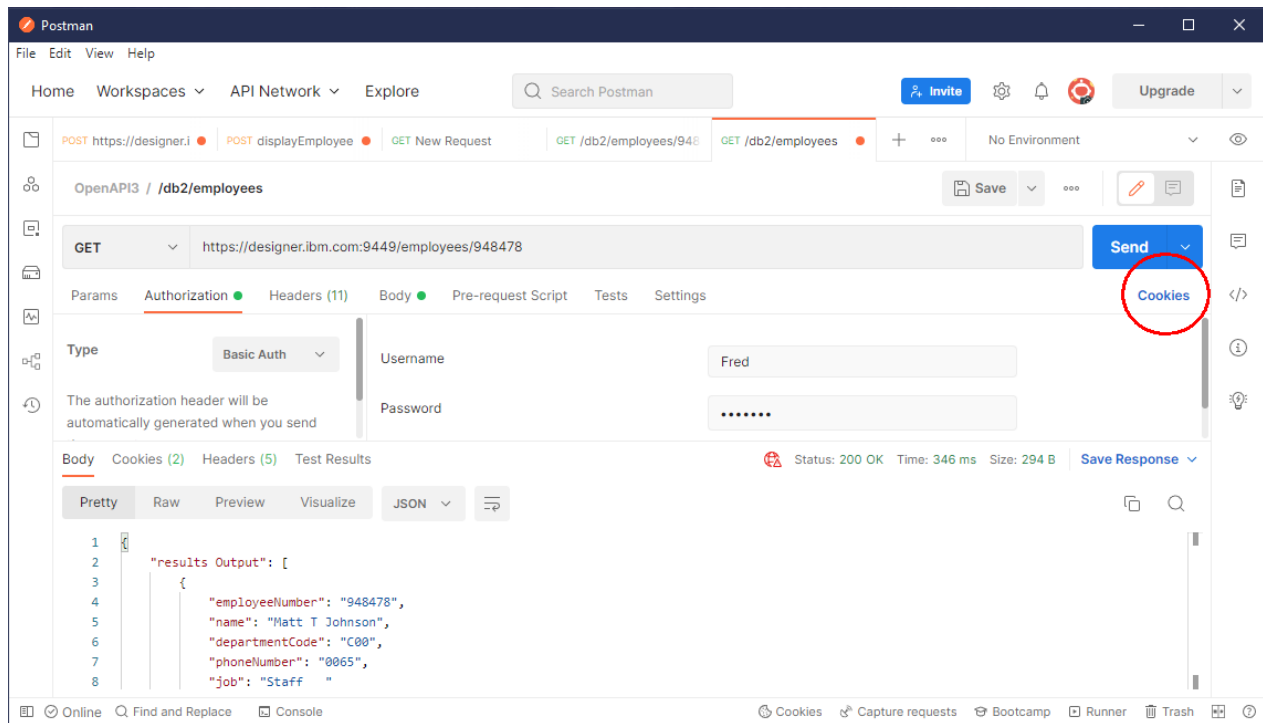
Or if you want to do a *PUT* with URI path `/roster/employees/{employee}` use this JSON request message.

<https://wg31.washington.ibm.com:9445/roster/employees/648489>

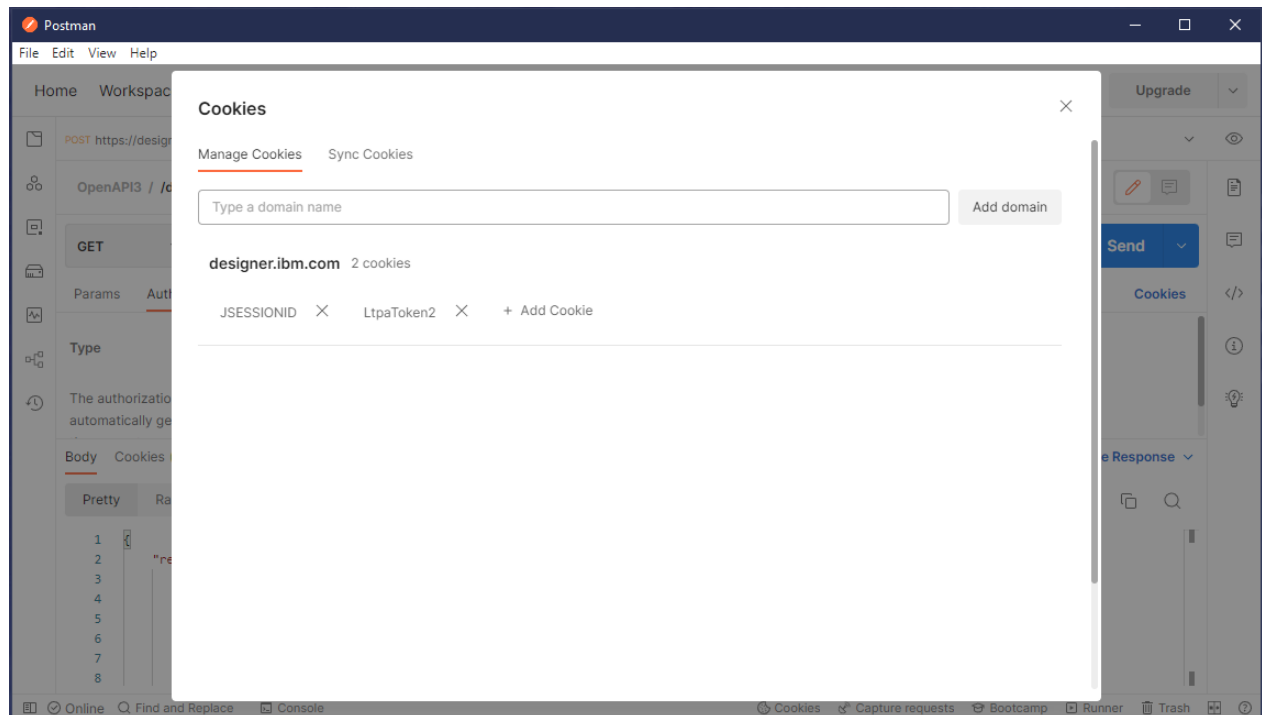
```
{  
  "salary": 5000.00,  
  "bonus": 500.00,  
  "commission": 400.00  
}
```



4. Up until this point you have been using the role assigned to user *Fred*. Now experiment using user *user1* and/or *user2*. Before we can use other credentials we have to clear the credentials that cached by *Postman*. Unless this is done, *Postman* will continue to use the credentials for *Fred* regardless of what is provided in the authorization header
5. To clear the *Postman* cached tokens, click on the *Cookies* section of the *Postman* window and



And delete any *JSESSIONID* and *LtpaToken2* cookies displayed.



Test various methods using Username *user1* and *user2* and observe the results. Remember, *user1* can only invoke GET methods and *user2* can not invoke any method.

## Using cURL

*Client for URL (cURL)* is a common tool for driving REST client request to APIs. In this section, the *curl* command will be used to test the API's methods deployed into the *z/OS Connect Designer's* container and more importantly, demonstrate role-based security. *Postman* caches security credentials between tests and the cached credentials must be cleared if the identity being used is changed. *cURL* does not this caching of credentials and therefore it is easier to change security credentials between request with *cURL* than with *Postman*.

\_\_\_\_ 1. Start a DOS command prompt session and go to directory *c:\z\openapi3*, e.g., *cd \z\openap3*.

\_\_\_\_ 2. Enter the *curl* command below and observe the response.

```
curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/roster/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://localhost:9449/roster/employees/details/000010
{"retrievedResults Output":[{"employeeID":"000010","name":"CHRISTINE I HAAS",
"departmentCode":"A00","phoneNumber":"3978","dateOfHire":"1965-01-01","job":"PRES  ",
"educationLevel":18,"sex":"F","dateOfBirth":"1933-08-14","annualSalary":52750.0,
"lastBonus":1000.0,"lastCommision":4220.0}]} - HTTP CODE 200
```

Fred is a member of the *Staff* group and has *Staff* access to the **Staff** role. Any identity with **Staff** access can invoke one of the GET methods.

\_\_\_\_ 3. Enter the *curl* command below and observe the response.

```
curl -X GET -w " - HTTP CODE %{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/roster/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w " - HTTP CODE %{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://localhost:9449roster//employees/details/000010
- HTTP CODE 403
```

**Tech-Tip:** If you had provided an invalid password, e.g., *--user user2:userx*, the request would have failed with an HTTP status of 401, *Unauthorized*.

4. Enter the curl command below and observe the response.

```
curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred --data @insertEmployee.json https://wg31.washington.ibm.com:9445/roster/employees/
```

In the above command, the file insertEmployee.json has the contents below:

```
{
  "employeeNumber": "748478",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

```
c:\z\openapi3>curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type:
application/json" --insecure --user Fred:fred --data @insertEmployee.json
https://wg31.washington.ibm.com:9445/roster/employees/
```

5. Enter the curl command below to invoke the *GET* method with URI path */roles/{job}*.

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred https://wg31.washington.ibm.com:9445/roster/roles/PRES?dept=A00
```

6. Enter the curl command below to invoke the *DELETE* method with URI path */employees/{employee}*.

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred https://wg31.washington.ibm.com:9445/roster/employees/000012
```

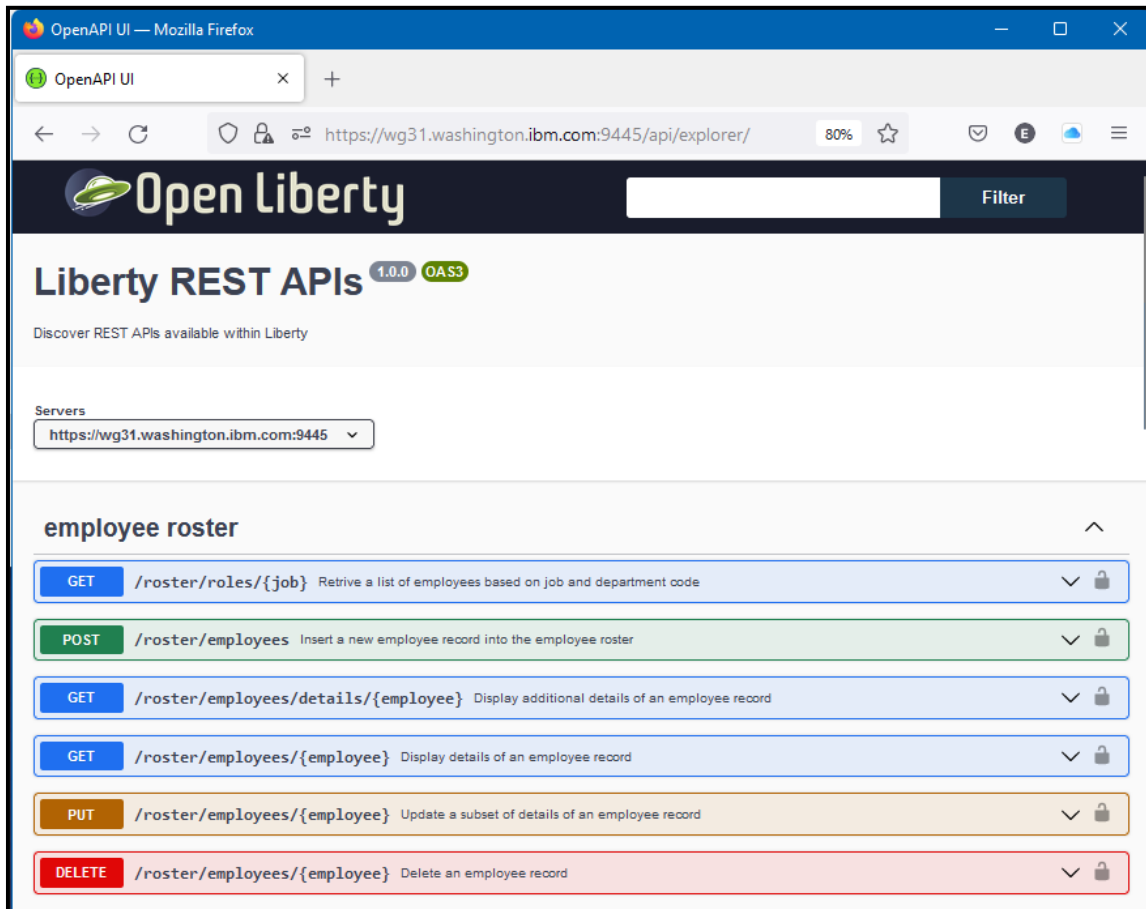
7. Enter the curl command below to invoke the *DELETE* method with URI path */employees/{employee}*.

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user1:user1 https://wg31.washington.ibm.com:9445/roster/employees/000012
```

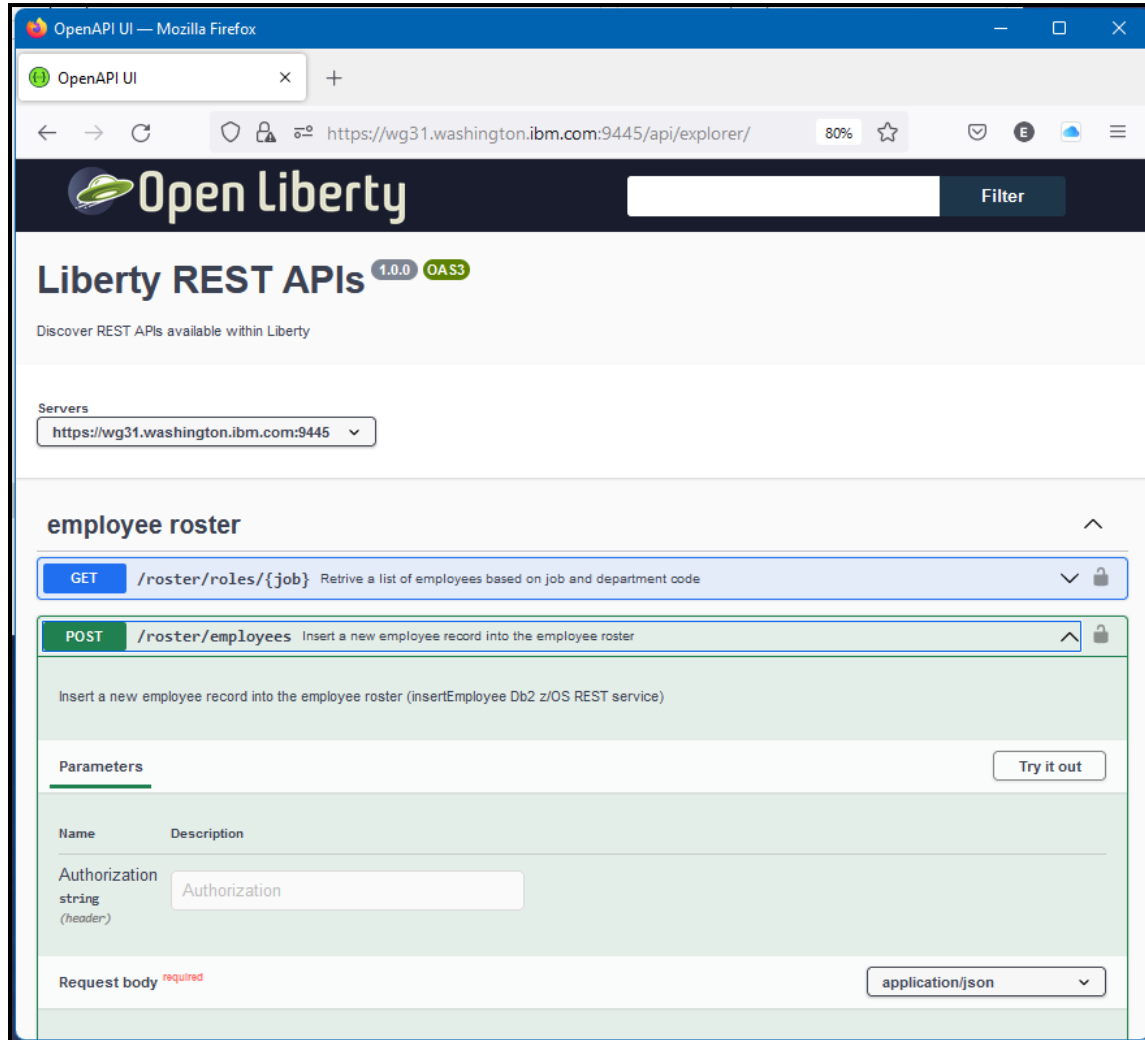
## Using the API Explorer

The API Explorer used to test the APIs in the Designer can also be used to test API once they are deployed to a native server.

1. Using the Firefox browser, go to URL <https://wg31.washington.ibm.com:9445/api/explorer> to start the API Explorer.



- \_\_\_ 2. Use the pull-down arrow in the *Servers* box at the top of the page and select <https://wg31.washington.ibm.com:9445>
- \_\_\_ 3. Click on *Post /roster/employees* URI path to display the request body view of the URI path.





3. Next press the **Try it out** button to enable the entry of a request message body

POST /roster/employees Insert a new employee record into the employee roster

Insert a new employee record into the employee roster (insertEmployee Db2 z/OS REST service)

Parameters Cancel

Name	Description
Authorization string (header)	Authorization

Request body required application/json

request body

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "str",
  "phoneNumber": "str",
  "dateOfHire": "stringst",
  "job": "string",
  "educationLevel": 32767,
  "sex": "s",
  "dateOfBirth": "stringst",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

4. Enter the JSON request message below in the *Request body* section and press the **Execute** button.

```
{
  "employeeNumber": "848478",
  "firstName": "Matt",
  "middleInitial": "T",
  "lastName": "Johnson",
  "departmentCode": "C00",
  "phoneNumber": "0065",
  "dateOfHire": "10/15/1980",
  "job": "Staff",
  "educationLevel": 21,
  "sex": "M",
  "dateOfBirth": "06/18/1960",
  "salary": 3999.99,
  "lastBonus": 399.99,
  "lastCommission": 119.99
}
```

5. Security was enabled in the original specification document, so you will be required to sign in with one of the identities defined in the basicSecurity.xml file explored earlier. Use **Fred** for the *Username* and **fredpwd** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

6. Scroll down the view and you should see the *Response body* with the expected successful message.

Responses

Curl

```
curl -X 'POST' \
  'https://wg31.washington.ibm.com:9445/roster/employees' \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
    "employeeNumber": "848478",
    "firstName": "Matt",
    "middleInitial": "T",
    "lastName": "Johnson",
    "departmentCode": "C00",
    "phoneNumber": "0009",
    "dateOfHire": "10/15/1980",
    "job": "staff",
    "educationLevel": 21,
    "sex": "M",
    "dateOfBirth": "06/18/1960",
    "salary": 3999.99,
    "lastBonus": 399.99,
    "lastCommission": 119.99
  }'
```

Request URL

https://wg31.washington.ibm.com:9445/roster/employees

Server response

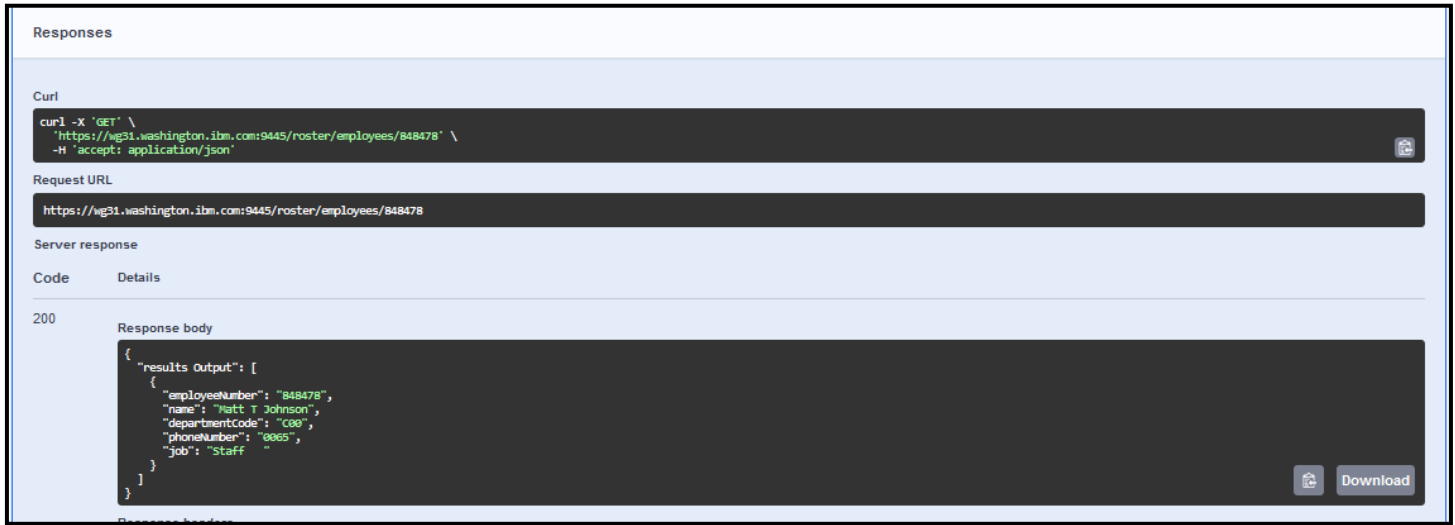
Code	Details
200	<p>Response body</p> <pre>{   "message": "Record for employee number 848478 was added successfully" }</pre> <p>Response headers</p> <pre>content-language: en-US content-length: 70 content-type: application/json date: Sun, 02 Oct 2022 15:11:29 GMT x-firefox-spydy: h2 x-powered-by: Servlet/4.0</pre>

7. Press the **Execute** button again and observe the results. A row for this employee number already existed in the employee roster (a Db2 tables) so the request failed with an HTTP 500.

Server response

Code	Details
500	<p>Error: Internal Server Error</p> <p>Response body</p> <pre>{   "message": "A severe error has occurred - Service zCEEService.insertEmployee.(v1) execution failed due to SQL error, SQLCODE=-803, SQLSTATE=23505, Message=AN INSERTED OR UPDATED VALUE IS INVALID BECAUSE INDEX IN INDEX SPACE EMPYLYCA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS. RID OF EXISTING ROW IS X'000000022A'. Error Location: }</pre>

8. Scroll down and click on *GET /roster/employees/{employees}* URI path to display the request body view of the URI path for this method. Next click on the **Try it out** button to enable the entry of data for this method. Enter **848478** as the employee identity and press the **Execute** button to retrieve a subset of data for this employee.



Responses

Curl

```
curl -X 'GET' \
  'https://wg31.washington.ibm.com:9445/roster/employees/848478' \
  -H 'accept: application/json'
```

Request URL

```
https://wg31.washington.ibm.com:9445/roster/employees/848478
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "results Output": [     {       "employeeNumber": "848478",       "name": "Matt T Johnson",       "departmentCode": "C00",       "phoneNumber": "0000",       "job": "Staff"     }   ] }</pre> <p>Download</p>

9. Try this again using number **121212** and observe the results. You see the message that the employee was not found.
10. Expand the *PUT /roster/employees/{employees}* method and enter press the **Try it out** button.
11. Enter **848478** in the *employee* field and paste the JSON below in the request body area.

```
{
  "salary": 5000.00,
  "bonus": 500.00,
  "commission": 400.00
}
```

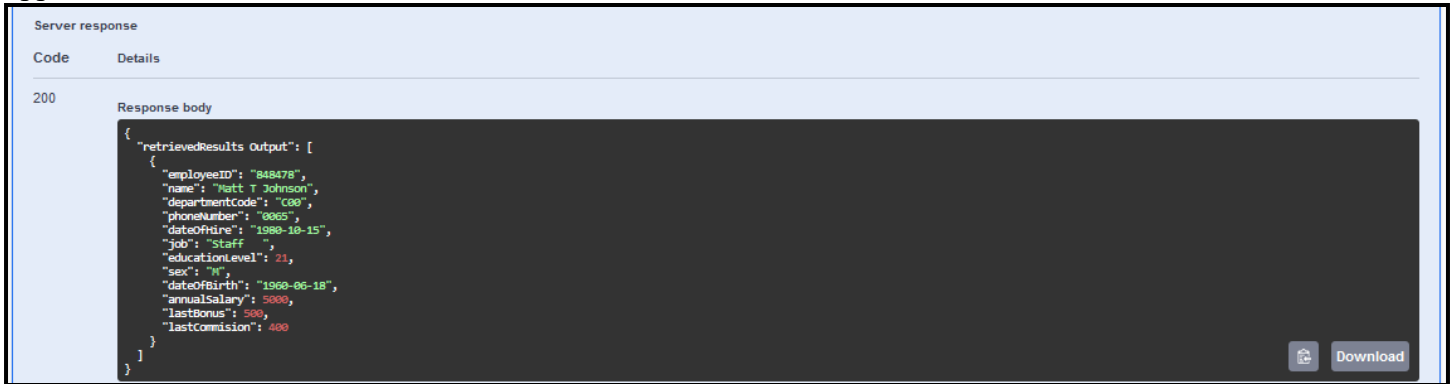
12. Press the **Execute** button.



Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Record for employee 848478 successfully updated" }</pre> <p>Download</p>

13. Expand the *GET* method for URI path */roster/employees/details/{employee}* and enter press the **Try it out** button.
14. Enter **848478** in the *employee* field and press the **Execute** button. Observe that the updates values have been applied.



Server response

Code	Details
200	<p>Response body</p> <pre>{   "retrievedResults Output": [     {       "employeeID": "848478",       "name": "Matt T Johnson",       "departmentCode": "000",       "phoneNumber": "8885",       "dateOfHire": "1980-10-15",       "job": "Staff",       "educationLevel": 21,       "sex": "M",       "dateOfBirth": "1960-06-18",       "annualSalary": 5000,       "lastBonus": 500,       "lastCommision": 400     }   ] }</pre> <p>Download</p>

15. Expand the *DELETE* method for URI path */employees/{employee}* and enter 848478 as the employee number press the **Try it out** button. Observe the record has been deleted.



Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "record deleted" }</pre> <p>Download</p>

16. Repeat either of the two *GET* method request for employee **948478** and you see a message that the record could not be found.

Try other methods using other rows in the table. The initial contents of the Db2 table are shown below. See the table on page 85 for the contents of the Db2 table

**Congratulations, you have completed this exercise.**

## Additional information and samples

This section provides the contents of the Db2 table used in this exercise as well as the JCL used to load the Db2 table. There is an introduction to performing problem determination while developing APIs.

The initial contents of the Db2 table are shown below.

EMPNO	FIRSTNAME	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

***JCL to define and load the Db2 table USER1.EMPLOYEE***

Below is the JCL used to load the Db2 table accessed by this API. This table was based on the standard Db2 sample employee with some constraints removed.

```
//EMPLOYEE EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSN2)
RUN PROGRAM(DSNTIAD) PLAN(DSNTIA12) -
LIB('DSN2.RUNLIB.LOAD')
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN DD *
DROP TABLE USER1.EMPLOYEE;
COMMIT;

CREATE TABLE USER1.EMPLOYEE
(EMPNO CHAR(6) NOT NULL,
FIRSTNME VARCHAR(12) NOT NULL,
MIDINIT CHAR(1) NOT NULL,
LASTNAME VARCHAR(15) NOT NULL,
WORKDEPT CHAR(3) ,
PHONENO CHAR(4) ,
HIREDATE DATE ,
JOB CHAR(8) ,
EDLEVEL SMALLINT ,
SEX CHAR(1) ,
BIRTHDATE DATE ,
SALARY DECIMAL(9, 2) ,
BONUS DECIMAL(9, 2) ,
COMM DECIMAL(9, 2) ,
PRIMARY KEY(EMPNO));

GRANT ALL PRIVILEGES ON TABLE USER1.EMPLOYEE TO USER1;

//LOAD EXEC DSNUPROC,PARM='DSN2,DSNTEX',COND=(4,LT)
//SORTLIB DD DSN=SYS1.SORTLIB,DISP=SHR
//SORTOUT DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK01 DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK02 DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK03 DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK04 DD UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//DSNTRACE DD SYSOUT=*
//SYSRECEM DD DSN=DSN1210.DB2.SDSNSAMP(DSN8LEM),
// DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(4000,(50,50),,,ROUND)
//SYSIN DD *

LOAD DATA INDDN(SYSRECEM) CONTINUEIF(72:72)='X'
INTO TABLE USER1.EMPLOYEE
(EMPNO POSITION( 1) CHAR(6),
FIRSTNME POSITION( 8) VARCHAR,
MIDINIT POSITION(21) CHAR(1),
LASTNAME POSITION(23) VARCHAR,
WORKDEPT POSITION(36) CHAR(3),
PHONENO POSITION(40) CHAR(4),
HIREDATE POSITION(45) DATE EXTERNAL,
JOB POSITION(56) CHAR(8),
EDLEVEL POSITION(65) INTEGER EXTERNAL(2),
SEX POSITION(68) CHAR(1),
BIRTHDATE POSITION(80) DATE EXTERNAL,
SALARY POSITION(91) INTEGER EXTERNAL(5),
BONUS POSITION(97) INTEGER EXTERNAL(5),
COMM POSITION(103) INTEGER EXTERNAL(5))
ENFORCE NO
```

## Designer problem determination

In this section, we will explore various scenarios using tracing to resolve API development issues, the trace output was created using this trace specification.

```
<logging traceSpecification="
  zosConnectCics=all:
  zosConnectDb2=all"
/>
```

1. Invoking a method returned an HTTP 404 response and no other response. There should have been a message displayed to state that a record for this employee did not exist. A review of the trace showed the Db2 REST service did return an HTTP 404 status code and a status message. The status message indicated that the request Db2 REST service did not exist. The resource not found (HTTP 404) was the REST service, not the employee record.

The screenshot shows a web browser window displaying the trace log for the z/OS Connect Designer. The log contains several entries, including a POST request to the zCEEService/displayEmployee/V1 endpoint. The response is an HTTP 404 status code, indicating that the resource was not found. The log also shows the response headers and the status message.

```
[6/29/22 12:21:08:558 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > sendRequest
Entry
/services
{"employeeNumber":"948499"}
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > POST
setShouldRebuild Entry
false
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < setBasicAuth
Entry
org.apache.cxf.jaxrs.client.spec.InvocationBuilderImpl@20fb0e37
java.lang.String@5ee6633
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < setBasicAuth
Exit
[6/29/22 12:21:08:573 UTC] 00000226 id=00000000 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > Sending POST
request to http://wg31.washington.ibm.com:2446/services/zCEEService/displayEmployee/V1
[6/29/22 12:21:08:779 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < sendRequest
Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl < invoke Exit
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > db2AssetWrapper
Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset >
constructHeadersObject Entry
org.apache.cxf.jaxrs.impl.ResponseImpl@418aa7b3
[6/29/22 12:21:08:779 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < {}
constructHeadersObject Exit
[6/29/22 12:21:08:780 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < db2AssetWrapper
Exit
{"headers":
{"connection":"close","Content-Language":"en-US","Content-Length":"158","content-type":"application/json; charset=UTF-8",
"Date":"Wed, 29 Jun 2022 12:21:11 GMT","Server":"DB2 DDF Native REST, DSN2LOC, DSNLJENG 10/02/19 UI65644","X-Correlation-ID":"C0A8006A.664A.DBB4578FD7F8","X-Powered-By":"DB2 for z/OS","Content-Type":"application/json; charset=UTF-8"},"body":
{"statusCode":404,"statusDescription":"Service zCEEService.displayEmployee.(V1) execution failed due to the service is
undefined. Error Location:DSNLJACC:86"},"cookies":{},"statusCode":404}
[6/29/22 12:21:08:780 UTC] 00000226 id=b5ec8df4 com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < invoke Exit
```

2. Here is another situation. Invoking a method returned an HTTP 404 response and no other details. There should have been a message displayed to state that a record for this employee did not exist. A review of the trace showed the Db2 REST service did return an HTTP 200 code but not the results array. In this case a review of trace showed the issue was that the response mapping for this situation, *response\_404.yaml*, did not exist in the API. It was this resource not being found which generated the HTTP 404 (not found) condition, not the employee record and not the Db2 REST service.

