

IBM z/OS Connect (OpenAPI 3.0)

Developing Native Server RESTful APIs for accessing Db2 REST Services



**IBM Z
Wildfire Team –
Washington System Center**

Table of Contents

Overview	3
Db2 REST services and z/OS Connect.....	4
Creating and testing Db2 REST Services.....	4
The z/OS Connect Designer Container environment	13
The container configuration file.....	13
Connecting to Db2 and the required server XML configuration.....	14
Basic security and the required server XML configuration	14
Accessing the z/OS Connect Designer log and trace files	16
Developing a z/OS Connect APIs that accesses Db2	18
Configure the POST method for URI path /employees.....	20
Configure the GET method for URI path /employees/{employee}.....	35
Testing the API's POST and GET methods	40
Compete the configuration of the API (Optional).....	44
Configure the PUT method for URI path /employees/{employee}	44
Configure the GET method for URI path /employees/details/{employee}	48
Configure the DELETE method for URI path /employees/{employee}	53
Configure the GET method for URI path /roles/{job}	57
Testing APIs deployed in a z/OS Connect Designer container	62
Using Postman.....	63
Using cURL	70
Using the API Explorer.....	73
Deploying and installing APIs in a z/OS Connect Native Server	79
Moving the API Web Archive file from the container to a z/OS OMVS directory.....	79
Updating the server xml	80
Defining the required RACF EJBRole resources.....	81
Testing APIs deployed in a native z/OS server	82
Using Postman.....	82
Using cURL	87
Additional information and samples.....	89
JCL to define and load the Db2 table USER1.EMPLOYEE.....	90
Designer problem determination.....	91
The contents of the employees.yaml file	93

Important: On the desktop there is a file named *OpenAPI 3 development APIs CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

Overview

The objective of these exercises is to gain experience with working with *z/OS Connect* and 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 on the desktop
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise. For example, the text might reference host name *designer:washington.ibm.com* when a screen shot shows the host as *designer:ibm.com* or even *localhost*. All of 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 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

Creating and testing 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). Only the latter technique that will be shown in this exercise.

1. Log onto TSO and access data set USER1.ZCEE.CNTL
2. Select member *DB2REST1* in *USER1.ZCEE.CNTL* and right- mouse button clicking and submit this job for execution. It should complete with a completion code of 0.

```
//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).

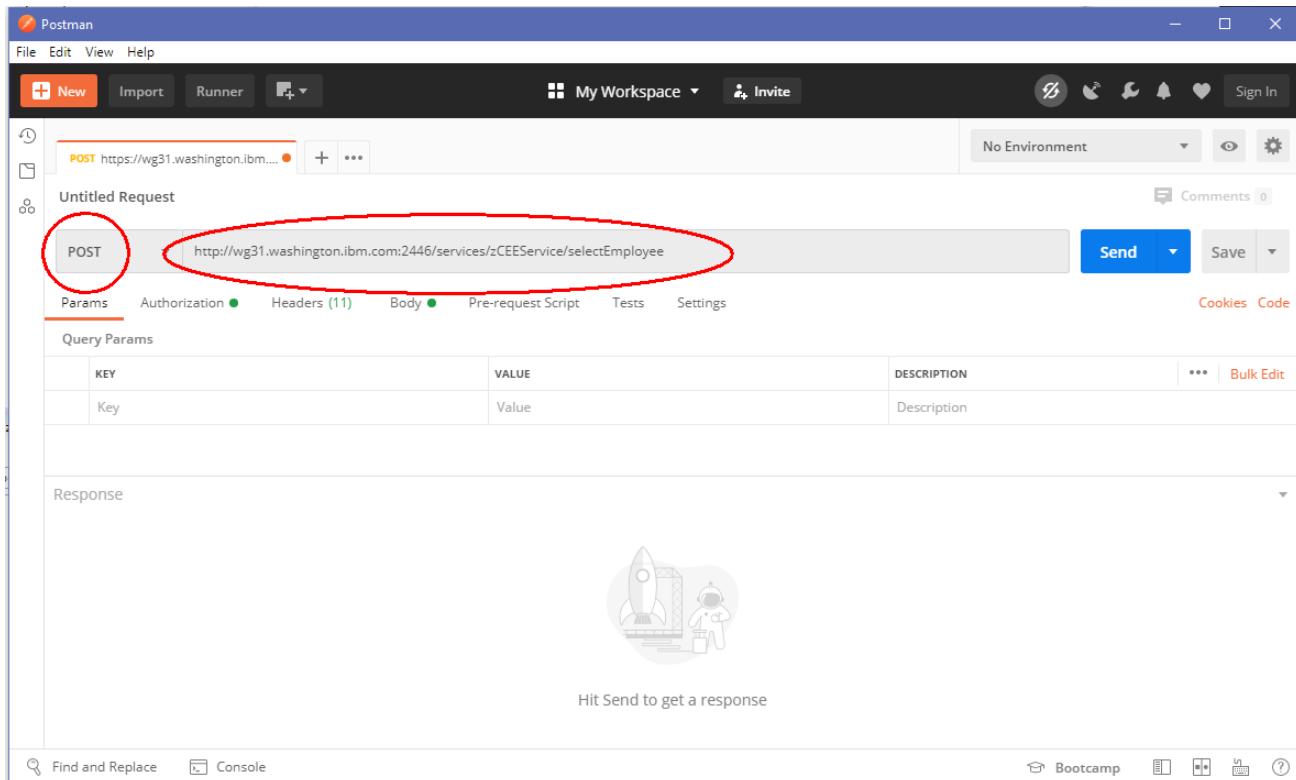
Important: The DBA creating this native Db2 REST service is excluding other table columns, e.g., SEX, SALARY, BONUS, COMMISION, etc. from the selection by omitting these columns from the SELECT statement. The DBA's use of the AS cause will also ensure the assigning of meaningful JSON property names rather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

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

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

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

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



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.

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

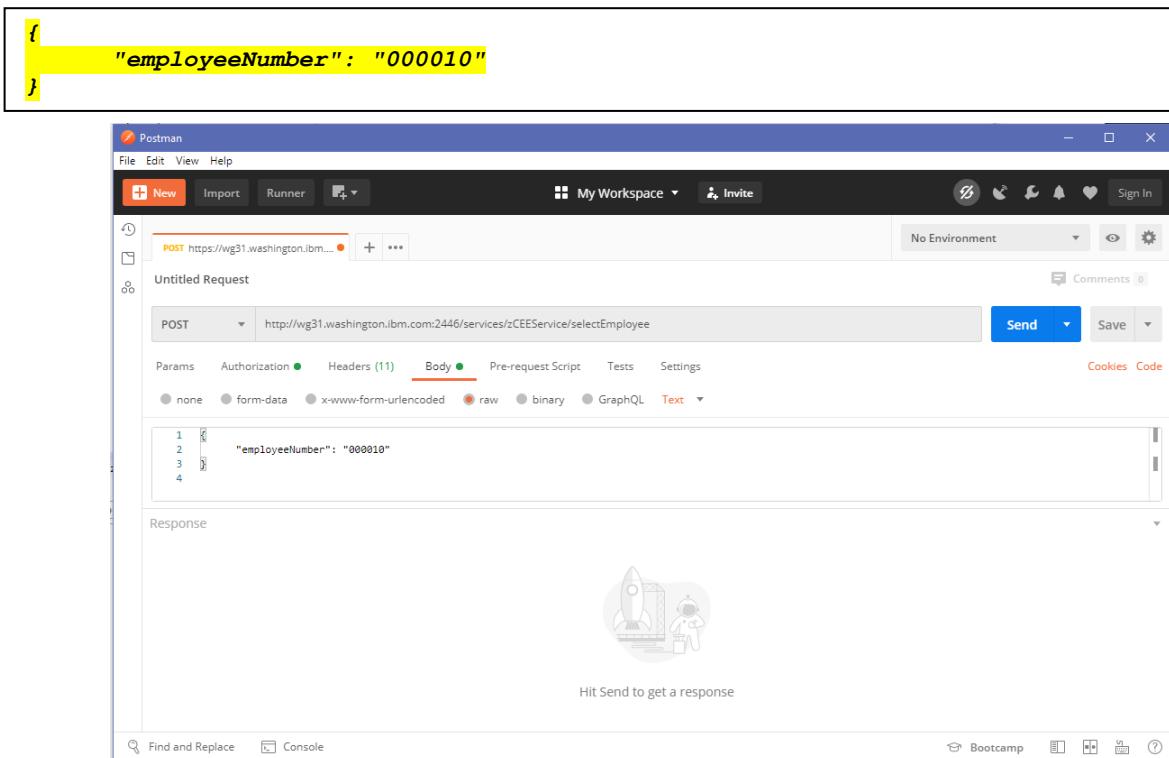
The screenshot shows the Postman interface with a POST request to <http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee>. The 'Authorization' tab is selected. A dropdown menu under 'TYPE' shows 'Basic Auth' highlighted and circled in red. The 'Username' field contains 'USER1' and the 'Password' field contains '.....'. There is a 'Show Password' checkbox. The status bar at the bottom says 'Hit Send to get a response'.

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

The screenshot shows the Postman interface with a POST request to <http://wg31.washington.ibm.com:2446/services/zCEEService/selectEmployee>. The 'Headers' tab is selected. A table shows a row with 'Key' set to 'Content-Type' (with a checked checkbox) and 'Value' set to 'application/json'. Both the 'Key' and 'Value' fields are circled in red. The status bar at the bottom says 'Hit Send to get a response'.

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

6. Next select the *Body* tab and select the *raw* radio button and enter the JSON message below in the *Body* area and press the **Send** button.



The screenshot shows the Postman interface with a raw JSON request body:

```

t
"employeeNumber": "000010"
j

```

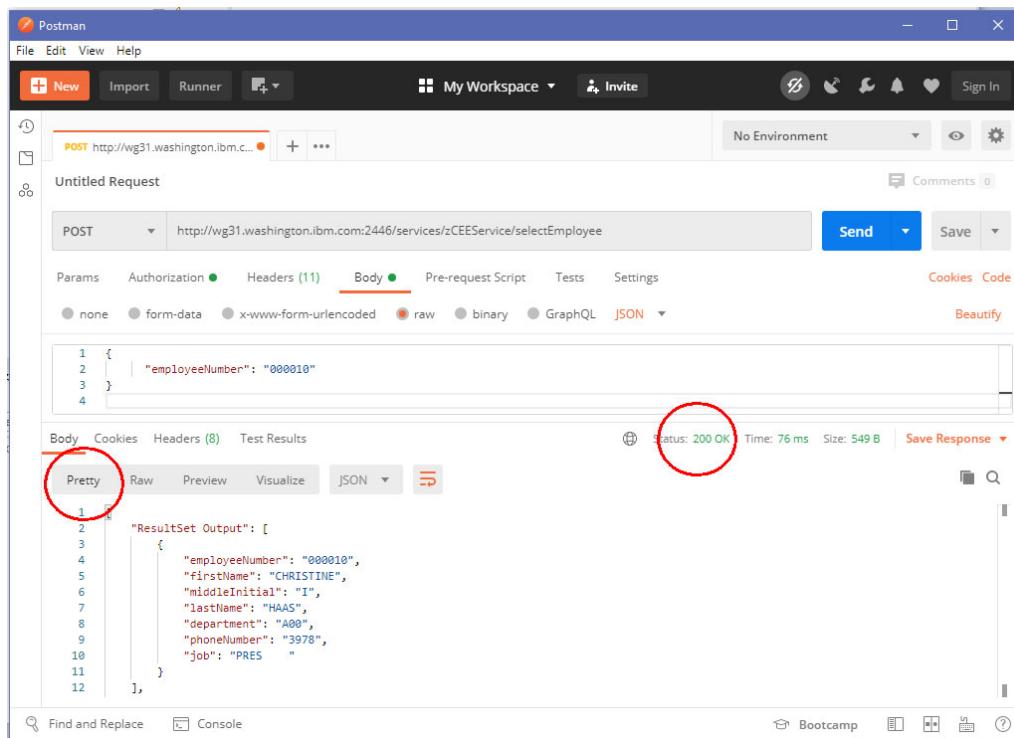
The Body tab is selected, and the raw radio button is chosen. The JSON content is:

```

1 {
2   "employeeNumber": "000010"
3 }
4

```

7. Pressing the **Send** button invokes the API. The Status of request should be *200 OK* and pressing the *Pretty* tab will display the response



The screenshot shows the Postman interface after sending the request. The status is 200 OK. The Pretty tab is highlighted with a red circle.

The response body is:

```

1 {
2   "employeeNumber": "000010"
3 }
4

```

The response details show:

- Status: 200 OK
- Time: 76 ms
- Size: 549 B
- Save Response

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

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

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

```

This creates a user Db2 native REST Service named *deleteEmployee* that deletes a row from table *USER1.EMPLOYEE* using the same JSON request message used in Step 6. Optionally test this service by using same Postman session with URL

<http://wg31.washington.ibm.com:2446/services/zCEEService/deleteEmployee> and the JSON request message below.

```
t
  "employeeNumber": "000340"
j
```

You should see this result in the response area.

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

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

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

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

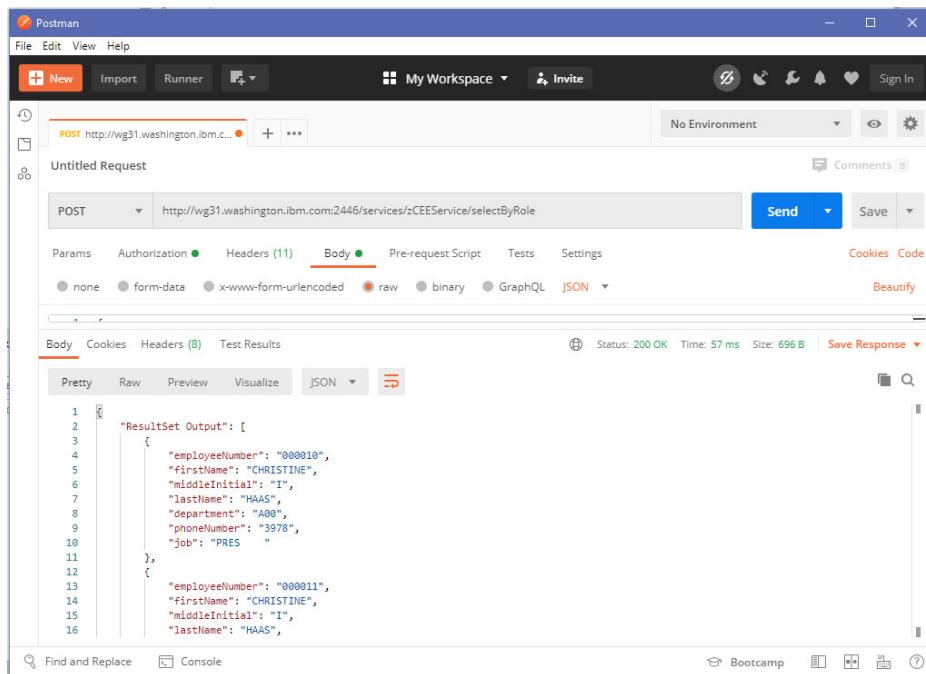
```

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

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

10. Test this service by using a URL of <http://wg31.washington.ibm.com:2446/services/zCEEService/selectByRole> and a JSON request message of:

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



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

```

//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DSN1210.DB2.SDSNEXIT,DISP=SHR
//          DD DSN=DSN1210.DB2.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSSOUT=*
//SYSPRINT DD SYSSOUT=*
//SYSUDUMP DD SYSSOUT=*
//DSNSTMT DD *
      INSERT INTO USER1.EMPLOYEE
        (EMPNO,FIRSTNAME,MIDINIT,LASTNAME,WORKDEPT,PHONENO,
         HIREDATE,JOB,EDLEVEL,SEX,BIRTHDATE,SALARY,BONUS,COMM)
      VALUES (:employeeNumber, :firstName, :middleInitial, :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 creates a user Db2 native REST Service named *insertEmployee* that inserts a new row into table USER1.EMPLOYEE).

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

12. Test this service by using a URL of

<http://wg31.washington.ibm.com:2446/services/zCEEService/insertEmployee>

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

You should see this result in the response area.

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

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

```
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//DSNSTMT DD *
  UPDATE USER1.EMPLOYEE
    SET SALARY = :salary, BONUS = :bonus, COMM = :commission
    WHERE EMPNO = :employeeNumber
//SYSTSIN DD *
DSN SYSTEM(DSN2)
BIND SERVICE("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')
```

This completes the creation of the Db2 native REST Services that will be used in an API. In the next section, the OpenAPI 3 specification document will be imported into the *z/OS Connect Designer* and used to develop an API.

But before we continue; go back and review the Db2 response messages. Notice that there is a pattern in the response messages from the Db2 REST services. When a Db2 resources 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:
      - 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.

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="zCERRoles">
    <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 *USER2* 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 will receive an HTTP 403 (Forbidden) response if they try to invoke one of these privileged methods.

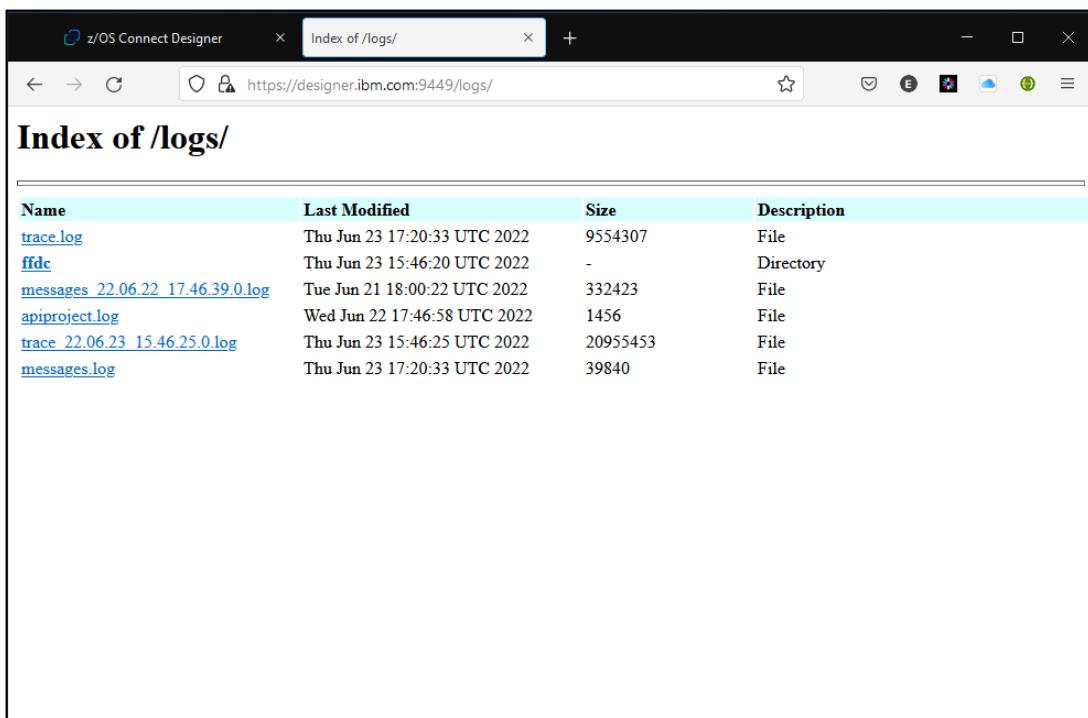
```
openapi: 3.0.0
x-ibm-zcon-roles-allowed:
- Manager
security:
- BasicAuth: []
- BearerAuth: []
paths:
  "/roles/{job}":
    get:
      x-ibm-zcon-roles-allowed:
        - Staff
  /employees:
    post:
  /employees/details/{employee}:
    get:
      x-ibm-zcon-roles-allowed:
        - Staff
  "/employees/{employee}":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

Accessing the z/OS Connect Designer log and trace files

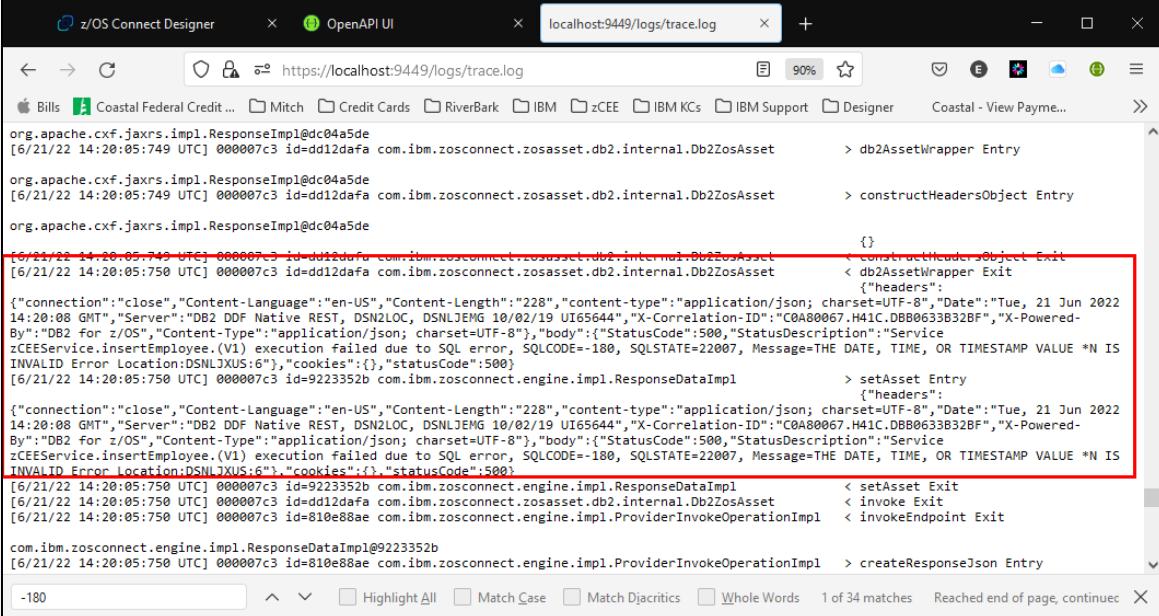
The Liberty server in which the z/OS Connect Designer has been further customized with the addition of the server XML configuration elements below. These XML configuration elements enables the Liberty server to become a file server.

```
<?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-servering-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-servering-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.



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.



The screenshot shows a browser window with the URL <https://localhost:9449/logs/trace.log>. The page displays a log of events from the z/OS Connect Designer container. A specific error entry is highlighted with a red box:

```
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > db2AssetWrapper Entry
[6/21/22 14:20:05:749 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset > constructHeadersObject Entry
[6/21/22 14:20:05:749 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
[6/21/22 14:20:05:750 UTC] 000007c3 id=dd12dafa com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset {"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, DSNLJEMG 10/02/19 U165644", "X-Correlation-ID": "C0A80067.H41C.DB80633B32BF", "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
[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 > createResponseJson Entry
[6/21/22 14:20:05:750 UTC] 000007c3 id=810e88ae com.ibm.zosconnect.engine.impl.ProviderInvokeOperationImpl > createResponseJson Exit
```

The error message indicates that the `zCEEService.insertEmployee.(V1)` operation failed due to a SQL error (SQLCODE=-180, SQLSTATE=22007), with the message being "THE DATE, TIME, OR TIMESTAMP VALUE *N IS INVALID".

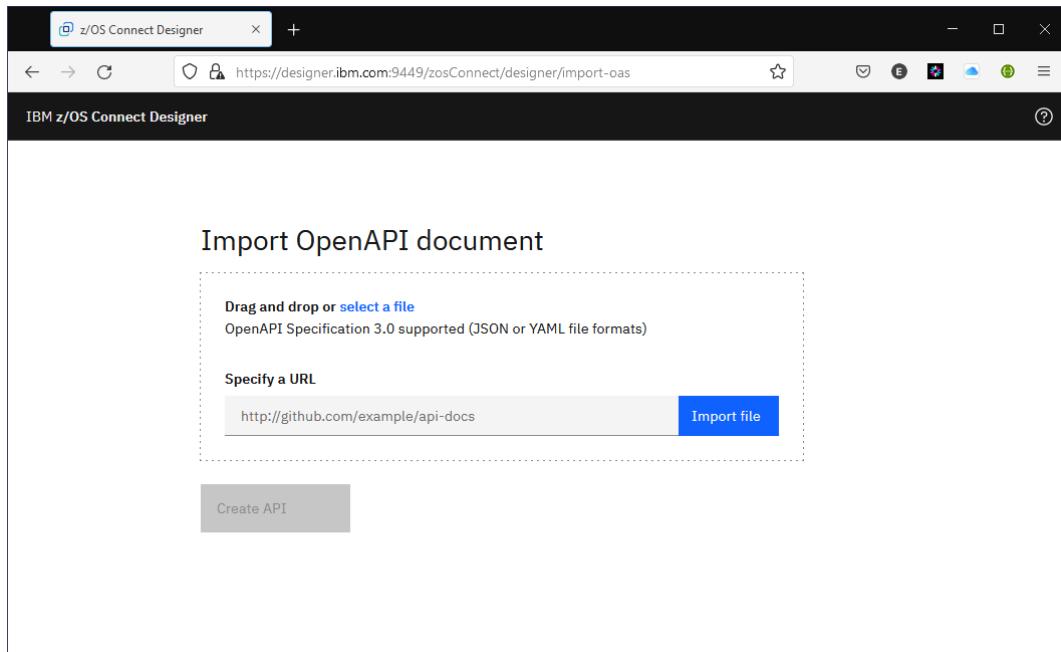
Developing a z/OS Connect APIs that accesses Db2

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

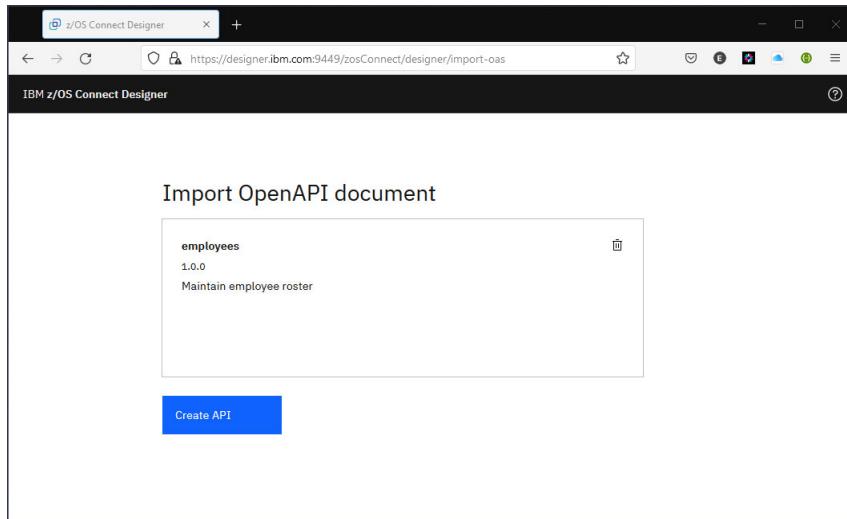
1. Start by opening the Firefox browser and going to URL

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

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

IBM z/OS Connect (OpenAPI 3.0)

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).

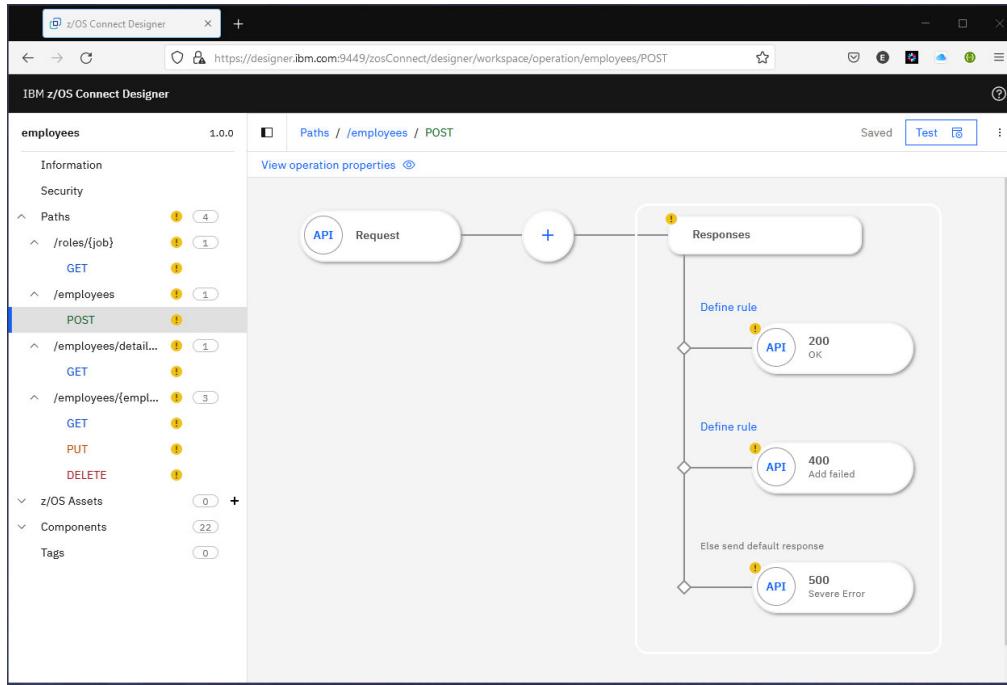
The screenshot shows the IBM z/OS Connect Designer application window. The left sidebar displays a tree view of API endpoints under the 'employees' resource. The 'Paths' section is expanded, showing various HTTP methods (GET, POST, PUT, DELETE) and their corresponding URI patterns. Some items in the tree have yellow exclamation marks next to them, indicating incomplete configuration. The right panel contains detailed configuration sections for the selected endpoint, including 'General' (Title: employees, Version: 1.0.0, Description: Maintain employee roster), 'Contact' (Name, Email, URL), 'License' (Name, URL), and 'Servers' (URL: /, Description). A 'Test' button is visible at the top right of the main panel.

*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.

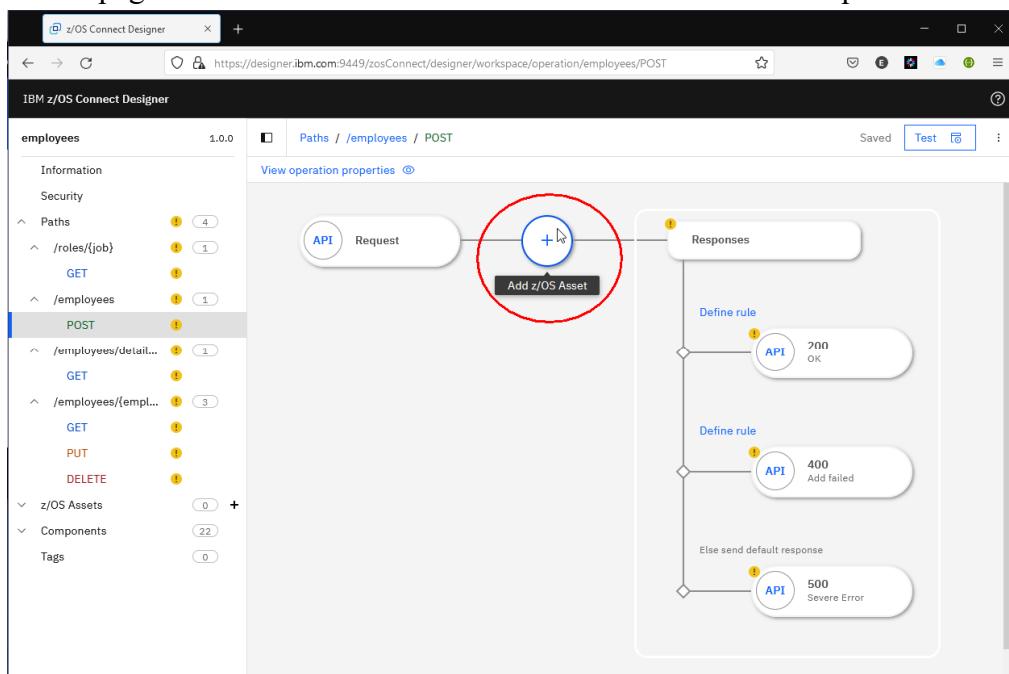
Configure the POST method for URI path /employees

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.

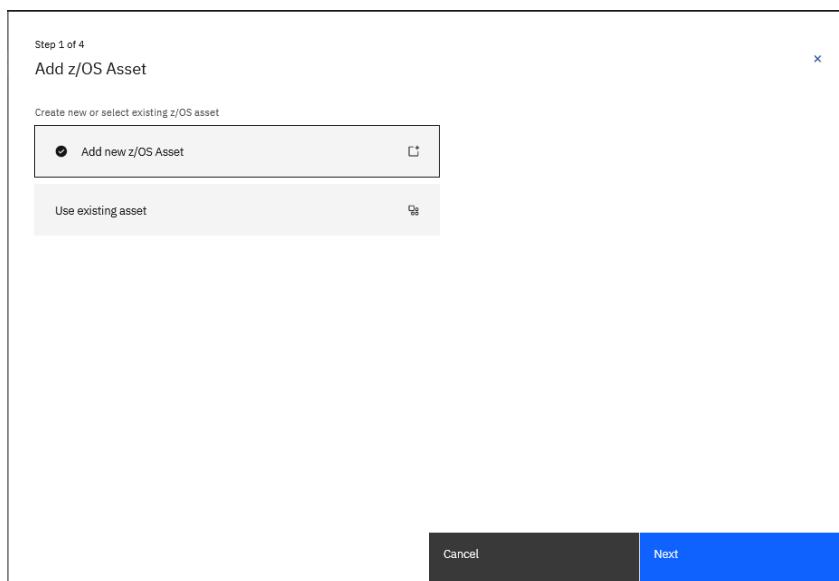


In the example above, we see that the specification document defined 3 responses for this method. One is a 200-status code which indicate the invocation of the method (an insert) was successful. A 400-status code which indicates, in this case, that the request to insert an employee record failed. And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

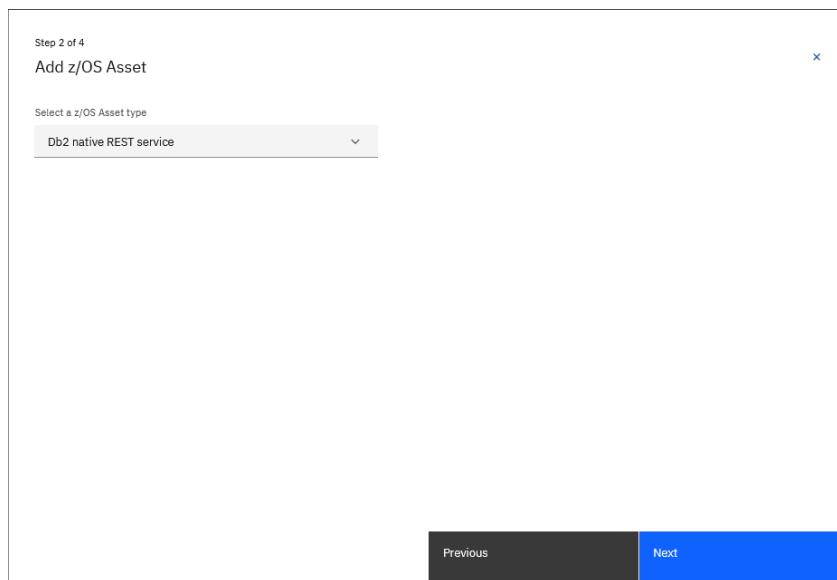
2. The first step in configuring this method for this URI path is to associate it with a z/OS asset or resource. Click the plus sign on the page to start the association of a z/OS asset with this URI path.



___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.



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

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 has...	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.

The screenshot shows a list of 12 Db2 native REST services found. The 'insertEmployee' service in the 'zCEEService' collection is highlighted with a red circle. The 'Import' button at the bottom right is also highlighted in blue.

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/...	Delete an employee fro...	Available
displayEmployee	V1	zCEEService	/services/zCEEService/...	Display an employee in ...	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

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.

The screenshot shows the 'Step 3 of 4' page of the 'Add z/OS Asset' wizard. The 'Import from Db2 service manager' button is highlighted in blue. The imported service details are displayed in the form fields:

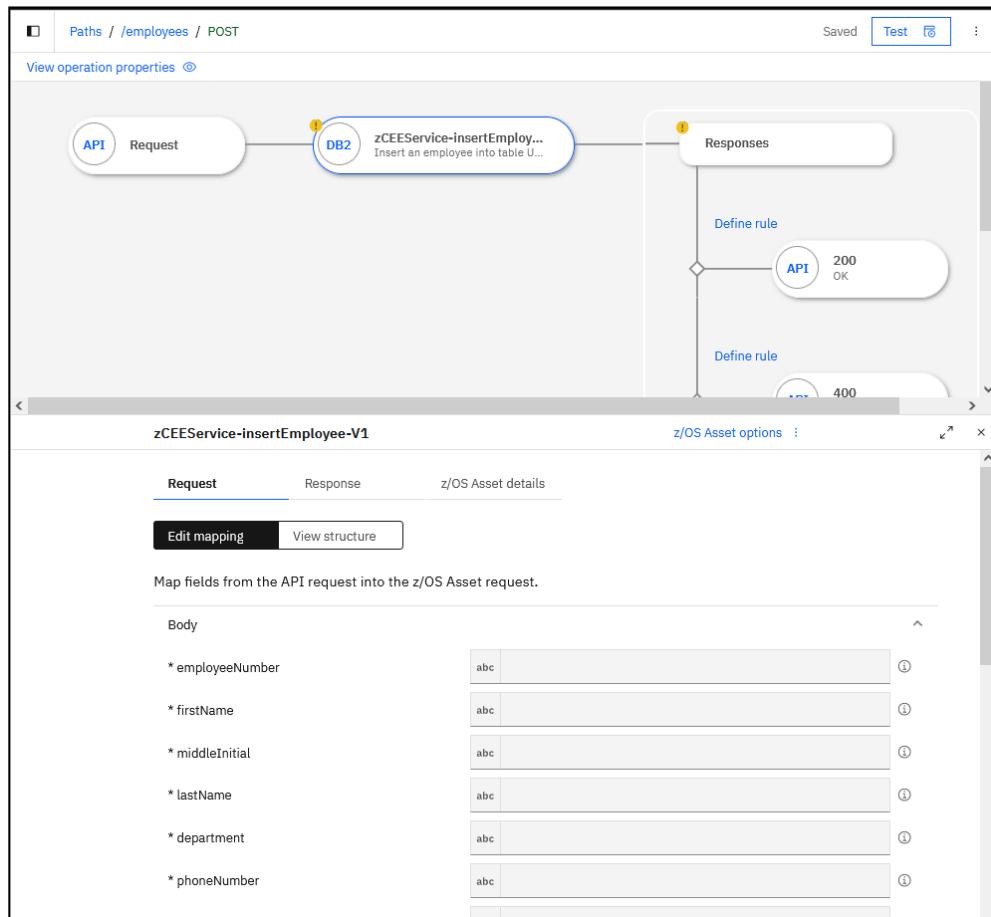
- Db2 native REST service collection ID: zCEEService
- Db2 native REST service name: insertEmployee
- Db2 native REST service version (optional): V1
- Import Db2 native REST service request schema (JSON snippet):

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

- Import Db2 native REST service response schema (JSON snippet):

```
{
  "$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": "..."
  }
}
```

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.



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.

Request	Response	z/OS Asset details													
Edit mapping	View structure														
Map fields from the API request into the z/OS Asset request.															
<table border="1"> <thead> <tr> <th>Body</th> </tr> </thead> <tbody> <tr> <td>* employeeNumber</td> <td>abc</td> </tr> <tr> <td>* firstName</td> <td>abc</td> </tr> <tr> <td>* middleInitial</td> <td>abc</td> </tr> <tr> <td>* lastName</td> <td>abc</td> </tr> <tr> <td>* department</td> <td>abc</td> </tr> <tr> <td>* phoneNumber</td> <td>abc</td> </tr> </tbody> </table>			Body	* employeeNumber	abc	* firstName	abc	* middleInitial	abc	* lastName	abc	* department	abc	* phoneNumber	abc
Body															
* employeeNumber	abc														
* firstName	abc														
* middleInitial	abc														
* lastName	abc														
* department	abc														
* phoneNumber	abc														

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.

The screenshot shows the z/OS Asset options interface for the zCEEService-insertEmployee-V1 API. The Request tab is selected. Below it, there is a section titled "Map fields from the API request into the z/OS Asset request." This section contains a table where API fields are mapped to z/OS Asset fields. The columns are "Body" and "z/OS Asset". The rows are:

Body	z/OS Asset
* employeeNumber	abc <input type="text" value="employeeNumber"/> ⓘ
* firstName	abc ⓘ
* middleInitial	abc ⓘ
* lastName	abc ⓘ
* department	abc ⓘ
* phoneNumber	abc ⓘ

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).

The screenshot shows the 'zCEEService-insertEmployee-V1' interface. The 'Edit mapping' tab is active. In the 'Body' section, there are fields for 'employeeNumber', 'firstName', 'middleInitial', 'lastName', 'department', and 'phoneNumber'. The 'firstName' field has a red circle around its mapping icon (a blue square with a white plus sign). The 'employeeNumber' field also has a red circle around its mapping icon.

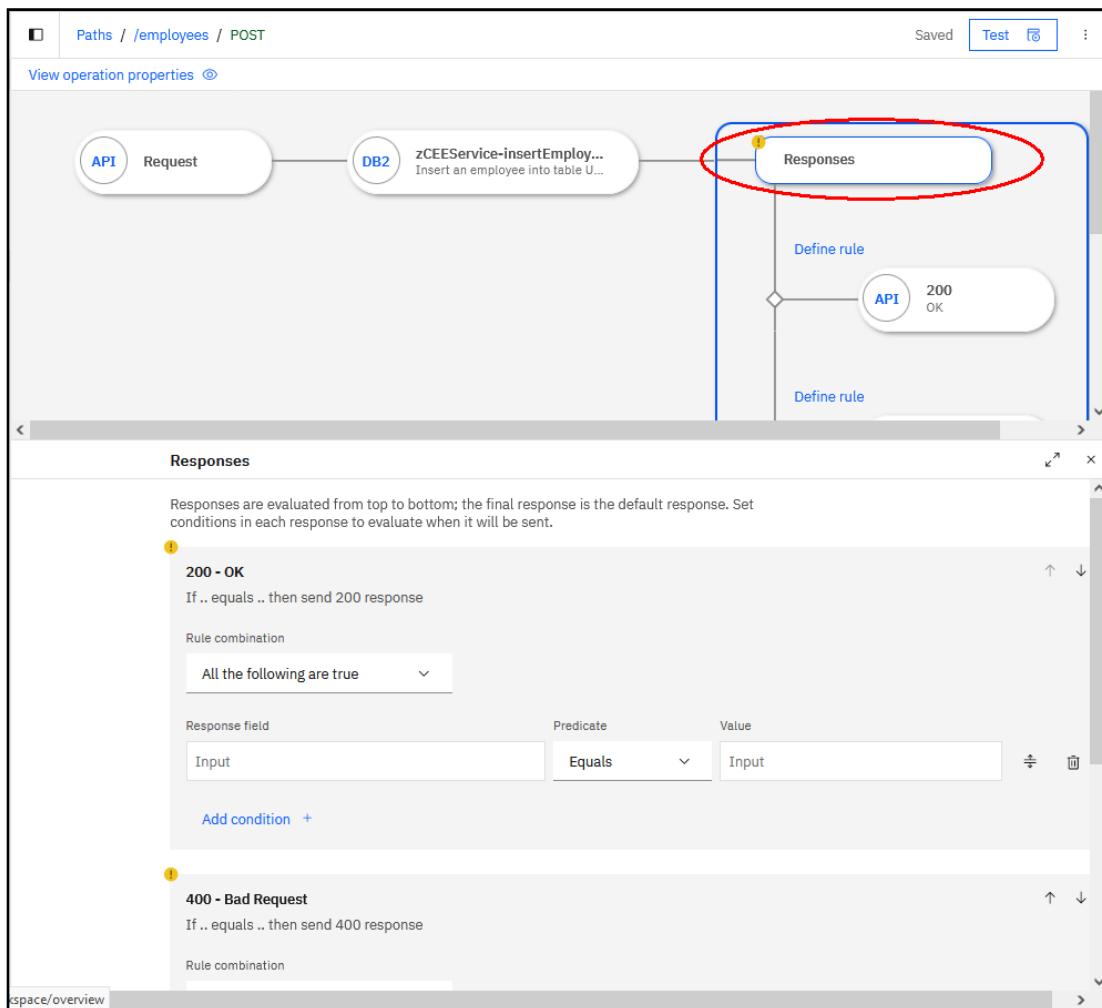
15. This will display a list of available mapping fields. Since this is a request message and the fields are in the request *body*. Scroll up or down and choose appropriate field from the fields from the *body*, not a query parameter, nor a path parameter. In this case, the field to select is the *firstName*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

The screenshot shows the 'zCEEService-insertEmployee-V1' interface. The 'Edit mapping' tab is active. In the 'Body' section, there are fields for 'employeeNumber', 'firstName', 'middleInitial', 'lastName', 'department', and 'phoneNumber'. The 'firstName' field is highlighted with a red box. A dropdown menu titled 'Available mappings' is open, showing a list of fields under 'body': 'Object', 'employeeNumber', 'firstName', 'middleInitial', and 'lastName'. The 'Object' item is expanded, showing an empty object structure. The 'firstName' item is also highlighted with a red box.

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

Body	z/OS Asset
* employeeNumber	abc [API] employeeNumber
* firstName	abc [API] firstName
* middleInitial	abc [API] middleInitial
* lastName	abc [API] lastName
* department	abc [API] departmentCode
* phoneNumber	abc [API] phoneNumber
* hireDate	abc [API] dateOfHire
* job	abc [API] job
* educationLevel	123 [API] educationLevel
* sex	abc [API] sex
* birthDate	abc [API] dateOfBirth
* salary	123 [API] salary
* bonus	123 [API] lastBonus
* commission	123 [API] lastCommission

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.



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.

The screenshot shows the 'Responses' configuration for a POST endpoint at `/employees`. The interface is divided into three main sections corresponding to different HTTP status codes:

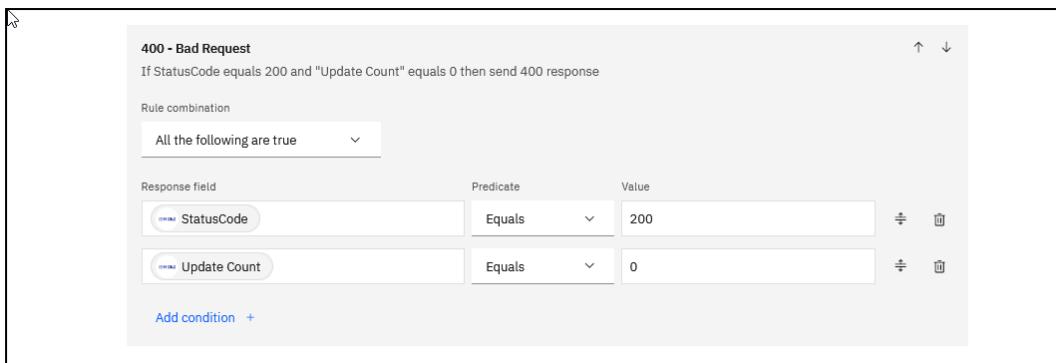
- 200 - OK:** This section contains a condition: "If .. equals .. then send 200 response". Below it, there is a "Rule combination" dropdown set to "All the following are true" and a table for defining response fields, predicates, and values. The table has one row: Response field "Input", Predicate "Equals", and Value "Input".
- 400 - Bad Request:** This section contains a condition: "If .. equals .. then send 400 response". Below it, there is a "Rule combination" dropdown set to "All the following are true" and a table for defining response fields, predicates, and values. The table has one row: Response field "Input", Predicate "Equals", and Value "Input".
- 500 - Internal Server Error:** This section contains the text "Else send default response".

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*.

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

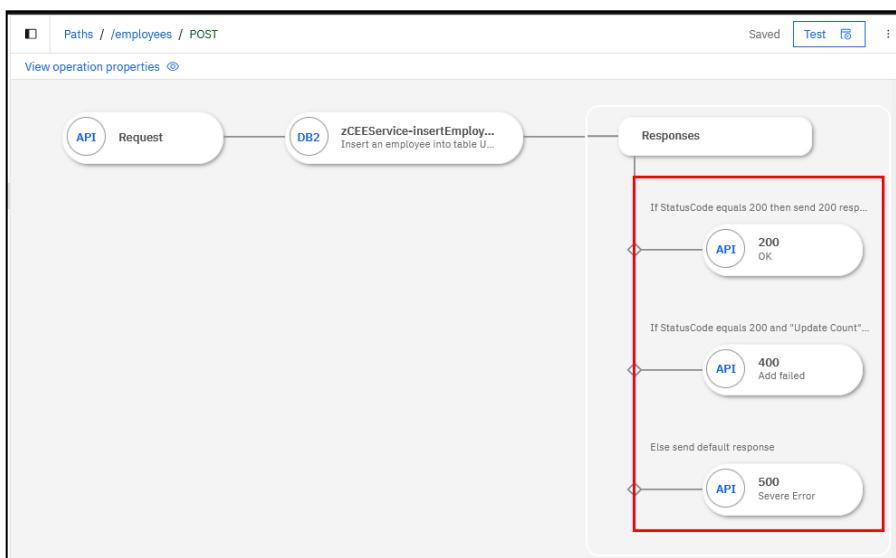
21. Use the same technique described above to add a check for response field *Update Count*. Leave the *Predicate* as *Equals* and set the value to ***1*** below:

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



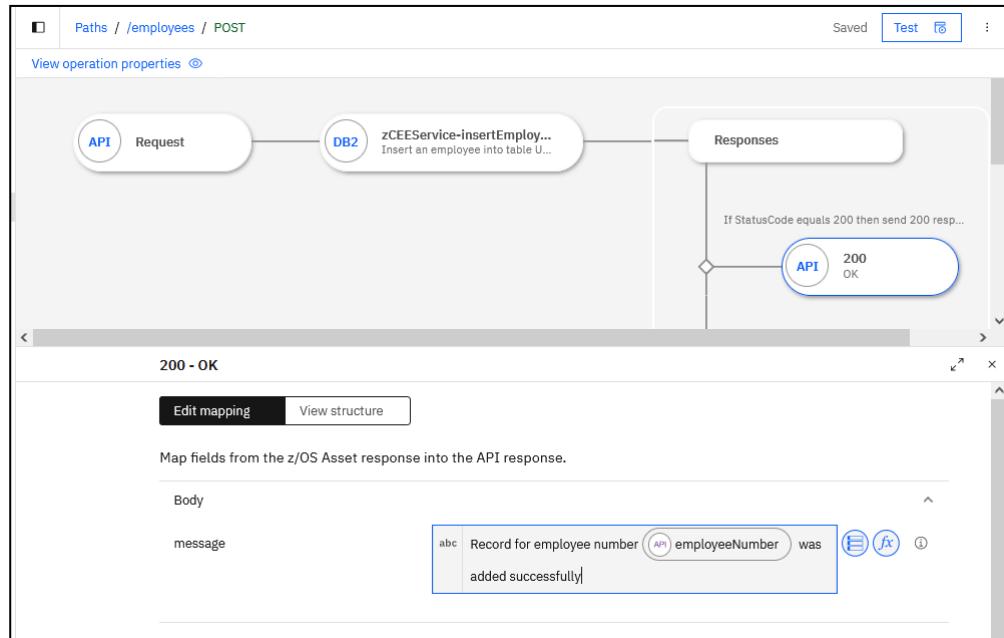
23. If neither of these connections are met, simply return with a HTTP 500 status code.

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.

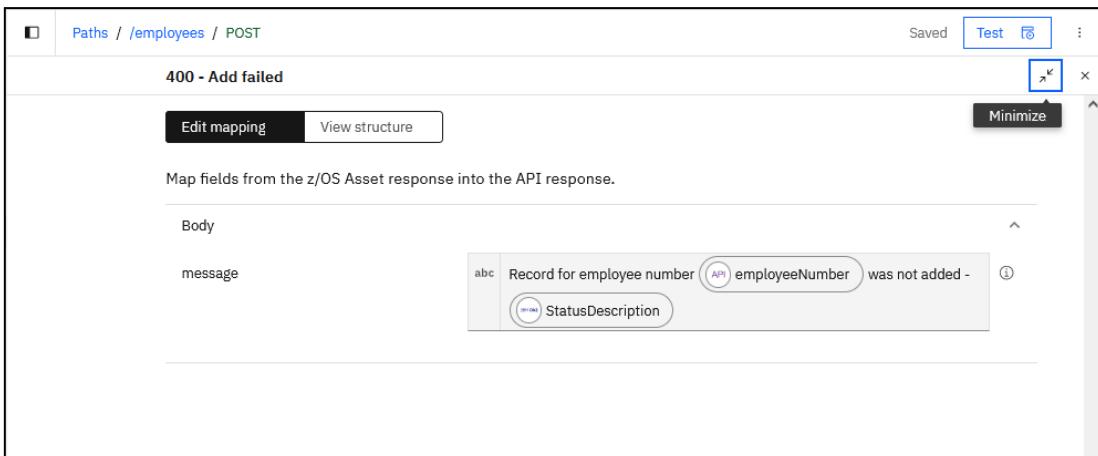
Record for employee number {{\\$apiRequest.body.employeeNumber}} was added successfully



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.

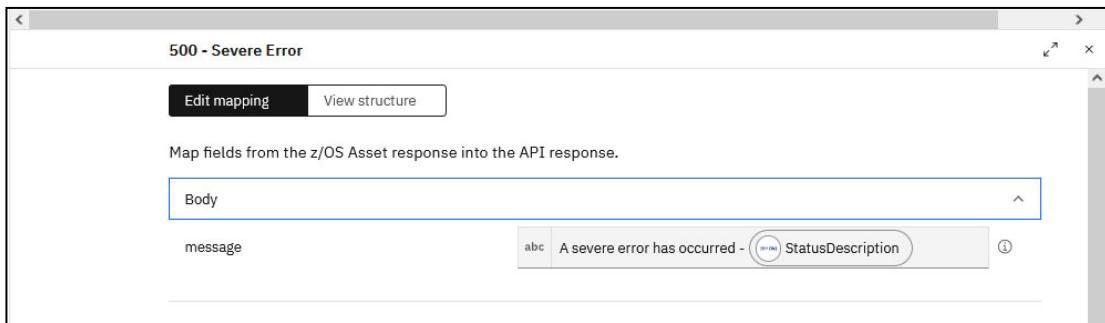
Record for employee number {{\\$apiRequest.body.employeeNumber}} was not added - {{\\$zosAssetResponse.body.StatusDescription}}



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}

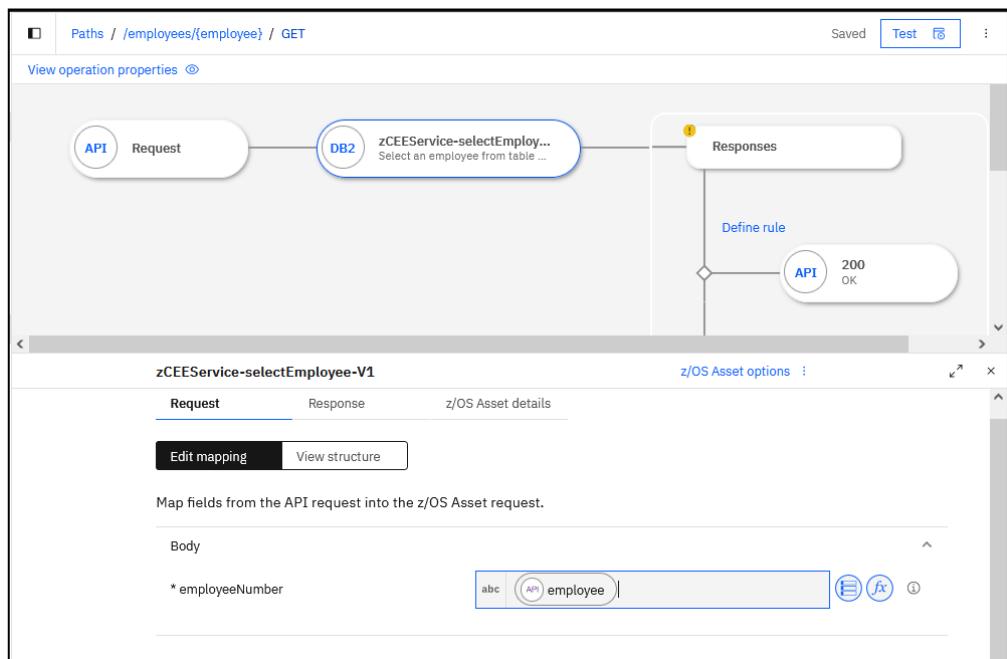


This completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

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' configuration for a specific API endpoint. The top bar indicates the path is '/employees/{employee}' and the method is 'GET'. The 'Test' button is highlighted. The 'Responses' section contains three entries:

- 200 - OK**: A condition where 'Input' equals 'Input'. The rule combination is 'All the following are true'. This condition is currently active.
- 404 - Not Found**: A condition where 'Input' equals 'Input'. The rule combination is 'All the following are true'.
- 500 - Internal Server Error**: A condition where 'Else send default response' is selected. This condition is currently inactive.

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(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

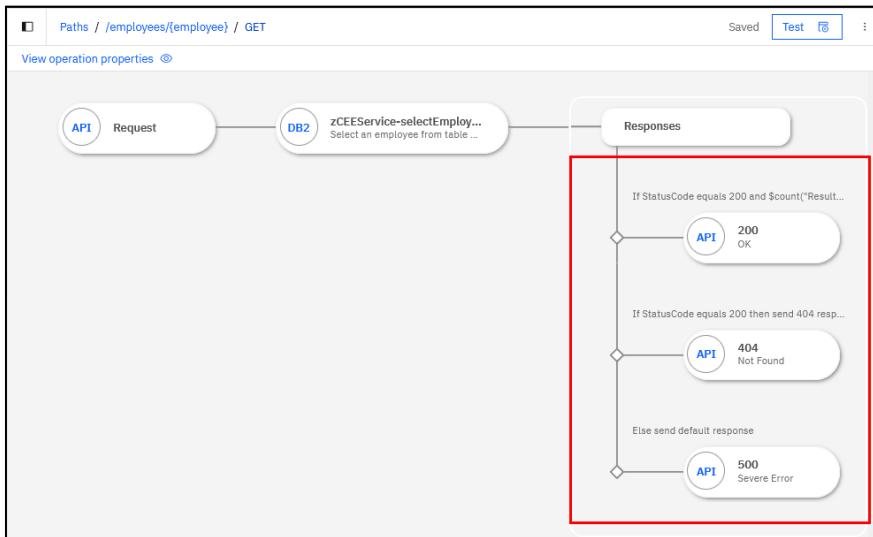
Rule combination
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
count(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.

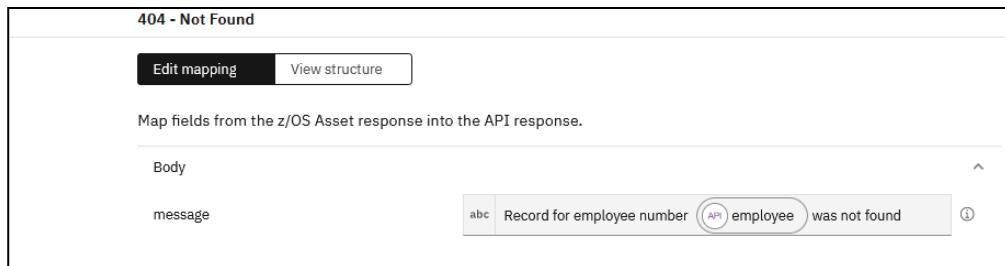
The screenshot shows the 'Body' section of the mapping interface. The 'zosAssetResponse / Object' node has a child node 'ResultSet Output'. This node is highlighted with a blue border, and a tooltip '[] Res' is displayed above the mapping area. Other properties listed include employeeNumber, name, departmentCode, phoneNumber, and job.

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*.

The screenshot shows the completed mapping. The 'zosAssetResponse / Object' node now has a child node 'ResultSet Output'. This node has several children: 'employeeNumber', 'name' (which contains 'firstName', 'middleInitial', and 'lastName'), 'departmentCode', 'phoneNumber', and 'job'. The 'Edit mapping' button is visible at the top left of the interface.

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

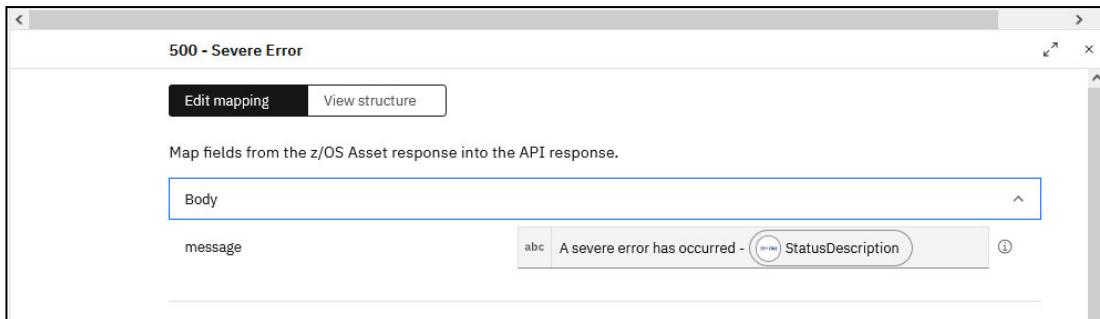
Record for employee number `>{{$apiRequest.pathParameters.employee}}` was not found



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.

The screenshot shows the Open Liberty API Explorer interface. At the top, it displays the URL <https://localhost:9449/api/explorer/>. Below the header, there's a search bar and a 'Filter' button. The main content area is titled 'Liberty REST APIs 1.0.0 OAS3'. Under the heading 'employee roster', there are five entries:

- GET /roles/{job}** Retrieve a list of employees based on job and department code
- POST /employees** Insert a new employee record into the employee roster
- GET /employees/details/{employee}** Display additional details of an employee record
- PUT /employees/{employee}** Update a subset of details of an employee record
- DELETE /employees/{employee}** Delete an employee record

Each method entry includes a lock icon and a dropdown arrow. The 'POST' method is highlighted with a green background, while the others are in blue, orange, and red boxes respectively. Below the 'employee roster' section, there's a collapsed section labeled 'employeeRos' under 'Schemas'.

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

The screenshot shows the z/OS Connect Designer interface with the 'OpenAPI UI' tab selected. The URL is https://localhost:9449/api/explorer/. The main content area displays the 'employee roster' API documentation. Under the 'POST /employees' endpoint, the 'Request body' section is expanded, showing a JSON schema for an employee record:

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "string",
  "lastName": "string",
  "departmentCode": "string",
  "phoneNumber": "string",
  "dateHire": "string",
  "job": "string",
  "educationLevel": 32767,
  "sex": "string",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 9999999.99,
  "lastCommission": 9999999.99
}
```

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

The screenshot shows the 'Try it out' dialog for the POST /employees endpoint. The 'Request body' section is expanded, showing the same JSON schema as the previous screenshot. Below the schema is a large text input field where the request body content can be entered. The 'Execute' button is at the bottom of the dialog.

4. 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
}
```

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.

The screenshot shows the IBM z/OS Connect API interface. At the top, there is a status bar with the URL `/employees` and a dropdown menu. Below the status bar is a search bar with a dropdown arrow. To the right of the search bar are two buttons: **Execute** (in blue) and **Clear**. Underneath these buttons is a section titled **Responses**. In the **Curl** section, the command to make a POST request to `https://localhost:9449/employees` is shown. The request includes headers for accept and content-type, and a JSON payload identical to the one in the question. In the **Request URL** section, the URL `https://localhost:9449/employees` is displayed. In the **Server response** section, the status code is 200, and the **Response body** contains the message: `{"message": "Record for employee number 948478 was added successfully"}`. There is a **Download** button next to the response body. Below the response body, the **Response headers** are listed, including `content-language: en-US`, `content-length: 100`, `content-type: application/json`, `date: Fri, 17 Jun 2022 17:15:13 GMT`, `x-firefox-spdy: h2`, and `x-powered-by: Servlet/4.0`.

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.

The screenshot shows the API interface with the following details:

- Curl:**

```
curl -X 'POST' \
  'https://localhost:9449/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://localhost:9449/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 EMPLOYEECA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS.          RID OF EXISTING ROW IS X'00000000229'. Error Location:DSNLJXUS:6"
}
```

Response headers:

```
content-language: en-US
content-length: 400
content-type: application/json
date: Fri,17 Jun 2022 17:18:19 GMT
x-firebase-spdy: h2
x-powered-by: Servlet/4.0
```

8. Scroll down and click on *GET /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.

The screenshot shows the API interface with the following details:

- Execute** button is visible at the top.
- Responses** section:
 - Curl:**

```
curl -X 'GET' \
  'https://localhost:9449/employees/948478' \
  -H 'accept: application/json'
```
 - Request URL:** <https://localhost:9449/employees/948478>
- Server response:**

Code	Details
200	Response body

Response body:

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

Response headers:

```
content-language: en-US
content-length: 133
content-type: application/json
date: Fri,17 Jun 2022 17:31:03 GMT
x-firebase-spdy: h2
x-powered-by: Servlet/4.0
```

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

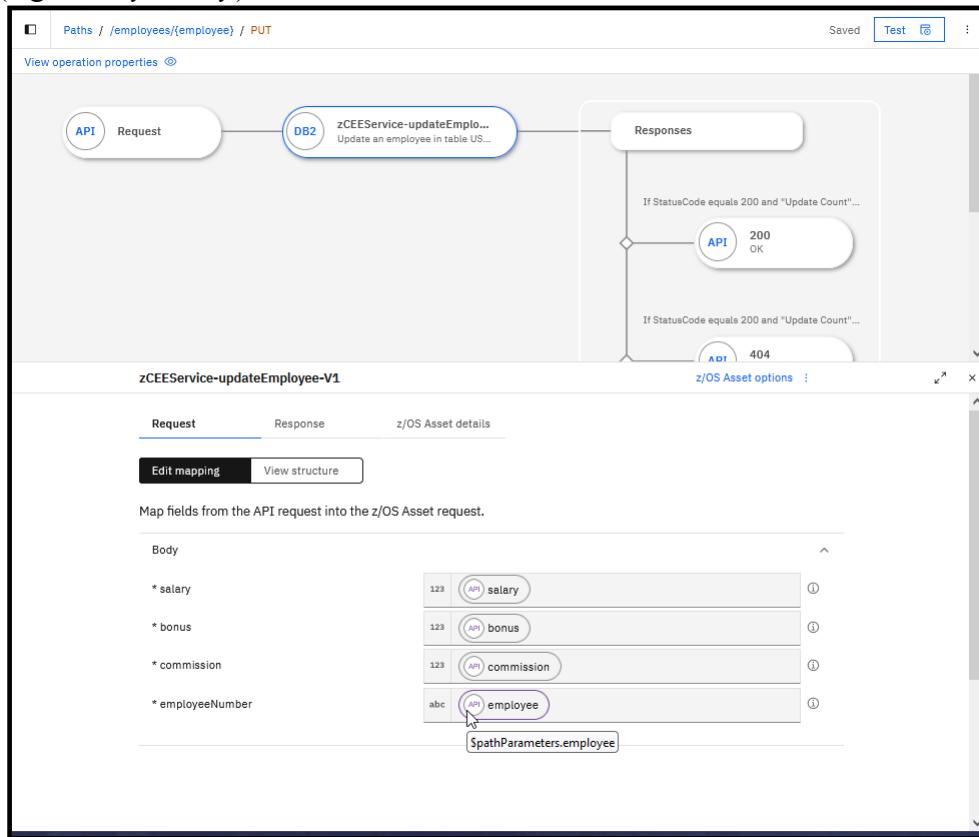
Compete the configuration of the API (Optional)

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-updateEmployee* 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' section of the API configuration interface. The top navigation bar includes 'Paths / /employees/{employee} / PUT', 'Saved', 'Test', and a 'Responses' tab. The main area is titled 'Responses' with the note: '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 .. equals .. then send 200 response
Rule combination: All the following are true
Response field: Input, Predicate: Equals, Value: Input
Add condition +

404 - Not Found
If .. equals .. then send 404 response
Rule combination: All the following are true
Response field: Input, Predicate: Equals, Value: Input
Add condition +

500 - Internal Server Error
Else send default response

At the bottom left, there is a link to 'xspace/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 *Status Code* field. Leave the *Predicate* as *Equals* and enter ***200*** in the *Value* field.

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 *Update Count* equaling zero.

404 - Not Found
If StatusCode equals 200 and "Update Count" equals 0 then send 404 response

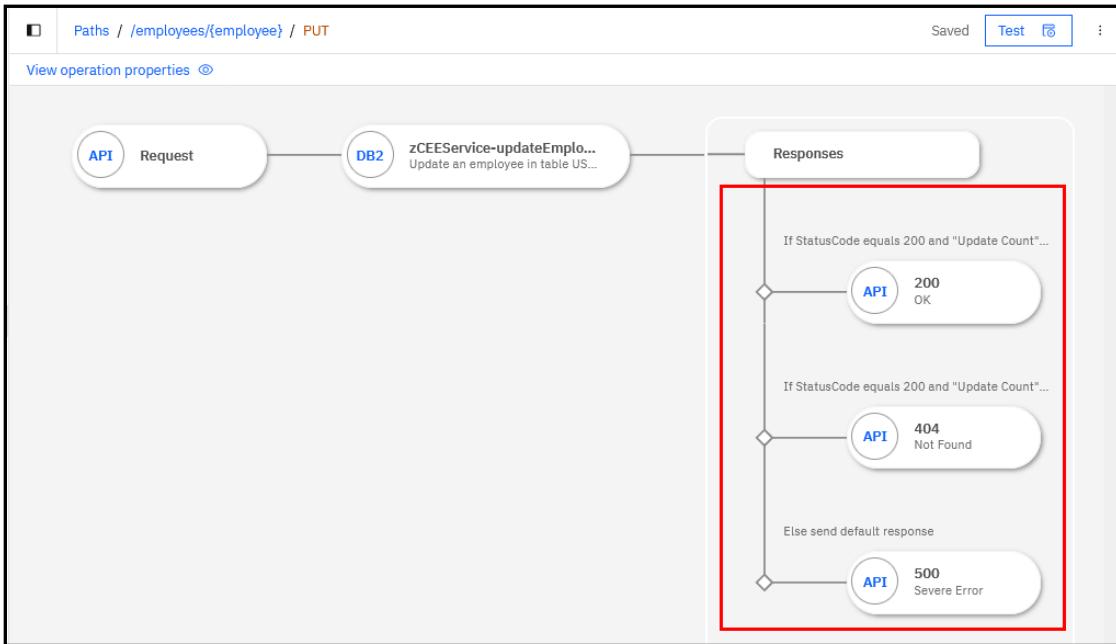
Rule combination
All the following are true

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

200 - OK

Edit mapping View structure

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

Body

message abc 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

404 - Not Found

Edit mapping View structure

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

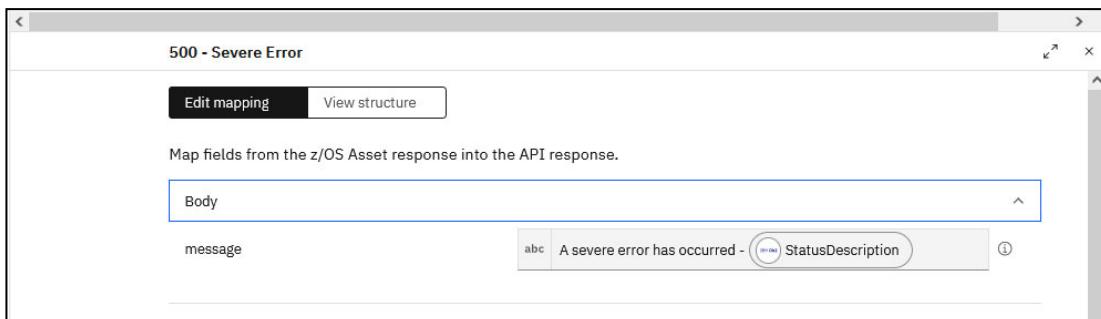
Body

message abc Record for employee number {{\\$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 - {\$zosAssetResponse.body.StatusDescription}

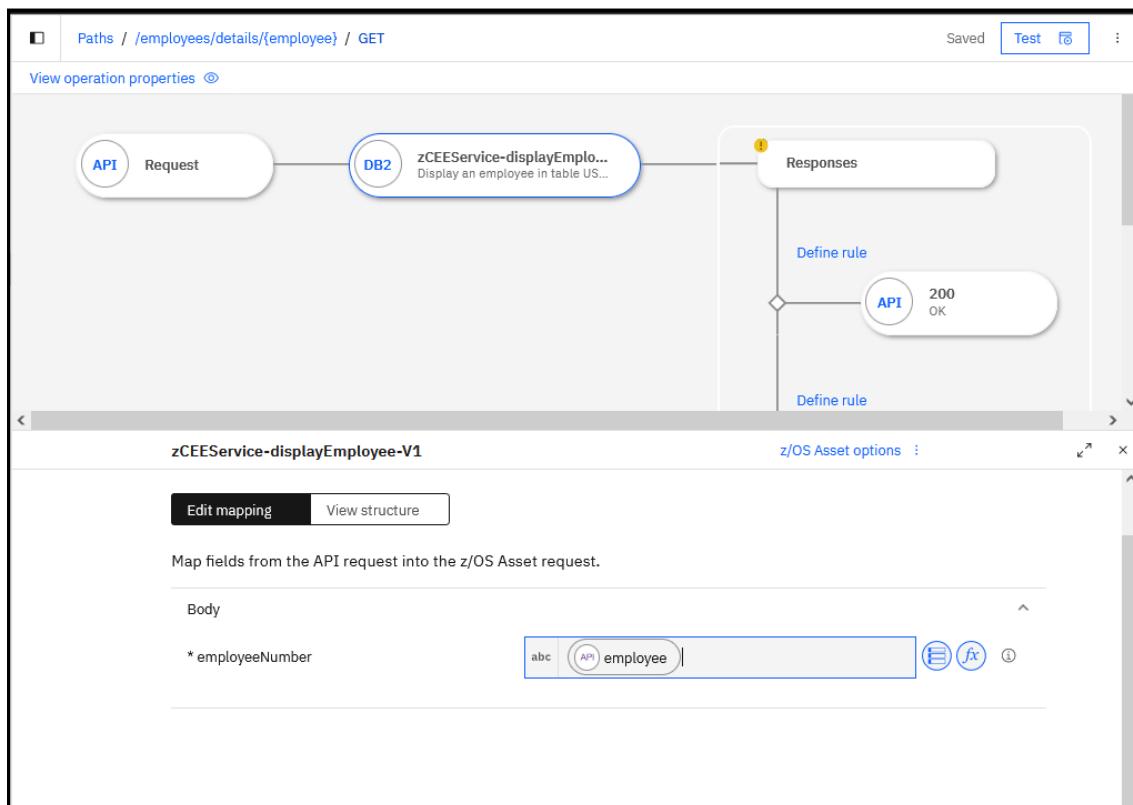


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

Configure the GET method for URI path /employees/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*.

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

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

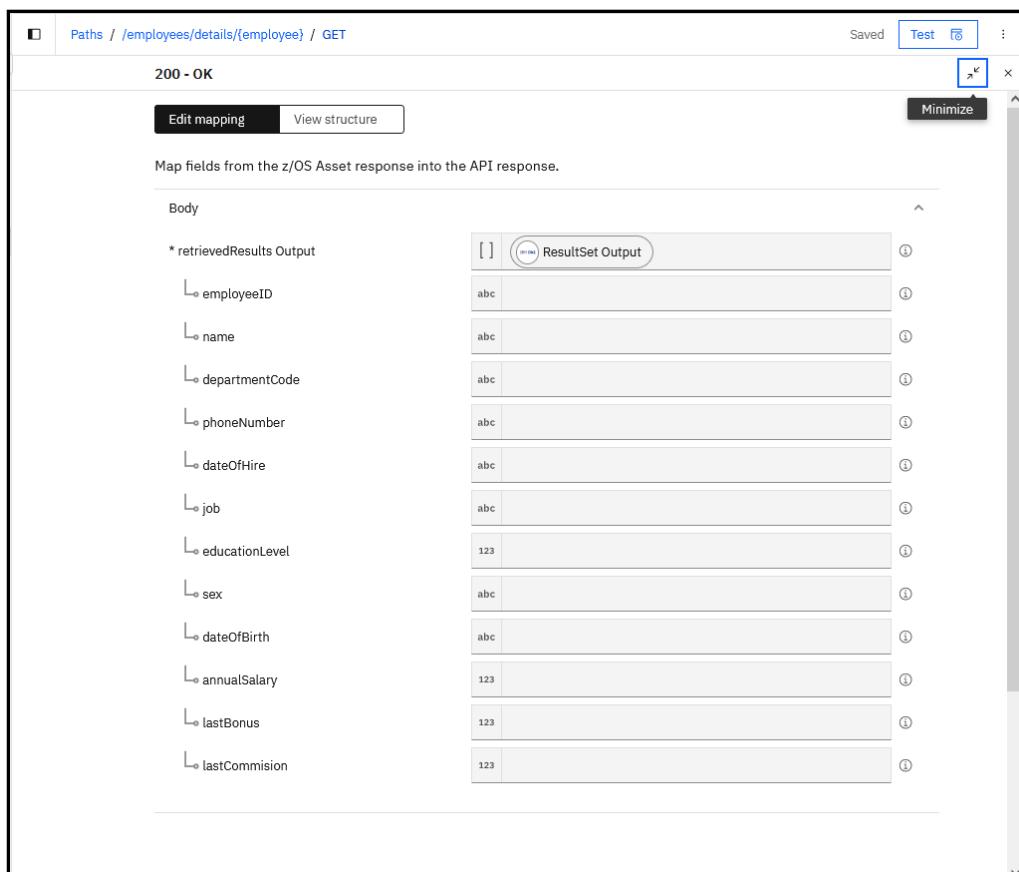
Response field	Predicate	Value
StatusCode	Equals	200
count(ResultSet Output)	Equals	0

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.



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 API mapping interface for a GET request to /employees/details/{employee}. The interface displays a tree view of fields under the 'Body' section and their corresponding mappings to 'ResultSet Output' fields.

DB2 Field	Mapping
* retrievedResults Output	[] (ResultSet Output)
employeeID	abc (EMPNO)
name	abc (FIRSTNME) (MIDINIT) (LASTNAME)
departmentCode	abc (WORKDEPT)
phoneNumber	abc (PHONENO)
dateOfHire	abc (HIREDATE)
job	abc (JOB)
educationLevel	123 (EDLEVEL)
sex	abc (SEX)
dateOfBirth	abc (BIRTHDATE)
annualSalary	123 (SALARY)
lastBonus	123 (BONUS)
lastCommission	123 (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 '404 - Not Found' response mapping configuration. At the top, there are 'Edit mapping' and 'View structure' buttons. Below them is a section titled 'Map fields from the z/OS Asset response into the API response.' Under 'Body', the text 'Record for employee number {{ \$apiRequest.pathParameters.employee }} was not found' is displayed. Below 'Body' is a 'message' field.

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}}`

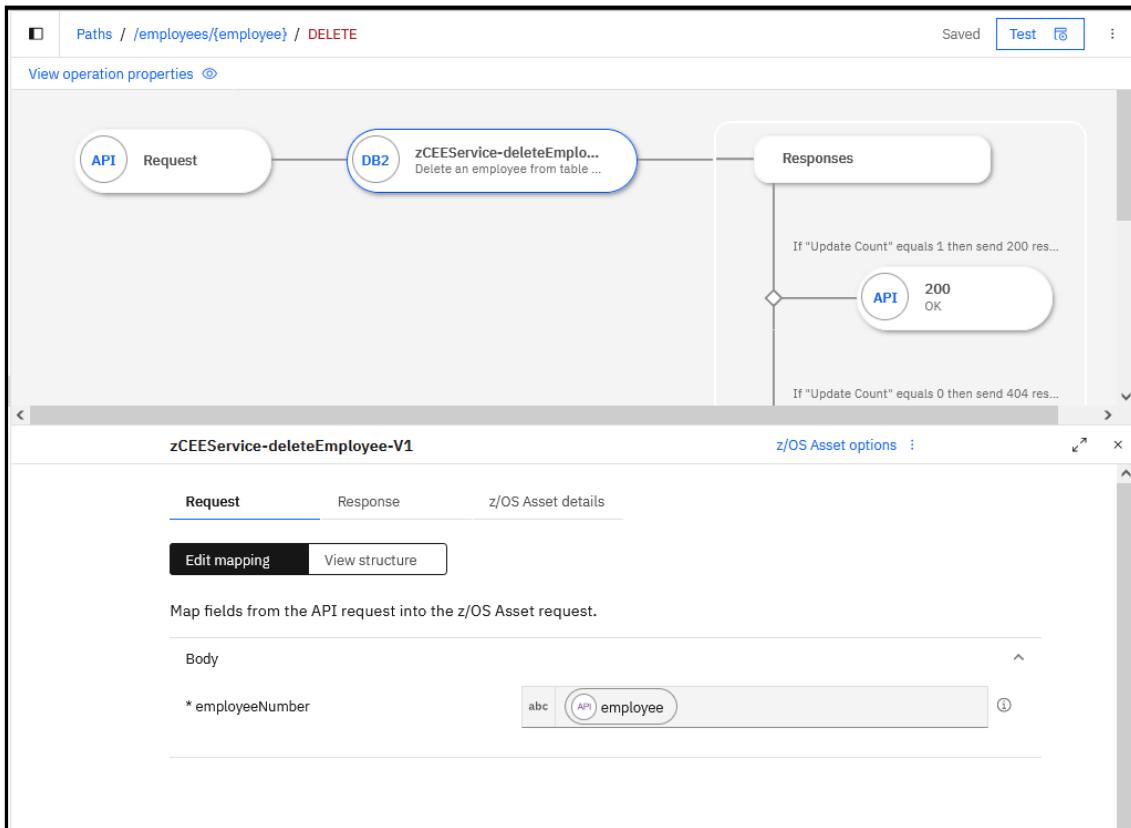
The screenshot shows the '500 - Severe Error' response mapping configuration. At the top, there are 'Edit mapping' and 'View structure' buttons. Below them is a section titled 'Map fields from the z/OS Asset response into the API response.' Under 'Body', the text 'A severe error has occurred - StatusDescription' is displayed. Below 'Body' is a 'message' field. The 'zosAssetResponse' component is highlighted with a blue oval.

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 *Value* 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 an update count equaling zero.

404 - Not Found
If StatusCode equals 200 and "Update Count" equals 0 then send 404 response

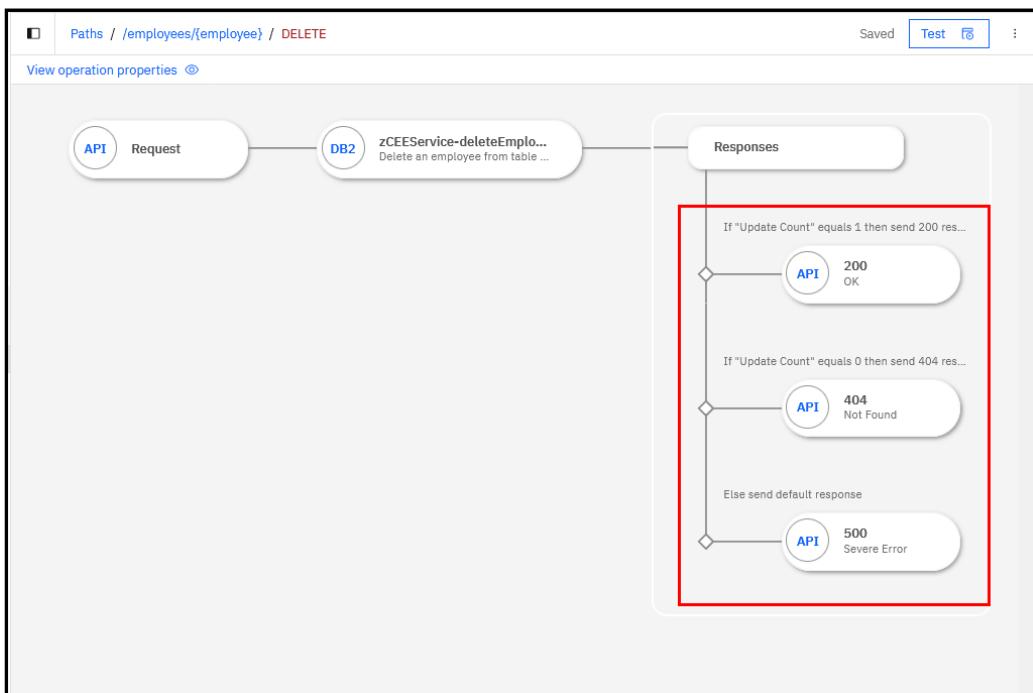
Rule combination
All the following are true

Response field	Predicate	Value
StatusCode	Equals	200
Update Count	Equals	0

Add condition +

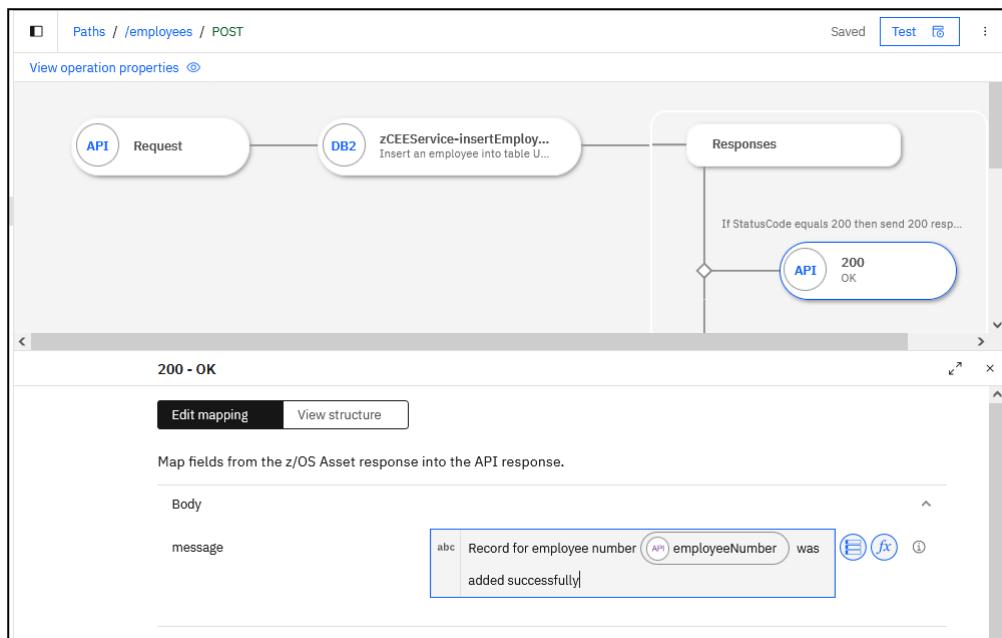
5. If neither of these connections are met, simply return with a HTTP 500 status code.

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



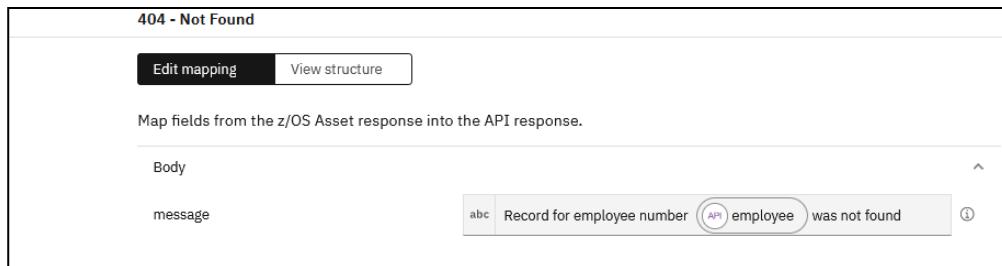
28. Select the response for *200 OK* paste the text below in the *message* area.

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.

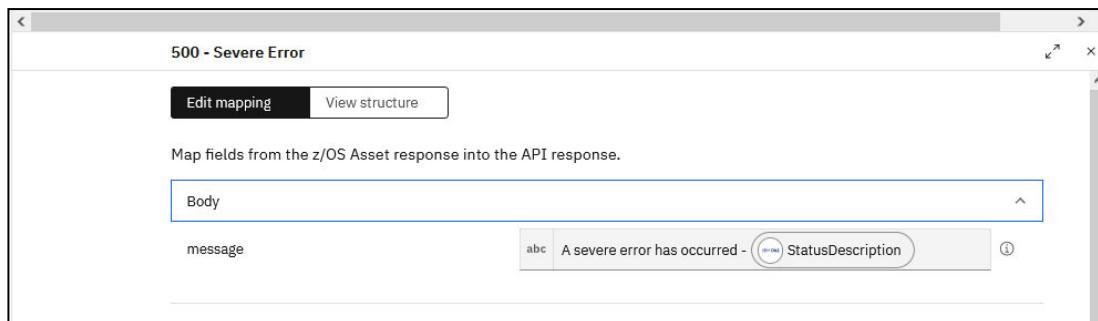
Record for employee number `>{{$apiRequest.pathParameters.employee}}` was not found



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}}`



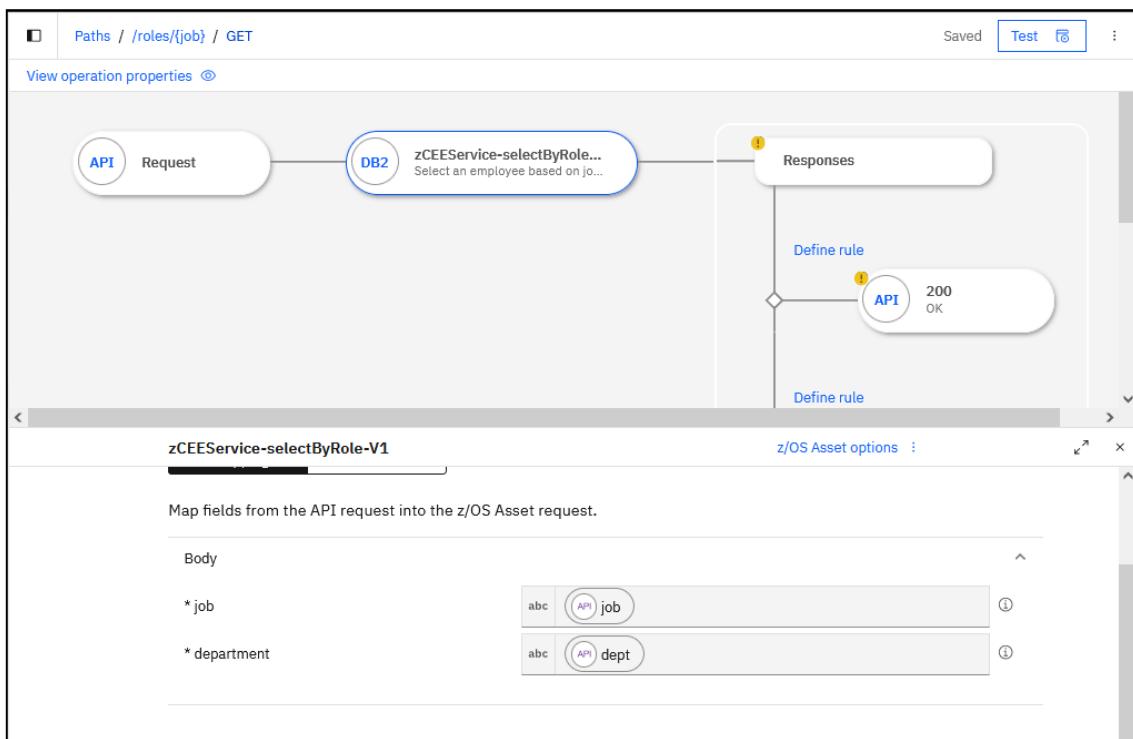
Tech-Tip: Db2 REST response property field *StatusDescription* provides more information regarding the issue that caused the insert to fail.

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*.

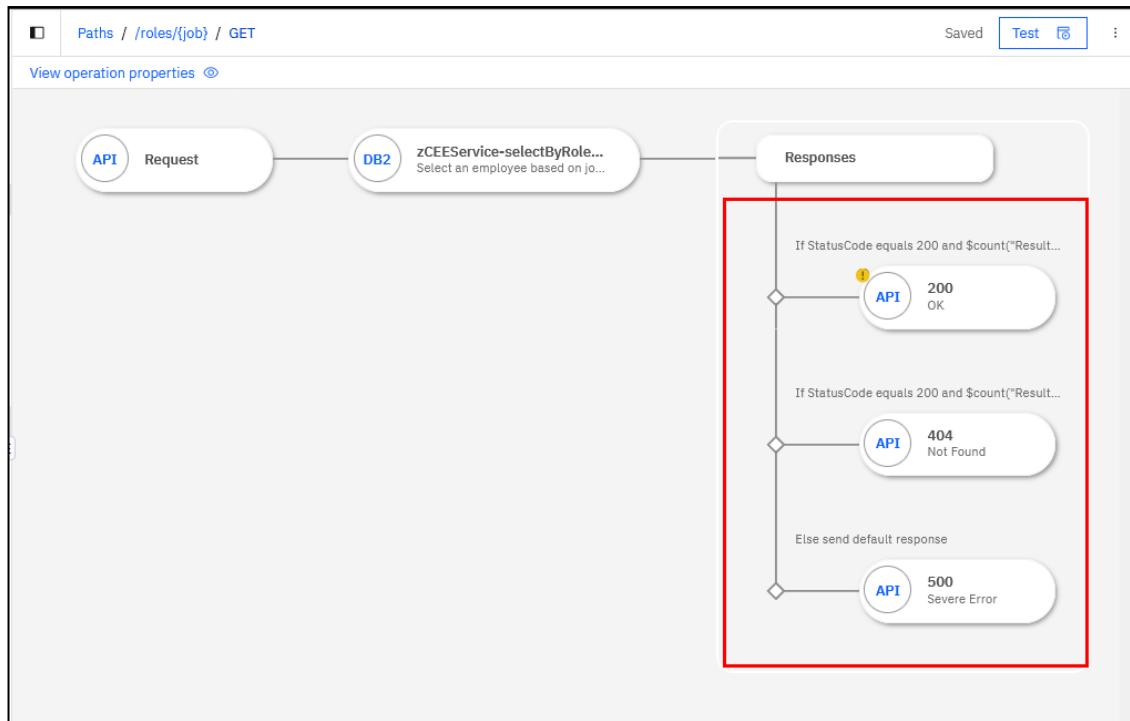
Response field	Predicate	Value
StatusCode	Equals	200
\$count(ResultSet Output)	Is greater than or equal to	1

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

Response field	Predicate	Value
StatusCode	Equals	200
count(ResultSet Output)	Equals	0

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.

The screenshot shows the mapping configuration for the `200 - OK` response. It has two tabs: `Edit mapping` (selected) and `View structure`. The `Edit mapping` tab displays the mapping rules for the `Body` section:

Path	Value
<code>* results Output</code>	<code>[]</code> <code>ResultSet Output</code>
<code> └* employeeNumber</code>	<code>abc</code>
<code> └* name</code>	<code>abc</code>
<code> └* department</code>	<code>abc</code>
<code> └* phoneNumber</code>	<code>abc</code>
<code> └* job</code>	<code>abc</code>

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 'Edit mapping' interface for a GET request to '/roles/{job}'. The 'Body' section contains the following mappings:

- * results Output: Mapped to ResultSet Output (array). Sub-elements include employeeNumber, name, department, phoneNumber, and job.
- employeeNumber: Mapped to employeeNumber.
- name: Mapped to abc (containing firstName, middleInitial, lastName).
- department: Mapped to department.
- phoneNumber: Mapped to phoneNumber.
- job: Mapped to job.

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

No records were found

404 - Not Found

Edit mapping View structure

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

Body

message abc No record were not found ⓘ

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}}

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 ⓘ

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 as long as 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. This section will demonstrate using common HTTP clients to test APIs specifically with 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/employee/{employee}>

The screenshot shows the Open Liberty API Explorer interface. At the top, it says "Liberty REST APIs 1.0.0 OAS3". Below that, it says "Discover REST APIs available within Liberty". Under "Servers", it lists "https://designer.ibm.com:9449". The main area displays the "employee roster" endpoint with the following methods:

- GET /roles/{job}** Retrieve a list of employees based on job and department code
- POST /employees** Insert a new employee record into the employee roster
- GET /employees/details/{employee}** Display additional details of an employee record
- GET /employees/{employee}** Display details of an employee record
- PUT /employees/{employee}** Update a subset of details of an employee record
- DELETE /employees/{employee}** Delete an employee record

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

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

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. Set *Authorization* type to basic and use *Fred* as the *Username* and *fredpwd* as the *Password* and add a header property of ***Content-Type*** with a value of *application/json*. Then use the down arrow in the *Body* tab to select **GET** and enter <https://designer.washington.ibm.com:9449/employees/details/000010> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

The screenshot shows the Postman interface with a successful API call. The URL in the request field is <https://designer.ibm.com:9449/employees/details/000010>. The response status is circled in red as 200 OK. The response body is a JSON object containing employee details:

```

1  {
2      "retrievedResults": [
3          {
4              "employeeID": "000010",
5              "name": "CHRISTINE I HAAS",
6              "departmentCode": "A00",
7              "phoneNumber": "3978",
8              "dateOfHire": "1965-01-01",
9              "job": "PRES",
10             "educationLevel": 18,
11             "sex": "F",
12             "dateOfBirth": "1933-08-14",
13             "annualSalary": 52750.0,
14             "lastBonus": 1000.0,
15             "lastCommission": 4220.0
16         }
    ]
}
  
```

2. Next enter an invalid employee number such as 121212,

<https://designer.washington.ibm.com:9449/employees/details/121212>

in the URL area (see below) and press **Send**. You should see results like below in the response Body area.

The screenshot shows the Postman interface with an unsuccessful API call. The URL in the request field is <https://designer.ibm.com:9449/employees/details/121212>. The response status is circled in red as 404 Not Found. The response body is a JSON object with a message:

```

1  {
2      "message": "Record for employee number 121212 was not found"
3  }
  
```

IBM z/OS Connect (OpenAPI 3.0)

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

The screenshot shows the Postman interface with a successful API call. The URL is `http://wg31.washington.ibm.com:2446/services/zCEEService/displayEmployee`. The response status is 200 OK, and the JSON body is:

```
1 [ {  
2   "ResultSet Output": [],  
3   "StatusCode": 200,  
4   "StatusDescription": "Execution Successful"  
5 } ]
```

This demonstrates one of the advantages of using z/OS Connect APIs as a gateway to Db2 REST services.

3. Next use *Postman* to invoke a *POST* method with URL

<https://designer.washington.ibm.com:9449/employees> and the JSON below for the request message.

```
{  
  "employeeNumber": "948499",  
  "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  
}
```

Pressing the **Send** button invokes the POST method of URI path */employees* which in turns invokes the Db2 REST service *zCEEService.insertEmployee*.

The screenshot shows the Postman application interface. A POST request is being made to <https://designer.ibm.com:9449/employees>. The request body is a JSON object containing employee details:

```

1 "employeeNumber": "948499",
2 "firstName": "Matt",
3 "middleInitial": "T",
4 "lastName": "Johnson",
5 "departmentCode": "C00",
6 "phoneNumber": "0065",
7 "dateOfHire": "10/15/1980",
8
  
```

The response status is 200 OK, and the message is: "Record for employee number 948499 was added successfully".

4. Invoke the GET method with URL

<https://designer.washington.ibm.com:9449/employees/details/948499> to display the record just inserted.

IBM z/OS Connect (OpenAPI 3.0)

The screenshot shows the Postman interface with a successful GET request to `https://designer.ibm.com:9449/employees/948499`. The response status is 200 OK, and the response body is a JSON object:

```
1 "retrievedResults Output": [
2   {
3     "employeeID": "948499",
4     "name": "Matt T Johnson",
5     "departmentCode": "C00",
6     "phoneNumber": "00651",
7     "dateOfHire": "1980-10-15",
8     "job": "Staff",
9     "educationLevel": 21,
10    "sex": "M",
11    "dateOfBirth": "1960-06-18",
12    "annualSalary": 3999.99,
13    "lastBonus": 399.99,
14    "lastCommission": 119.99
15  }
16 ]
17 ]
```

5. Invoke a *PUT* method with URL **`https://designer.washington.ibm.com:9449/employees/948499`** and the JSON below for the request message to update the *salary*, *bonus*, and *commission* columns of this row.

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

6.

The screenshot shows the Postman interface with a successful PUT request to `https://designer.ibm.com:9449/employees/948499`. The response status is 200 OK, and the response body is a JSON object:

```
1 {
2   "message": "Record for employee 948499 successfully updated"
3 }
```

7. Display the updated record by using a *GET* method with URL

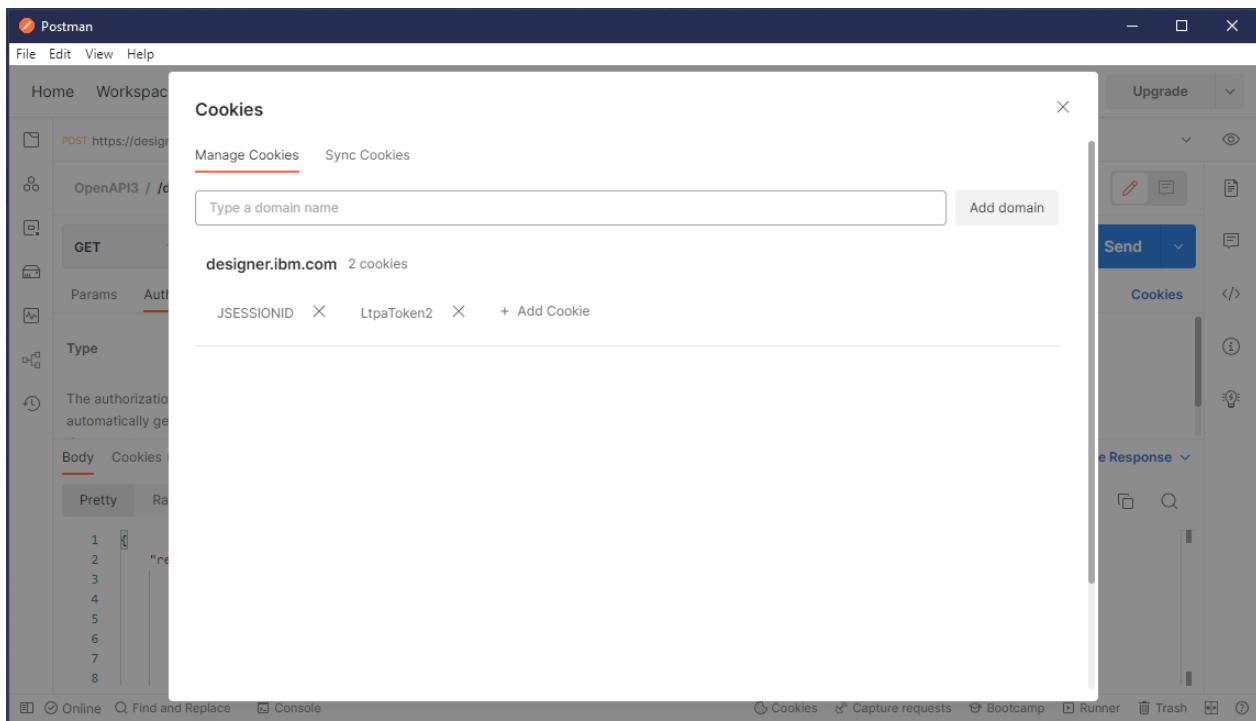
<https://designer.washington.ibm.com:9449/employees/details/948499>

```

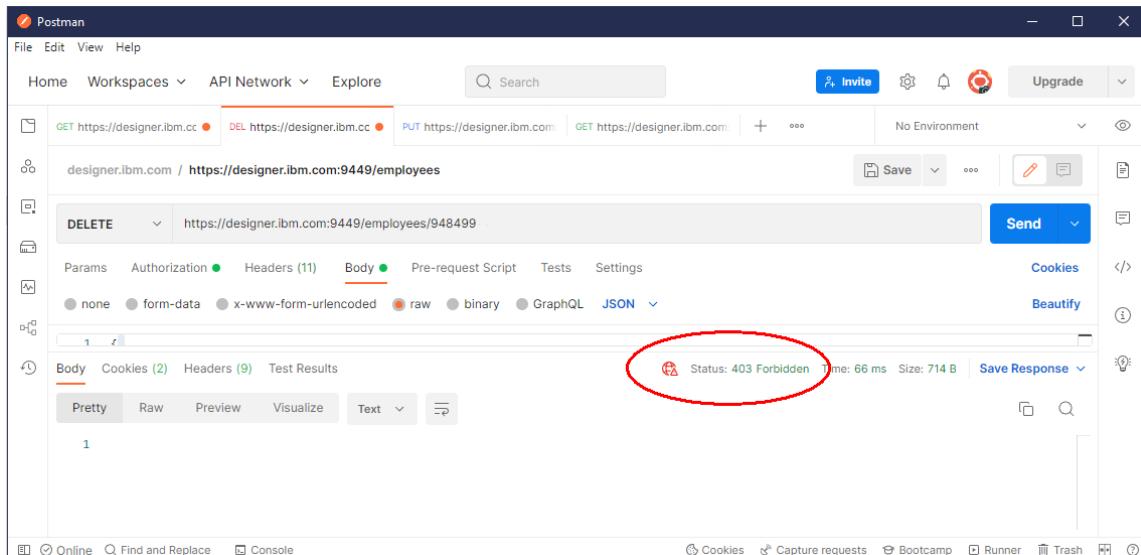
1  {
2      "retrievedResults Output": [
3          {
4              "employeeID": "948499",
5              "name": "Matt T Johnson",
6              "departmentCode": "C00",
7              "phoneNumber": "0065",
8              "dateOfHire": "1980-10-15",
9              "job": "Staff",
10             "educationLevel": 21,
11             "sex": "M",
12             "dateOfBirth": "1960-06-18",
13             "annualSalary": 5000.0,
14             "lastBonus": 500.0,
15             "lastCommission": 400.0
16         }
17     ]
18 }
```

8. 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.9. To clear the *Postman* cached security tokens, click on the *Cookies* section of the *Postman* window and

IBM z/OS Connect (OpenAPI 3.0)
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. For example, if you try to delete a record as *user1*, you should see an HTTP Status code of 403 (Forbidden).



While an delete request by Fred is successful.

The screenshot shows the Postman application interface. A DELETE request is being made to `https://designer.ibm.com:9449/employees/948499`. The response status is 200 OK, and the message body contains the string "record deleted".

```
1
2   "message": "record deleted"
3 }
```

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\openapi3`.
2. Enter the *curl* command below and observe the response.

```
curl -X GET -w " - HTTP CODE ${http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure
https://designer.washington.ibm.com:9449/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/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://designer.washington.ibm.com:9449/employees/details/000010
```

```
curl -X GET -w " - HTTP CODE ${http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure
https://localhost:9449/employees/details/000010
```

A review of the `trace.log` file will show the HTTP 403 (Forbidden) occurred because the identity *user2* is not a member of the *Staff* group.

```
curl -X GET -w " - HTTP CODE ${http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure
https://localhost:9449/employees/details/000010
```

The screenshot shows a browser window with the title "z/OS Connect Designer" and the URL "localhost:9449/logs/trace.log". The page content is a log file with several entries. Each entry shows a sequence of authorization checks for a user with ID 0000034e and password f33f56e8. The logs indicate that the user is not a member of the 'Staff' group, leading to a 403 Forbidden response. The log entries are as follows:

```
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService > useRolesAsGroupNameForAccessDecision Entry
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService > useRolesAsGroupNameForAccessDecision Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService > getGroups Entry
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService > getGroups Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService > getRealName Entry
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService > getRealName Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService < getRealName Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService < useRolesAsGroupNameForAccessDecision Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService < isSubjectAuthorized Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService 1 Subject is NOT authorized to access resource api as the Subject does not possess one of the required roles: [Staff]
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService < isAuthorized Exit
[6/21/22 19:36:43:372 UTC] 0000034e id=f33f56e8 y.authorization.builtin.internal.BuiltinAuthorizationService < isAuthorized Exit
```

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:fredpwd --data @insertEmployee.json
https://designer.washington.ibm.com:9449/employees/
```

```
c:\z\openapi3>curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @insertEmployee.json
https://designer.washington.ibm.com:9449/employees/
{"message":"Record for employee number 948489 was added successfully"} - HTTP CODE 200
```

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

```
{
  "employeeNumber": "948489",
  "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. 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:fredpwd
https://designer.washington.ibm.com:9449/roles/PRES?dept=A00
```

```
curl -X GET -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd
https://designer.washington.ibm.com:9449/roles/PRES?dept=A00
{"results Output": [{"employeeNumber": "[\"000010\", \"000011\"]", "name": "[\"CHRISTINE\", \"CHRISTINE\"]", "job": "[\"PRES\", \"PRES\"]"}, {"employeeNumber": "[\"000010\", \"000011\"]", "name": "[\"CHRISTINE\", \"CHRISTINE\"]", "job": "[\"PRES\", \"PRES\"]"}]} - HTTP CODE 200
```

Now try using user2's credentials.

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:fredpwd https://designer.washington.ibm.com:9449/employees/948489
```

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9449/employees/948489
{"message":"record deleted"} - HTTP CODE 200
```

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

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user1:user1 https://designer.washington.ibm.com:9449/employees/948489
```

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user user1:user1 https://designer.washington.ibm.com:9449/employees/948489
- HTTP CODE 403
```

8. Enter the *curl* command below to invoke the *PUT* with URI path */employees/{employee}*.

```
curl -X PUT -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @updateEmployee.json https://designer.washington.ibm.com:9449/employees/948489
```

```
curl -X PUT -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd --data @updateEmployee.json https://designer.washington.ibm.com:9449/employees/948489
- HTTP CODE 200
```

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

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

Invoke other curl commands varying the credentials , methods, etc. until you feel comfortable the API is working as intended.

Using the API Explorer

The API Explorer was used in the Designer to test the APIs as they were being developed. The API Explorer 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.

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

IBM z/OS Connect (OpenAPI 3.0)

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

The screenshot shows the z/OS Connect Designer OpenAPI UI. The browser title bar says "z/OS Connect Designer" and "OpenAPI UI". The address bar shows "https://localhost:9449/api/explorer/". The main content area displays the "employee roster" section. Under the "POST /employees" endpoint, there is a "Request body" section with a "Try it out" button. Below the "Request body" section, there is a "request body" example value and a schema definition:

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "string",
  "phoneNumber": "string",
  "dateOfBirth": "string",
  "job": "string",
  "educationLevel": 3267,
  "sex": "s",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

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

The screenshot shows a modal dialog titled "Parameters" with a "Cancel" button in the top right corner. It contains the same "Authorization" parameter and "Request body" fields as the previous screenshot. The "Request body" field is expanded to show the JSON schema again:

```
{
  "employeeNumber": "string",
  "firstName": "string",
  "middleInitial": "s",
  "lastName": "string",
  "departmentCode": "string",
  "phoneNumber": "string",
  "dateOfBirth": "string",
  "job": "string",
  "educationLevel": 3267,
  "sex": "s",
  "dateOfBirth": "string",
  "salary": 999999.99,
  "lastBonus": 999999.99,
  "lastCommission": 999999.99
}
```

Below the "Parameters" section is a "Servers" section with a note: "These path-level options override the global server options." A dropdown menu is shown with a slash character. At the bottom is a large blue "Execute" button.

3. 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
}
```

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.

The screenshot shows the IBM z/OS Connect API testing interface. At the top, there is a message: "These path-level options override the global server options." Below this is a search bar and a toolbar with "Execute" and "Clear" buttons. The main area is titled "Responses". Under "Responses", there is a "curl" section containing the cURL command used to make the request:

```
curl -X 'POST' \
  'https://localhost:9449/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
  }
```

Below the curl command is the "Request URL" field, which contains "https://localhost:9449/employees". Under "Server response", the status code is 200, and the "Response body" is shown as:

```
{
  "message": "Record for employee number 948478 was added successfully"
}
```

There are "Download" and "Copy" buttons next to the response body. The "Response headers" section shows the following:

```
content-language: en-US
content-length: 103
content-type: application/json
date: Fri, 17 Jun 2022 17:15:13 GMT
x-firefox-spdy: h2
x-powered-by: Servlet/4.0
```

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.

Curl

```
curl -X 'POST' \
  'https://localhost:9449/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://localhost:9449/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 EMPLOYEECA CONSTRAINS COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE VALUES IN THOSE COLUMNS.          RID OF EXISTING ROW IS X'0000000229'. Error Location:DSNLJXUS:6"
}
```

Response headers

```
content-language: en-US
content-length: 400
content-type: application/json
date: Fri,17 Jun 2022 17:18:19 GMT
x-firebase-spdy: h2
x-powered-by: Servlet/4.0
```

7. Scroll down and click on *GET /employees/{employee}* 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.

Execute **Clear**

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:9449/employees/948478' \
  -H 'accept: application/json'
```

Request URL

<https://localhost:9449/employees/948478>

Server response

Code	Details
200	Response body

Response body

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

Response headers

```
content-language: en-US
content-length: 133
content-type: application/json
date: Fri,17 Jun 2022 17:31:03 GMT
x-firebase-spdy: h2
x-powered-by: Servlet/4.0
```

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 **948478** 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 shows the 'Server response' interface. At the top, there are tabs for 'Code' (selected) and 'Details'. Below that, the status code '200' is shown, followed by 'Response body' which contains a JSON object with a single key 'message': "Record for employee 948478 successfully updated". To the right of the body are 'Copy' and 'Download' buttons. Below the body is the 'Response headers' section, which lists several HTTP headers: content-language: en-US, content-type: application/json, date: Mon, 18 Jul 2022 22:33:29 GMT, x-firebase-spdy: h2, and x-powered-by: Servlet/4.0.

12. Expand the **GET** method for URI path */employees/details/{employee}* and enter press the **Try it out** button.

13. Enter **948478** in the *employee* field and press the **Execute** button. Observe that the updates values have been applied.

The screenshot shows the 'Server response' interface. At the top, there are tabs for 'Code' (selected) and 'Details'. Below that, the status code '200' is shown, followed by 'Response body' which contains a JSON object with a key 'retrievedResults output': [{ ... }]. The detailed output includes fields like employeeID, name, departmentCode, phoneNumbers, dateOfBirth, job, educationLevel, sex, dateHire, annualSalary, lastBonus, and lastCommission. To the right of the body are 'Copy' and 'Download' buttons. Below the body is the 'Response headers' section.

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

The screenshot shows the 'Server response' interface. At the top, there are tabs for 'Code' (selected) and 'Details'. Below that, the status code '200' is shown, followed by 'Response body' which contains a JSON object with a single key 'message': "record deleted". To the right of the body are 'Copy' and 'Download' buttons.

15. 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.

EMPN	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

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>.

Name	Last Modified	Size	Description
api.war	Sun Jun 19 16:08:11 UTC 2022	35491	File

Double click the *api.war* file and 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 *ftp* command, e.g., **ftp wg31**
- 4. Logon as USER1 and then use the *cd* command to change to directory to data set */var/zcee/openApi3/apps*, e.g. **cd /var/zcee/openApi3/apps**
- 5. Toggle prompting off by entering command **prompt**
- 6. Enter binary mode transmission but entering command **bin**
- 7. Perform multiple put requests by using the multiple put command, **mput employees.war**

8. When the last transfer has completed enter the **quit** command.

```
c:\z\openApi3> cd wars
c:\z\openApi3\wars>ftp wg31.washington.ibm.com
Connected to wg31.washington.ibm.com.
220-FTP 16:26:23 on 2018-02-15.
220 Connection will close if idle for more than 200 minutes.
User (wg31.washington.ibm.com:(none)): user1
331 Send password please. user1
Password:
230 USER1 is logged on. Working directory is "USER1.".
ftp> cd /var/zcee/openApi3
250 HFS directory /var/zcee/openApi3/apps is the current working directory
ftp> prompt
Interactive mode Off .
ftp> bin
200 Representation type is Image
ftp> mput employees.war
200 Port request OK.
125 Storing data set /var/zcee/openApi3/apps/employees.war
250 Transfer completed successfully.
ftp: 35491 bytes sent in 0.39Seconds 90.77Kbytes/sec.
ftp> 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="/db2" 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

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. Set *Authorization* type to basic and use *fred* as the *Username* and *fred* as the *Password* as we did when directly testing the Db2 REST services and add a header property of **Content-Type** with a value of *application/json*. Then use the down arrow in the *Body* tab to select **GET** and enter <https://wg31.washington.ibm.com:9445/db2/employees/details/000010> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

The screenshot shows the Postman application window. In the top navigation bar, there are tabs for Home, Workspaces, API Network, and Explore. Below the navigation bar, there is a search bar and a toolbar with various icons. The main workspace shows a list of requests. One request is selected, which is a GET request to the URL <https://wg31.washington.ibm.com:9445/db2/employees/details/000010>. The request details panel shows the method (GET), URL, and various configuration options like Headers, Body, and Settings. The Body tab is selected, showing the JSON response. The response status is 200 OK, and the response body is a JSON object containing employee details. The JSON output is displayed in a collapsible tree view. At the bottom of the interface, there are various toolbars and status indicators.

```

1 {
2   "retrievedResults_Output": [
3     {
4       "employeeID": "000010",
5       "name": "CHRISTINE I HAAS",
6       "departmentCode": "A00",
7       "phoneNumber": "3978",
8       "dateOfHire": "1965-01-01",
9       "job": "PRES",
10      "educationLevel": 18,
11      "sex": "F",
12      "dateOfBirth": "1933-08-14",
13      "annualSalary": 52750.0,
14      "lastBonus": 1000.0,
15      "lastCommision": 4220.0
16    }
17  ]

```

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. The other major differences was of the context root of */db2* in the URI path. This was the value of the *contextRoot* attribute in the *webApplication* configuration element that defined this application in the z/OS Connect server.

2. Next enter an invalid employee number such as 121212,

<https://wg31.washington.ibm.com:9445/db2/employees/details/121212> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.

The screenshot shows the Postman application interface. A GET request is made to <https://wg31.washington.ibm.com:9445/db2/employees/details/121212>. The response status is 404 Not Found, and the message is "Record for employee number 121212 was not found".

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

The screenshot shows the Postman application interface. A POST request is made to <http://wg31.washington.ibm.com:2446/services/zCEEService/displayEmployee> with the body: {"employeeNumber": "121212"}. The response status is 200 OK, and the message is "Execution Successful".

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).

IBM z/OS Connect (OpenAPI 3.0)

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

```
{  
    "employeeNumber": "948489",  
    "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  
}
```

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, Home, Workspaces, API Network, Explore, and a search bar. The main workspace shows a 'New Request' card with a POST method and URL. The 'Body' tab is selected, showing a JSON payload identical to the one above. Below the body, the response status is 200 OK with a message: "Record for employee number 948489 was added successfully".

Or if you want to do a *PUT* with URI path /db2/employees/{employee} use this JSON request message.

<https://wg31.washington.ibm.com:9445/db2/employees/948489>

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

The screenshot shows the Postman application interface. A PUT request is being made to the URL <https://wg31.washington.ibm.com:9445/db2/employees/948489>. The request body is a JSON object:

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

The response status is 200 OK, Time: 574 ms, Size: 221 B. The response body is:

```
1 "message": "Record for employee 948489 successfully updated"
```

-
3. 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

4. To clear the *Postman* cached tokens, click on the *Cookies* section of the *Postman* window and

The screenshot shows the Postman interface with a GET request to `https://designer.ibm.com:9449/employees/948478`. The 'Cookies' tab is circled in red. The response status is 200 OK.

```

1
2   "results": [
3     {
4       "employeeNumber": "948478",
5       "name": "Matt T Johnson",
6       "departmentCode": "C00",
7       "phoneNumber": "0065",
8       "job": "Staff"
  
```

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

The screenshot shows the 'Cookies' dialog box in Postman. It lists two cookies for the domain `designer.ibm.com`: `JSESSIONID` and `LtpaToken2`.

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\openapi3*.

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

```
curl -X GET -w "%{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/db2/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w "%{http_code}" --user Fred:fredpwd --header "Content-Type: application/json" --insecure https://localhost:9449/employees/details/000010
{"retrievedResults": [{"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}" --user user2:user2 --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/db2/employees/details/000010
```

```
c:\z\openapi3>curl -X GET -w "%{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://localhost:9449/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/db2/employees/
```

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

```
{
  "employeeNumber": "948489",
  "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/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/db2/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/db2/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/db2/employees/000012
```

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 as well as the contents of the original YAML file used to develop the Open API 3 API.

The initial contents of the Db2 table are shown below.

EMPNO	FIRSTNAME	MIDINIT	LASTNAME	WORKDEPT	PHONEENO	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	SVLAVTORE	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,
   FIRSTNAME  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),
     FIRSTNAME  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"/>
```

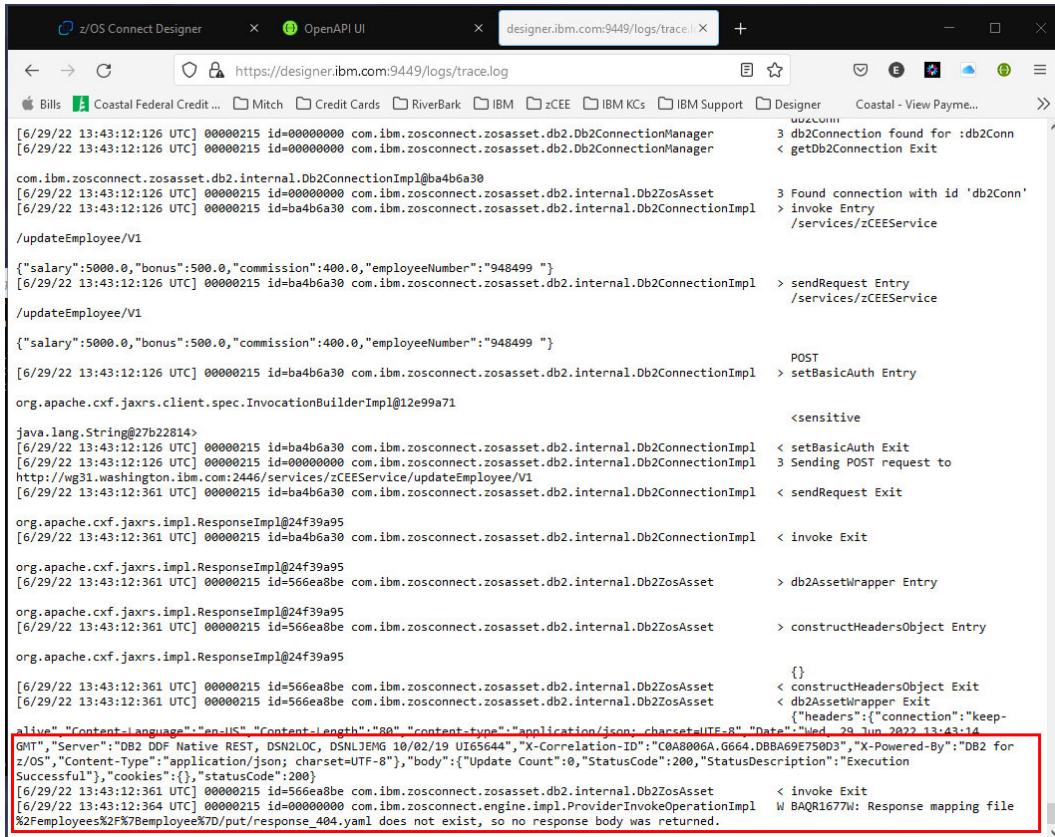
1. Invoking a method returned an HTTP 404 response and no other response. There should have been a message displayed to state that a record for this employee did not exist. A review of the trace showed the Db2 REST service did return an HTTP 404 status code and a status message. The status message indicated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

```

[6/29/22 12:21:08:558 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl > sendRequest
Entry
/zCEEService/displayEmployee/V1
{"employeeNumber": "948499"}
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl >
setShouldRebuild Entry
[6/29/22 12:21:08:573 UTC] 00000226 id=ba4b6a30 com.ibm.zosconnect.zosasset.db2.internal.Db2ConnectionImpl <
setShouldRebuild Exit
[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@5eeee633>
[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 3 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, DSNLJEMG 10/02/19 UI65644", "X-Correlation-ID": "C0A8006A.G64A.DBBA578FD7F8", "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.



The screenshot shows a browser window titled "z/OS Connect Designer" with the URL "designer.ibm.com:9449/logs/trace.log". The page displays a log of system events. A red box highlights a specific entry:

```
[6/29/22 13:43:12:361 UTC] 00000215 id=566eaab8e com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset < invoke Exit
[6/29/22 13:43:12:361 UTC] 00000215 id=566eaab8e com.ibm.zosconnect.zosasset.db2.internal.Db2ZosAsset W BAQR1677W: Response mapping file %2F%2Femployees%2F%2Femployee%70/put/response_404.yaml does not exist, so no response body was returned.
```

The contents of the employees.yaml file

```

openapi: 3.0.0
info:
  description: "Db2 Employees sample table"
  version: 1.0.0
  title: employees
servers:
- url: /
x-ibm-zcon-roles-allowed:
- Manager
security:
- BasicAuth: []
- BearerAuth: []
paths:
  /employees:
    post:
      tags:
        - db2employee
      operationId: postInsertEmployee
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/postInsertEmployee_request"
        description: request body
        required: true
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postInsertEmployee_response_200"
        "400":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postInsertEmployee_response_400"
        "500":
          description: OK
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/postInsertEmployee_response_500"
  /employees/department:
    get:
      tags:
        - db2employee
      operationId: getSelectByDepartments
      parameters:
        - name: Authorization
          in: header
          required: false
          schema:
            type: string
        - name: dept1
          in: query
          required: false
          schema:
            type: string
            maxLength: 3
        - name: dept2
          in: query
          required: false
          schema:
            type: string
            maxLength: 3

```

IBM z/OS Connect (OpenAPI 3.0)

```
responses:
  "200":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/getSelectByDepartments_response_200"
  "404":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/getSelectByDepartments_response_404"
  "500":
    description: OK
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/getSelectByDepartments_response_500"
"/employees/details/{employee}":
get:
  tags:
    - db2employee
  operationId: getDisplayEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/getDisplayEmployee_response_200"
    "404":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/getDisplayEmployee_response_404"
    "500":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/getDisplayEmployee_response_500"
"/employees/{employee}":
get:
  tags:
    - db2employee
  operationId: getSelectEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  responses:
    "200":
      description: OK
      content:
        application/json:
```

IBM z/OS Connect (OpenAPI 3.0)

```
schema:
  $ref: "#/components/schemas/getSelectEmployee_response_200"
"404":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/getSelectEmployee_response_404"
"500":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/getSelectEmployee_response_500"
put:
  tags:
    - db2employee
  operationId: putInsertEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  requestBody:
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/putInsertEmployee_request"
    description: request body
    required: true
  responses:
    "200":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/putInsertEmployee_response_200"
    "404":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/putInsertEmployee_response_404"
    "500":
      description: OK
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/putInsertEmployee_response_500"
delete:
  tags:
    - db2employee
  operationId: deleteDeleteEmployee
  parameters:
    - name: Authorization
      in: header
      required: false
      schema:
        type: string
    - name: employee
      in: path
      required: true
      schema:
        type: string
        maxLength: 6
  responses:
    "200":
      description: OK
      content:
        application/json:
```

IBM z/OS Connect (OpenAPI 3.0)

```

schema:
  $ref: "#/components/schemas/deleteDeleteEmployee_response_200"
"404":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/deleteDeleteEmployee_response_404"
"500":
  description: Not Found
  content:
    application/json:
      schema:
        $ref: "#/components/schemas/deleteDeleteEmployee_response_500"
"/employees/roles/{job}":
  get:
    tags:
      - db2employee
    operationId: getSelectByRole
    parameters:
      - name: Authorization
        in: header
        required: false
        schema:
          type: string
      - name: job
        in: path
        required: true
        schema:
          type: string
          maxLength: 8
      - name: dept
        in: query
        required: true
        schema:
          type: string
          maxLength: 3
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/getSelectByRole_response_200"
      "404":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/getSelectByRole_response_404"
      "500":
        description: OK
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/getSelectByRole_response_500"
components:
  schemas:
    getSelectEmployee_response_200:
      type: object
      required:
        - ResultSet Output
        - StatusCode
        - StatusDescription
      properties:
        ResultSet Output:
          type: array
          items:
            type: object
            description: ResultSet Row
            properties:
              employeeNumber:
                type: string
                description: CHAR(6)
                maxLength: 6
              firstName:
                type: string

```

IBM z/OS Connect (OpenAPI 3.0)

```

        description: VARCHAR(12)
        maxLength: 12
    middleInitial:
        type: string
        description: CHAR(1)
        maxLength: 1
    lastName:
        type: string
        description: VARCHAR(15)
        maxLength: 15
    department:
        type: string
        description: Nullable CHAR(3)
        maxLength: 3
    phoneNumber:
        type: string
        description: Nullable CHAR(4)
        maxLength: 4
    job:
        type: string
        description: Nullable CHAR(8)
        maxLength: 8
    required:
        - department
        - employeeNumber
        - firstName
        - job
        - lastName
        - middleInitial
        - phoneNumber
    StatusDescription:
        type: string
        description: Service invocation status description
    StatusCode:
        type: integer
        description: Service invocation HTTP status code
        minimum: 100
        maximum: 600
        description: Service selectEmployee invocation HTTP response body
getSelectEmployee_response_404:
    type: object
    properties:
        message:
            type: string
    example:
        message: Record could not be found
getSelectEmployee_response_500:
    type: object
    properties:
        message:
            type: string
    example:
        message: A severe error has occurred
deleteDeleteEmployee_response_200:
    type: object
    properties:
        message:
            type: string
    example:
        message: Record deleted
putInsertEmployee_request:
    type: object
    required:
        - bonus
        - commission
        - salary
    properties:
        salary:
            type: number
            description: Nullable DECIMAL(9, 2)
            minimum: -99999999.99
            maximum: 99999999.99
        bonus:
            type: number
            description: Nullable DECIMAL(9, 2)
            minimum: -99999999.99
            maximum: 99999999.99

```

IBM z/OS Connect (OpenAPI 3.0)

```
commission:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
description: Service updateEmployee invocation HTTP request body
deleteDeleteEmployee_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be found to be deleted
deleteDeleteEmployee_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
putInsertEmployee_response_200:
  type: object
  required:
    - StatusCode
    - StatusDescription
    - Update Count
  properties:
    Update Count:
      type: integer
      description: Update Count
      minimum: 0
      maximum: 32767
    StatusDescription:
      type: string
      description: Service invocation status description
    StatusCode:
      type: integer
      description: Service invocation HTTP status code
      minimum: 100
      maximum: 600
description: Service updateEmployee invocation HTTP response body
putInsertEmployee_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
putInsertEmployee_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
getDisplayEmployee_response_200:
  type: object
  required:
    - ResultSet Output
    - StatusCode
    - StatusDescription
  properties:
    ResultSet Output:
      type: array
      items:
        type: object
        description: ResultSet Row
        properties:
          EMPNO:
            type: string
            description: CHAR(6)
            maxLength: 6
          FIRSTNAME:
            type: string
            description: VARCHAR(12)
            maxLength: 12
          MIDINIT:
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

IBM z/OS Connect (OpenAPI 3.0)

```
type: string
description: CHAR(1)
maxLength: 1
LASTNAME:
  type: string
  description: VARCHAR(15)
  maxLength: 15
WORKDEPT:
  type: string
  description: Nullable CHAR(3)
  maxLength: 3
PHONENO:
  type: string
  description: Nullable CHAR(4)
  maxLength: 4
HIREDATE:
  type: string
  description: Nullable DATE yyyy-[m]m-[d]d
  minLength: 8
  maxLength: 10
  pattern: ^(?![0]{4})([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|1[2][0-9]|3[0-1])$
```

JOB:

```
  type: string
  description: Nullable CHAR(8)
  maxLength: 8
```

EDLEVEL:

```
  type: integer
  description: Nullable SMALLINT
  minimum: -32768
  maximum: 32767
```

SEX:

```
  type: string
  description: Nullable CHAR(1)
  maxLength: 1
```

BIRTHDATE:

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

SALARY:

```
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
```

BONUS:

```
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
```

COMM:

```
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
```

required:

- BIRTHDATE
- BONUS
- COMM
- EDLEVEL
- EMPNO
- FIRSTNME
- HIREDATE
- JOB
- LASTNAME
- MIDINIT
- PHONENO
- SALARY
- SEX
- WORKDEPT

StatusDescription:

```
  type: string
  description: Service invocation status description
```

StatusCode:

```
  type: integer
  description: Service invocation HTTP status code
  minimum: 100
```

IBM z/OS Connect (OpenAPI 3.0)

```
maximum: 600
description: Service displayEmployee invocation HTTP response body
getDisplayEmployee_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
getDisplayEmployee_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
postInsertEmployee_request:
  type: object
  required:
    - birthDate
    - bonus
    - commission
    - department
    - educationLevel
    - employeeNumber
    - firstName
    - hireDate
    - job
    - lastName
    - middleInitial
    - phoneNumber
    - salary
    - sex
  properties:
    employeeNumber:
      type: string
      description: Nullable CHAR(6)
      maxLength: 6
    firstName:
      type: string
      description: Nullable VARCHAR(12)
      maxLength: 12
    middleInitial:
      type: string
      description: Nullable CHAR(1)
      maxLength: 1
    lastName:
      type: string
      description: Nullable VARCHAR(15)
      maxLength: 15
    department:
      type: string
      description: Nullable CHAR(3)
      maxLength: 3
    phoneNumber:
      type: string
      description: Nullable CHAR(4)
      maxLength: 4
    hireDate:
      type: string
      description: Nullable DATE yyyy-[m]m-[d]d
      minLength: 8
      maxLength: 10
      pattern: ^(?![0]{4})([0-9]{4})-(0?[1-9]|1[0-2])-(0?[1-9]|1[0-2][0-9]|3[0-1])$ 
    job:
      type: string
      description: Nullable CHAR(8)
      maxLength: 8
    educationLevel:
      type: integer
      description: Nullable SMALLINT
      minimum: -32768
      maximum: 32767
    sex:
      type: string
      description: Nullable CHAR(1)
      maxLength: 1
```

Additional information and samples

© Copyright IBM Corporation 2022 All rights reserved
Mitch Johnson (mitchj@us.ibm.com)

100 of 103

IBM z/OS Connect (OpenAPI 3.0)

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

```
salary:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
```

```
bonus:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
```

```
commission:
  type: number
  description: Nullable DECIMAL(9,2)
  minimum: -9999999.99
  maximum: 9999999.99
```

```
description: Service insertEmployee invocation HTTP request body
```

```
postInsertEmployee_response_200:
  type: object
  required:
    - StatusCode
    - StatusDescription
    - Update Count
  properties:
    Update Count:
      type: integer
      description: Update Count
      minimum: 0
      maximum: 32767
    StatusDescription:
      type: string
      description: Service invocation status description
    StatusCode:
      type: integer
      description: Service invocation HTTP status code
      minimum: 100
      maximum: 600
```

```
description: Service insertEmployee invocation HTTP response body
```

```
postInsertEmployee_response_400:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
```

```
postInsertEmployee_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
```

```
getSelectByRole_response_200:
  type: object
  required:
    - ResultSet Output
    - StatusCode
    - StatusDescription
  properties:
    ResultSet Output:
      type: array
      items:
        type: object
        description: ResultSet Row
      properties:
        employeeNumber:
          type: string
          description: CHAR(6)
          maxLength: 6
        firstName:
          type: string
          description: VARCHAR(12)
```

IBM z/OS Connect (OpenAPI 3.0)

```
maxLength: 12
middleInitial:
  type: string
  description: CHAR(1)
  maxLength: 1
lastName:
  type: string
  description: VARCHAR(15)
  maxLength: 15
department:
  type: string
  description: Nullable CHAR(3)
  maxLength: 3
phoneNumber:
  type: string
  description: Nullable CHAR(4)
  maxLength: 4
job:
  type: string
  description: Nullable CHAR(8)
  maxLength: 8
required:
- department
- employeeNumber
- firstName
- job
- lastName
- middleInitial
- phoneNumber
StatusDescription:
  type: string
  description: Service invocation status description
StatusCode:
  type: integer
  description: Service invocation HTTP status code
  minimum: 100
  maximum: 600
  description: Service selectByRole invocation HTTP response body
getSelectByRole_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
getSelectByRole_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
getSelectByDepartments_response_200:
  type: object
  required:
- StatusCode
- StatusDescription
  properties:
    ResultSet 1 Output:
      type: array
      description: Stored Procedure ResultSet 1 Data
      items:
        type: object
        description: ResultSet Row
        properties: {}
  Anonymous ResultSets:
    type: integer
    description: Number of Anonymous ResultSets
    minimum: 0
    maximum: 1
  StatusDescription:
    type: string
    description: Service invocation status description
  StatusCode:
    type: integer
    description: Service invocation HTTP status code
    minimum: 100
```

IBM z/OS Connect (OpenAPI 3.0)

```
maximum: 600
description: Service selectByDepartments invocation HTTP response body
getSelectByDepartments_response_404:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record could not be inserted
getSelectByDepartments_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: A severe error has occurred
```