

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

# **Developing Native Server RESTful APIs for accessing a CICS Program**



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

*Date: April 27, 2023*

## Table of Contents

<b>Overview .....</b>	<b>3</b>
<b>The CSCVINC CICS application .....</b>	<b>4</b>
<b>The z/OS Connect Designer Container environment .....</b>	<b>4</b>
The container configuration file .....	5
Considerations when deploying multiple APIs in a native server .....	6
Connecting to a CICS region and the required server XML configuration .....	6
Basic security and the required server XML configuration .....	7
Accessing the z/OS Connect Designer log and trace files .....	9
<b>Developing a z/OS Connect APIs that accesses a CICS program .....</b>	<b>11</b>
Configure the POST method for URI path /employee .....	13
Configure the GET method for URI path /employee/{employee} .....	32
<b>Testing the API's POST and GET methods .....</b>	<b>38</b>
<b>Complete the configuration of the API (Optional) .....</b>	<b>43</b>
Configure the PUT method for URI path /employee/{employee} .....	43
Configure the DELETE method for URI path /employee/{employee} .....	48
<b>Testing APIs deployed in a z/OS Connect Designer container .....</b>	<b>53</b>
<b>Deploying and installing APIs in a z/OS Connect Native Server .....</b>	<b>59</b>
Moving the API Web Archive file from the container to a z/OS OMVS directory .....	59
Updating the server xml .....	61
Defining the required RACF EJBRole resources .....	62
<b>Testing APIs deployed in a native z/OS server .....</b>	<b>63</b>
Using Postman .....	63
Using cURL .....	69
Using the API Explorer .....	72
<b>Additional information and samples .....</b>	<b>82</b>
The table below list the contents of the VSAM data set. ....	82
Designer problem determination .....	83

**Important:** At URL <https://ibm.ent.box.com/v/WSC-OpenAPI3> there is a file named *OpenAPI3 development for CICS APIs CopyPaste.txt*. This file contains commands and other text used in this workshop. Locate that file and open it. Download this file and use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors. As a reminder text that appears in this file will be highlighted in yellow.

**Note:** Connectivity from the remote Desktop to this site is not always available. In this case, access this URL and download the file to your local Desktop and then copy the file from the local Desktop to the remote Desktop.

## 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 CICS to perform this exercise. Even if CICS 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 passwords for these users are *FRED*, *USER1* and *USER2* respectively and are case insensitive.
- ✓ Any time you have any questions about the use of screens, features or tools do not hesitate to reach out for assistance.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *OpenAPI 3 development APIs CopyPaste* file.
- ✓ Please note that there may be minor differences between the screen shots in this exercise versus what you see when performing this exercise. These differences should not impact the completion of this exercise. For example, the text might reference host name *designer.ibm.com* when a screen shot shows the host as *designer.ibm.com* or even *localhost*. All 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.

**Important:** On the desktop there is a file named *OpenAPI 3 Development for CICS 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.

**Note:** Connectivity from the remote Desktop to this site is not always available. In this case, access this URL from your local Desktop and then copy the file from the local Desktop to the remote Desktop.

## *The CSCVINC CICS application*

In this exercise we are developing an API to access a CICS application program named CSCVINC. This application is a COBOL application that uses a channel/container interface. This application, when invoked, accesses the channel provided by CICS and looks for the container in that channel. The name of the container is not hard coded in the application, so the application will accept a container with any name.

The container has a field named ACTION. The contents of this field determine what function the application will perform. Based on the value of ACTION, the application will access the CICS sample FILEA VSAM data set and either insert a new record (I), retrieve an existing record (G), update an existing record (U) or delete an existing record (D).

The application returns the results of the request including the values of the CICS EIBRESP and EIBRESP3 response code from the EXEC CICS WRITE, EXEC CICS READ, EXEC CICS REWRITE or EXEC CICS DELETE API performed by the program. Also returned is the identity under which the CICS transaction executed. The response container name will have the same name as the request container.

The COBOL copy book for the request container is shown here.

```

01 Request-Container.
   03 ACTION          PIC X(1) .
   03 USERID          PIC X(8) .
   03 FILEA-AREA.
       05 STAT        PIC X.
       05 NUMB        PIC X(6) .
       05 NAME        PIC X(20) .
       05 ADDR        PIC X(20) .
       05 PHONE       PIC X(8) .
       05 DATEX       PIC X(8) .
       05 AMOUNT      PIC X(8) .
       05 COMMENT     PIC X(9) .

```

The COBOL copy book for the response container is shown here.

```

01 Response-Container.
   03 ACTION          PIC X(1) .
   03 CEIBRESP        PIC S9(8) COMP.
   03 CEIBRESP2       PIC S9(8) COMP.
   03 USERID          PIC X(8) .
   03 FILEA-AREA.
       05 STAT        PIC X.
       05 NUMB        PIC X(6) .
       05 NAME        PIC X(20) .
       05 ADDR        PIC X(20) .
       05 PHONE       PIC X(8) .
       05 DATEX       PIC X(8) .
       05 AMOUNT      PIC X(8) .
       05 COMMENT     PIC X(9) .

```

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

```
version: "3.2"
services:
  zosConnect:
    image: icr.io/zosconnect/ibm-zcon-designer:3.0.57
    user: root
    environment:
      - BASE_PATH=cscvinc
      - 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:
      - "9447:9443"
      - "9084:9080"
    volumes:
      - ./project:/workspace/project
      - ./logs:/logs/
      - ./certs:/output/resources/security/
```

Connecting to a CICS region requires the addition of a `zosconnect_cicsIpicConnection` configuration element to the container's Liberty configuration. And since this API has role-based security elements configured, additional configuration elements for a basic registry and authorization roles are also required. These Liberty configuration elements are described next.

**Tech-Tip:** The contents of this `docker-compose.yaml` file were based on the example found in the z/OS Connect product documentation at URL <https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=desktop-create-cics-tutorial-workspace-using-docker>

## Considerations when deploying multiple APIs in a native server

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

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

```
servers:
  - url: /cscvinc
```

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

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

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

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

</server>
```

## Connecting to a CICS region and the required server XML configuration

The *zosconnect\_cicsIpicConnection* element used to connect to a CICS region in this exercise looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="IPIC connection to CICS">
  <featureManager>
    <feature>zosconnect:cics-1.0</feature>
  </featureManager>

  <zosconnect_cicsIpicConnection id="cicsConn"
    host="${CICS_HOST}"
    port="${CICS_PORT}"
    authDataRef="cicsCredentials" />

  <zosconnect_authData id="cicsCredentials"
    user="${CICS_USER}"
    password="${CICS_PASSWORD}" />

</server>
```

Notice the environment variables `${CICS_HOST}`, `${CICS_PORT}`, `${CICS_USER}` and `${CICS_PASSWORD}` are set to the values provided in the *docker-compose.yaml* file.

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

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

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

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

  <webAppSecurity allowFailOverToBasicAuth="true" />

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

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

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

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

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

```
openapi: 3.0.0
info:
  description: "CICS Employee Sample VSAM Application"
  version: 1.0.0
  title: Employee
x-ibm-zcon-roles-allowed:
- Manager
paths:
  /employee:
    post:
      -----
      "/employee/{employee}":
        get:
          tags:
            - Employee
          operationId: getEmployeeSelectService
          x-ibm-zcon-roles-allowed:
            - Staff
          -----
        put:
          tags:
            - Employee
          operationId: putEmployeeUpdateService
          x-ibm-zcon-roles-allowed:
            - Staff
          -----
        delete:
          tags:
            - Employee
          operationId: putEmployeeUpdateService
          x-ibm-zcon-roles-allowed:
            - Staff
          -----
```



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

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

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

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

Name	Last Modified	Size	Description
<a href="#">trace.log</a>	Fri Jul 15 18:11:11 UTC 2022	19700716	File
<a href="#">ffile</a>	Thu Jul 14 20:34:21 UTC 2022	-	Directory
<a href="#">messages_22.07.14_20.34.08.0.log</a>	Thu Jul 14 20:34:06 UTC 2022	40237	File
<a href="#">apiproject.log</a>	Fri Jul 15 13:13:06 UTC 2022	1396	File
<a href="#">messages.log</a>	Fri Jul 15 18:11:11 UTC 2022	235793	File
<a href="#">trace_22.07.15_14.10.16.0.log</a>	Fri Jul 15 14:10:16 UTC 2022	20971262	File

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

```

[7/15/22 18:22:43:224 UTC] 00000044 id=ad10b3fc com.ibm.zosconnect.cics.internal.conn.isc.Connection < sessionPoolIdle Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=a5b3060b com.ibm.zosconnect.cics.internal.conn.isc.SessionPool < deallocateSession Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.Session 3 Session state updated from FREEING to IDLE
[7/15/22 18:22:43:224 UTC] 00000044 id=2047cc94 com.ibm.zosconnect.cics.internal.conn.isc.Session < deallocateSession Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.Conversation 3 updateConvStatus, Old state:CICS_COMPLETE, New state:DEALLOCATED
[7/15/22 18:22:43:224 UTC] 00000044 id=1519b930 com.ibm.zosconnect.cics.internal.conn.isc.Conversation < updateConvStatus Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=e6867050 com.ibm.zosconnect.cics.internal.conn.isc.SessionManager < endConversation Exit
[7/15/22 18:22:43:224 UTC] 00000044 id=95c3f4d6 com.ibm.zosconnect.cics.ServerECIRequest < executeISC Exit
false
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.CicspicConnection 3 RequestFinished(com.ibm.zosconnect.cics.ServerECIRequest@95c3f4d6)
[7/15/22 18:22:43:224 UTC] 00000044 id=eab3c3c4 com.ibm.zosconnect.cics.CicspicConnection > workEnded Entry
[7/15/22 18:22:43:224 UTC] 00000044 id=00000000 com.ibm.zosconnect.cics.CicspicConnection 3 work ended, remaining work in progress is 0
[7/15/22 18:22:43:225 UTC] 00000044 id=eab3c3c4 com.ibm.zosconnect.cics.CicspicConnection < workEnded Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=eab3c3c4 com.ibm.zosconnect.cics.CicspicConnection < flow Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset < flowRequest Exit
[7/15/22 18:22:43:225 UTC] 00000044 id=2140441a com.ibm.zosconnect.cics.Channel > getContainer Entry
containerCscvnc
[7/15/22 18:22:43:225 UTC] 00000044 id=2140441a com.ibm.zosconnect.cics.Channel < getContainer Exit
com.ibm.zosconnect.cics.ChannelContainer@fec91d0
[7/15/22 18:22:43:225 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset > transformBytesToJson Entry
com.ibm.zosconnect.wv.metadata.transaction.Message@068c70cb
[B@4a269008,len=97
[0000] E2000000 00000000 50C3C9C3 E2E4E2C5 D940F1F2 F1F2F1F2 40404040 40404040
[0020] 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
[0040] 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040
[0060] 40
[7/15/22 18:22:43:242 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsZosAsset < transformBytesToJson Exit
{"Response-Container":{"ACTION":"S","CEIBRESP":13,"CEIBRESP2":00,"USERID":"CICSUSER","FILE-AREA":
[7/15/22 18:22:43:243 UTC] 00000044 id=4ad8146d com.ibm.zosconnect.zosasset.cics.internal.CicsChannelZosAsset < invoke Exit
  
```

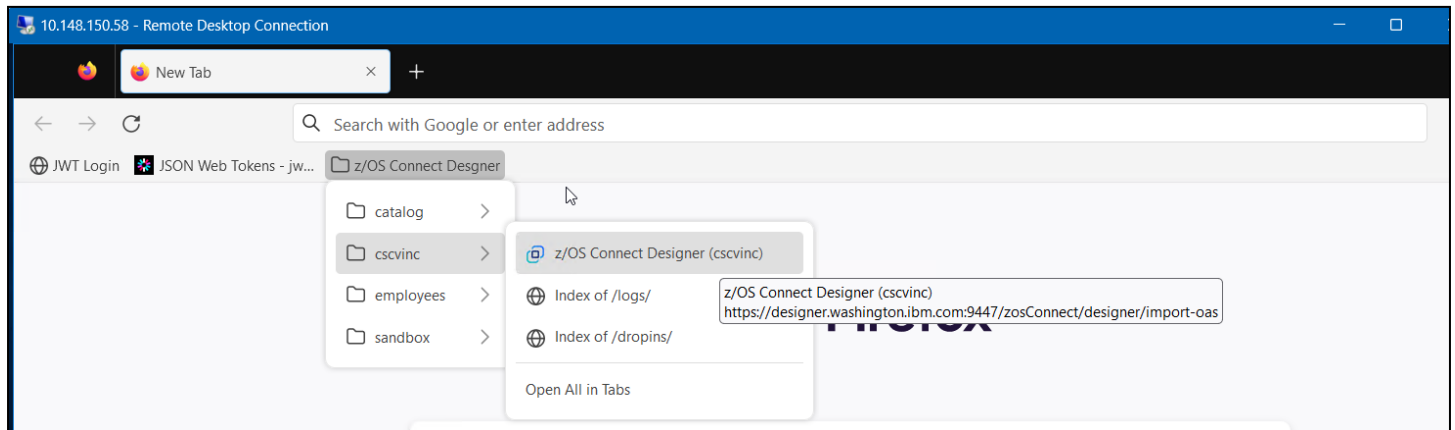
Search: cicsconn 2 of 90 matches

## Developing a z/OS Connect APIs that accesses a CICS program

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

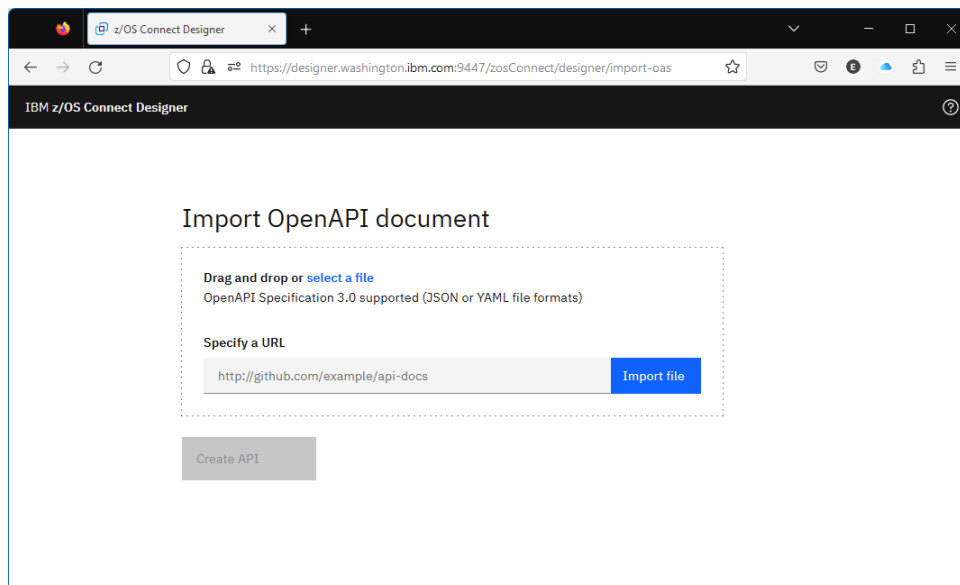
1. Open the Firefox browser and go to URL <https://designer.washington.ibm.com:9447/zosConnect/designer/>

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

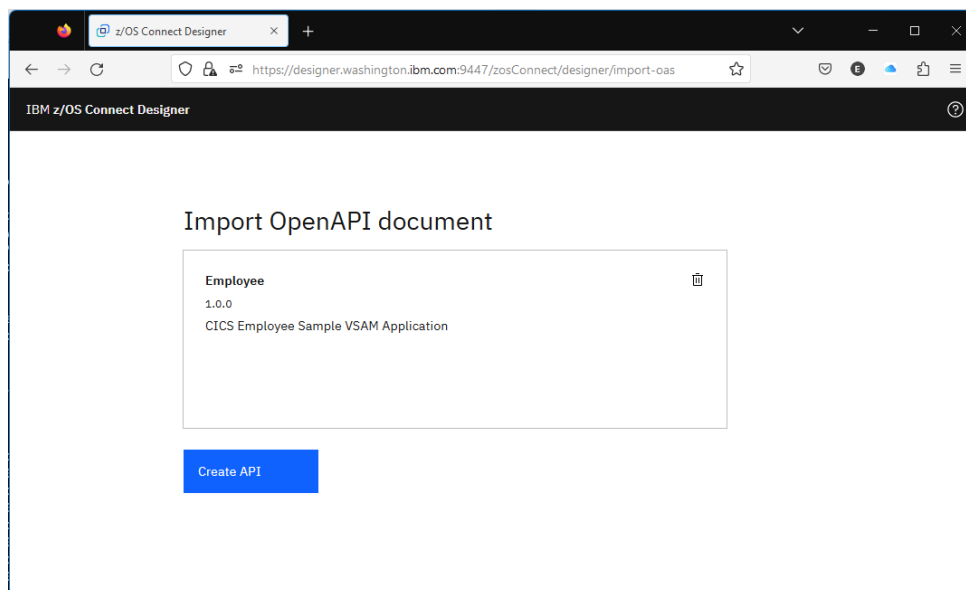


**Tech-Tip:** Be patient. It may take a while for the first page to fully load. Numerous background activities are being performed to fully initialize the application.

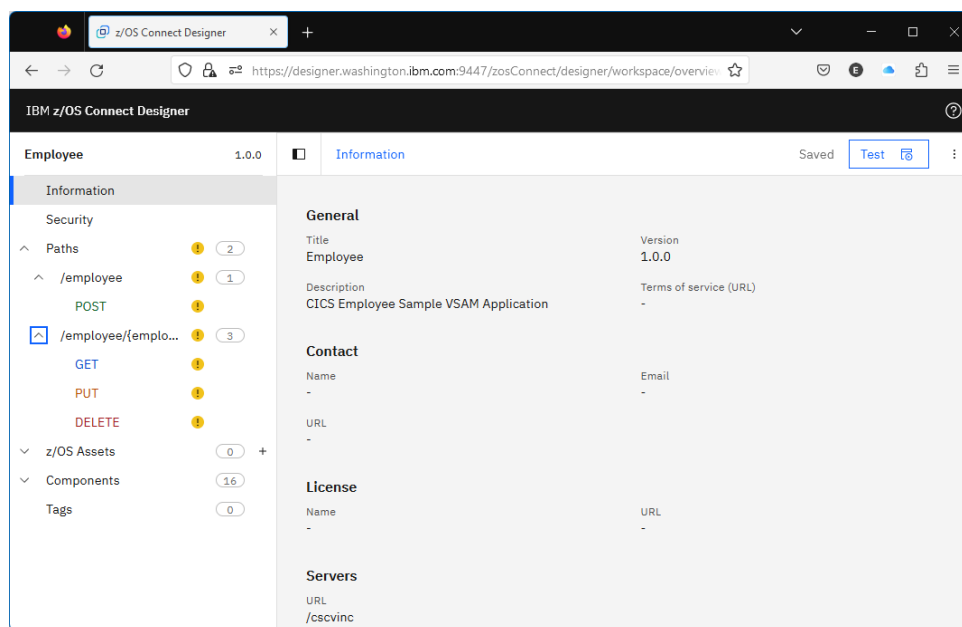
2. The first window you will see in a 'fresh' *Designer* environment gives you the opportunity to import an OpenAPI document. On the *Import OpenAPI document* window, click on [select a file](#) and traverse in the *File Upload* window to the directory where the specification document files are stored,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



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



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

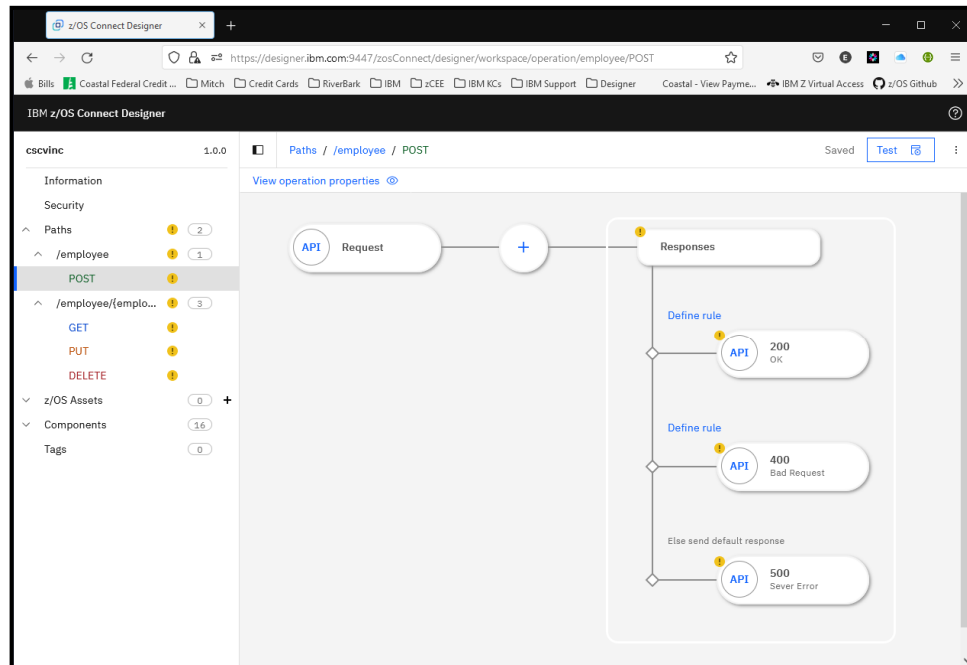


*Important:* When using this tool, monitor the upper right-hand corner of the page. You will see either status of either **Saved** or **Saving**. It is suggested that you wait until changes are saved before continuing using the *Designer*.

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

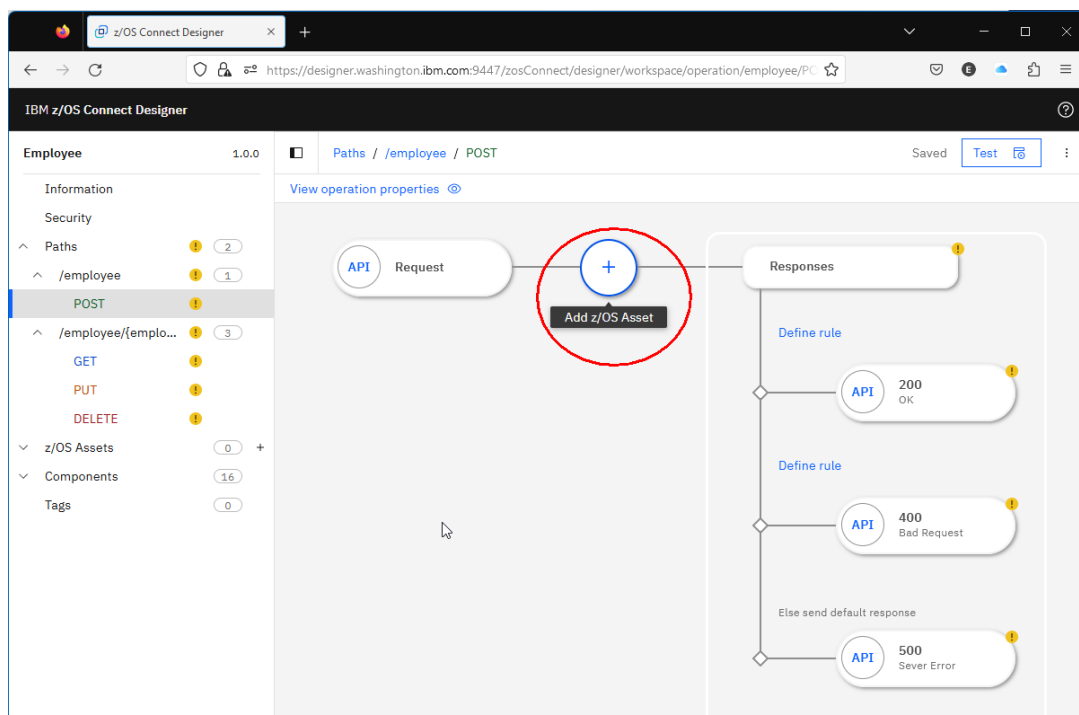
**Configure the POST method for URI path /employee**

1. Selecting a method will display the operation properties of the method. Start with the *POST* method under */employee* 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 insert of the 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.

Step 1 of 4  
Add z/OS Asset

Create new or select existing z/OS asset

☒ Add new z/OS Asset

☐ Use existing asset

Cancel Next

\_\_\_ 4. On the *Add z/OS Asset (Step 2 of 4)* page, use the pull-down arrow and select *CICS channel program*. This action will display additional items on the page.

IBM z/OS Connect Designer

Employee 1.0.0

Information  
Security  
Paths  
POST  
GET  
PUT  
DELETE  
z/OS Assets  
Components  
Tags

Step 2 of 5  
Add z/OS Asset

Select a z/OS Asset type  
CICS channel program

CICS program name  
Input program name

Program language  
Select a program language

CCSID  
Input CCSID

Select a CICS connection  
Select a connection

Optional configuration  
Transaction ID (optional)  
Input Transaction ID

Transaction ID usage (optional)  
Select usage

Previous Next

5. On the *Add z/OS Asset (Step 2 of 5)* page, enter a *CICS program name* of **CSCVINC**, use the pull-down arrow and select the **COBOL** as the *program language*. Enter **037** as the *CCSID* and use the pull-down arrow and select the **cicsConn** as the *CICS connection*. This enables the **Next** button. Press **Next** to continue.

Step 2 of 5

Add z/OS Asset

Select a z/OS Asset type

CICS channel program

CICS program name

CSCVINC

Program language

COBOL

CCSID

037

Select a CICS connection

cicsConn

**Optional configuration**

Transaction ID (optional)

Input Transaction ID

Transaction ID usage (optional)

Select usage

Previous Next

**Tech-Tip:** The name **cicsConn** is the name of the `zosconnect_cicsIpicConnection` configuration element described earlier in this exercise.

6. On the *Add z/OS Asset (Step 3 of 5)* page, details of the request container will be provided. Press the **Add Container** button and enter a name of **containerCscvnc** for the *CICS container name*.

7. Next press the **Import Data structure** to start the definition of the layout of this container.

Step 3 of 5

Add z/OS Asset

Define the request CICS channel program

**Request channel**

CICS container name	CICS container type
containerCscvnc	BIT

Import data structure

Add container

Previous Next

8. On the *Add z/OS Asset / Import data structure* page, press the *select a file* area to start the import of the COBOL structure that represent the request container.

Add z/OS Asset / Import data structure

Import a data structure into your request channel

Drag and drop or [select a file](#)  
Import data structure

Previous Add

9. Traverse to directory *c:/z/cicslab* and select file *CSCCREQ.cpy*. This file contains the COBOL source of the request container. Click the **Open** button to import the COBOL source into the *Designer*. Check the box beside *CSCCREQ.cpy* and click the **Add** button to continue.

Add z/OS Asset / Import data structure

Import a data structure into your request channel

Drag and drop or [select a file](#)  
Import data structure

<input checked="" type="checkbox"/>	Copybook name	Data structure	Status
<input checked="" type="checkbox"/>	CSCCREQ.cpy	Request-Container	Imported

Previous Add



10. On the *Add z/OS Asset (Step 3 of 6)* page, the COBOL structure of this container will be displayed (see below). Additional input or request containers be added by clicking the **Add container** button and repeating this process. Since this application expects only one input or request container, click the **Next** button to continue.

Step 3 of 5

Add z/OS Asset

Define the request CICS channel program

Request channel

CICS container name

containerCscvinc

CICS container type

BIT

↑ ↓

🗑

01 Request-Container.  
03 ACTION PIC X USAGE DISPLAY.  
03 USERID PIC X(8) USAGE DISPLAY.  
03 FILEA-AREA.  
05 STAT PIC X USAGE DISPLAY.  
05 NUMB PIC X(6) USAGE DISPLAY.  
05 NAME PIC X(20) USAGE DISPLAY.  
05 ADDR PIC X(20) USAGE DISPLAY.  
05 PHONE PIC X(8) USAGE DISPLAY.  
05 DATEX PIC X(8) USAGE DISPLAY.  
05 AMOUNT PIC X(8) USAGE DISPLAY.  
05 COMMENT PIC X(9) USAGE DISPLAY.

↑ ↓

🗑

Import data structure

Add container +

Previous

Next

11. The response container is configured on the *Add z/OS Asset (Step 4 of 5)* page. If the response container that exactly matches the request container, then clicking the *Replicate request structure* will create a response container with the same details as the request container. In this case, the response container does not match the request container, so click the **Add container** button to add the details of the response container.

12. This action will cause the **Import data structure** to appear. Enter *containerCscvinc* as the *CICS container name* (remember the application returns a response container using the same name as the request container). Click on the **Import data structure** button to import the layout of the response container.

13. On the *Add z/OS Asset/Import data structure* page. Click on the [select a file](#) area to continue.

14. Traverse to directory *c:/z/cicslab* and select file *CSCCRESP.cpy* which contains the COBOL source of the response container. Click the **Open** button to import the COBOL source into the *Designer*. Check the box beside *CSCCRESP.cpy* and click the **Add** button to continue.

<input checked="" type="checkbox"/>	Copybook name	Data structure	Status
<input checked="" type="checkbox"/>	CSCCRESP.cpy	Response-Container	Imported

15. On the *Add z/OS Asset (Step 4 of 5)* page, the COBOL structure of this container will be displayed (see below). Additional input or request containers be added by clicking the **Add container** button and repeating this process. This application only produces one response container, so click the **Next** button to continue.

Step 4 of 5

Add z/OS Asset

Define the response CICS channel program

Response channel

CICS container name: containerCscvinc

CICS container type: BIT

```

01 RESPONSE-CONTAINER.
03 ACTION PIC X USAGE DISPLAY.
03 CE1BRESP PIC S9(8) USAGE COMP.
03 CE2BRESP2 PIC S9(8) USAGE COMP.
03 USERID PIC X(8) USAGE DISPLAY.
03 FILEA-AREA.
05 STAT PIC X USAGE DISPLAY.
05 NUMB PIC X(6) USAGE DISPLAY.
05 NAME PIC X(20) USAGE DISPLAY.
05 ADDR PIC X(20) USAGE DISPLAY.
05 PHONE PIC X(8) USAGE DISPLAY.
05 DATEX PIC X(8) USAGE DISPLAY.
05 AMOUNT PIC X(8) USAGE DISPLAY.
05 COMMENT PIC X(9) USAGE DISPLAY.
    
```

Import data structure

Add container

Previous Next

16. On the *Add z/OS Asset (Step 5 of 5)* page, we are required to provide a name and/or optionally provide a description of the z/OS asset. In this exercise, enter a value of *programCscvinc* as the name of the z/OS asset (a CICS application) and click the **Add z/OS Asset** button to complete the definition of this program as an asset and to continue.

Step 5 of 5

### Add z/OS Asset

Provide a name and description so your z/OS Asset can be saved for reuse.

z/OS Asset name

programCscvinc

z/OS Asset description (optional)

Enter a description

**Summary**

z/OS Asset type  
CICS channel and containers program

CICS program name  
CSCVINC

Connection reference  
cicsConn

Previous Add z/OS Asset

17. 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 field names of the CICS request container. These fields to the request schema properties of the API as defined in the specification document that defined the API.

programCscvinc z/OS Asset options

Request Response z/OS Asset details

Edit mapping View structure

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

Channel

containerCscvinc

Request-Container	z/OS Asset field	Value	Help
ACTION		abc	?
USERID		abc	?
FILE-AREA			
STAT		abc	?
NUMB		abc	?
NAME		abc	?
ADDR		abc	?
PHONE		abc	?
DATEX		abc	?
AMOUNT		abc	?
COMMENT		abc	?

**Tech-Tip:** The container field names on the left-hand side are derived from the inbound container copy book.

```

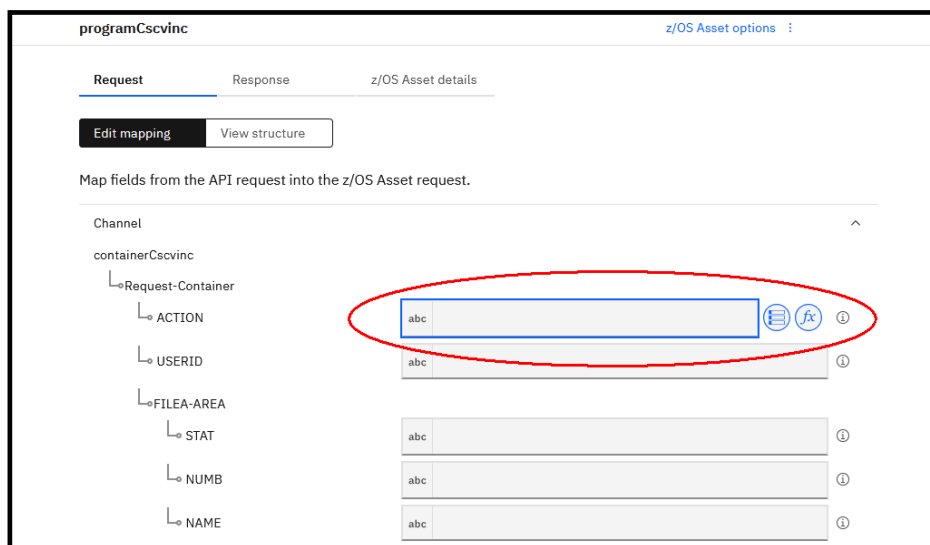
01 Request-Container.
   03 ACTION          PIC X(1).
   03 USERID          PIC X(8).
   03 FILEA-AREA.
      05 STAT         PIC X.
      05 NUMB         PIC X(6).
      05 NAME         PIC X(20).
      05 ADDR          PIC X(20).
      05 PHONE        PIC X(8).
      05 DATEX        PIC X(8).
      05 AMOUNT       PIC X(8).
      05 COMMENT      PIC X(9).
  
```

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

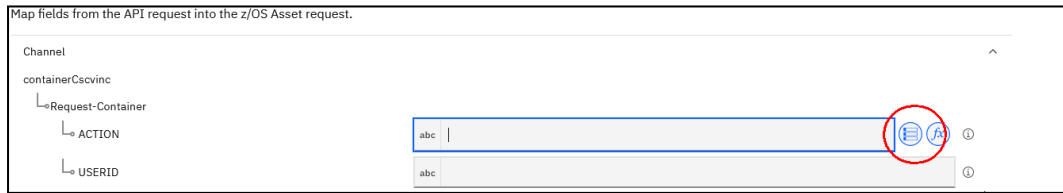
```

type: object
properties:
  EmployeeInsertServiceOperation:
    type: object
    properties:
      employeeData:
        type: object
        properties:
          request:
            type: object
            properties:
              employeeDetails:
                type: object
                properties:
                  employeeNumber:
                    type: string
                    maxLength: 6
                  name:
                    type: string
                    maxLength: 20
                  address:
                    type: string
                    maxLength: 20
                  phoneNumber:
                    type: string
                    maxLength: 8
                  date:
                    type: string
                    maxLength: 8
                  amount:
                    type: string
                    maxLength: 8
              required:
                - employeeData
  mncEmInInsertServiceResponse:
    type: object
  
```

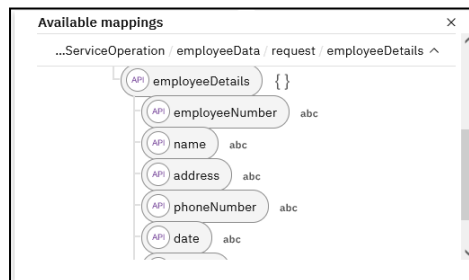
**Note:** It is very important that when working with mapping fields that the field has been properly selected. A properly selection field will be displayed in a **blue box** as shown below.



18. Now map the CICS container 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).

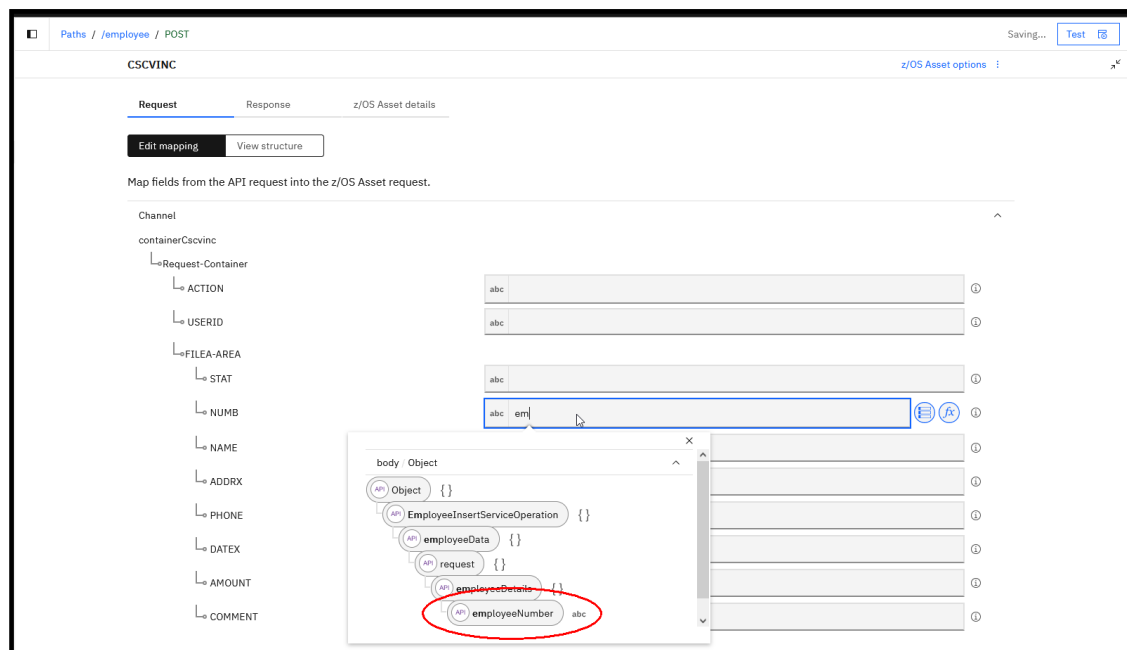


19. This will display the header and body mappings available for this method of the API. Become familiar with the mappings available in the body of the request message (you will have to use the scroll bar to see all the available mappings). Knowledge of the mappings in the body will help greatly in the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




*There are two ways to map the CICS request container fields to 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.*

20. Select an empty CICS request container field and start typing the corresponding API request message field name. For example, entering the string *em* in the area beside NUMB 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).



21. Select the field and this will cause the API's request message field name to populate the area beside field name in the container area and complete the mapping.

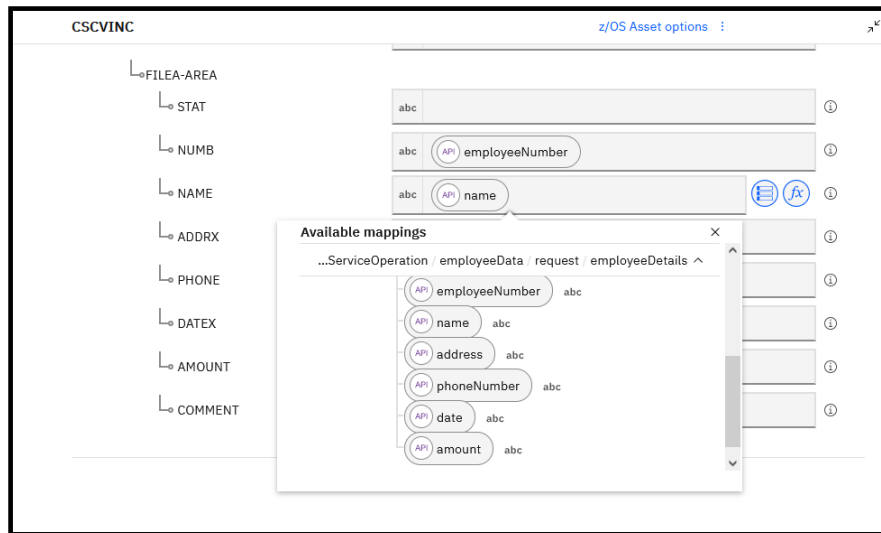
The screenshot shows the 'Request' tab of the 'CSCVINC' interface. It displays a tree structure on the left with fields like 'ACTION', 'USERID', 'STAT', 'NUMB', 'NAME', 'ADDRX', 'PHONE', 'DATEX', 'AMOUNT', and 'COMMENT'. On the right, each field has a corresponding input box. The 'NUMB' field's input box contains the text 'employeeNumber' with an 'API' icon to its left, indicating a successful mapping.

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

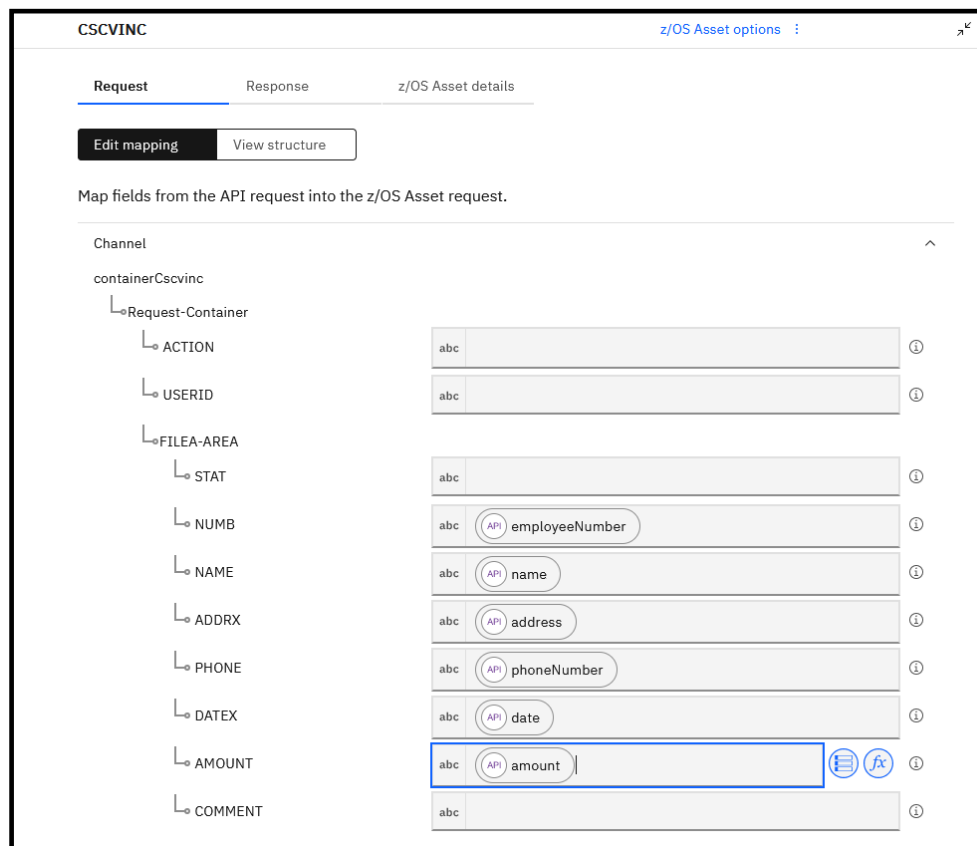
22. The alternate mapping method is to select the *Insert a mapping* tool (see below).

This screenshot shows the same mapping interface as before, but with the 'NAME' field's input box selected. A red circle highlights the 'Insert a mapping' button (a blue icon with a plus sign) located at the bottom right of the input box. A tooltip labeled 'insert a mapping' is visible over the button.

23. 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 request body *name* field.



24. Use either technique to complete the mappings of the container fields to the API request body fields. When completed, the results should look something like this the page below.





25. Note that not all of the container fields have been mapped from the API request message but by providing request message field names. For the *ACTION* field, the application requires that this field have a value of *I* in order for a record to be inserted in the VSAM data sets. The *USERID* is used by another client application and has no meaning for us, so leave this field blank. Set the value of the *STAT* field to an asterisk *\** and set the value of the *COMMENT* field to `{{ $substring($now(), 2,8) }}` (the current date).

CSCVINC
z/OS Asset options

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

Channel
^

containerCscvinc

Request-Container

ACTION
abc I

USERID
abc

FILEA-AREA

STAT
abc \*

NUMB
abc API employeeNumber

NAME
abc API name

ADDRX
abc API address

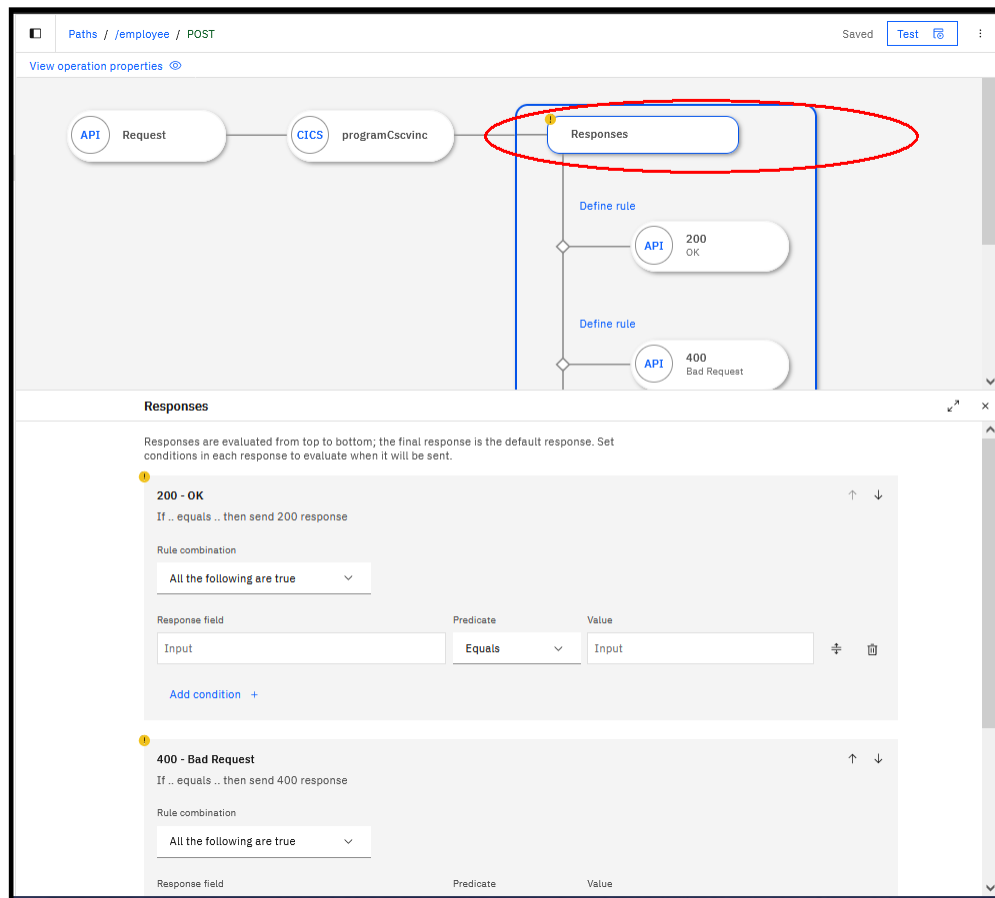
PHONE
abc API phoneNumber

DATEX
abc API date

AMOUNT
abc API amount

COMMENT
abc {{ \$substring(\$now(), 14,8) }}

26. The next step is to evaluate the responses that come back in the CICS program response container. Select the *Responses* box in the *View operation properties* page.



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

The contents of the CEIBRESP and CEIBRESP2 fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was inserted as intended. The application will set the CEIBRESP and CEIBRESP2 fields to zero if a record was successfully insert into the VSAM data set. Otherwise, the insert failed. We are going to check and provide a message if a duplicate record condition (CEIBRESP = 14 and CEIBRESP2=150) was raised. Otherwise, a message indicating a severe error has occurred along with the values of CEIBRESP and CEIBRESP2 fields.

**Responses**

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

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

Rule combination: All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**400 - Bad Request**  
If .. equals .. then send 400 response

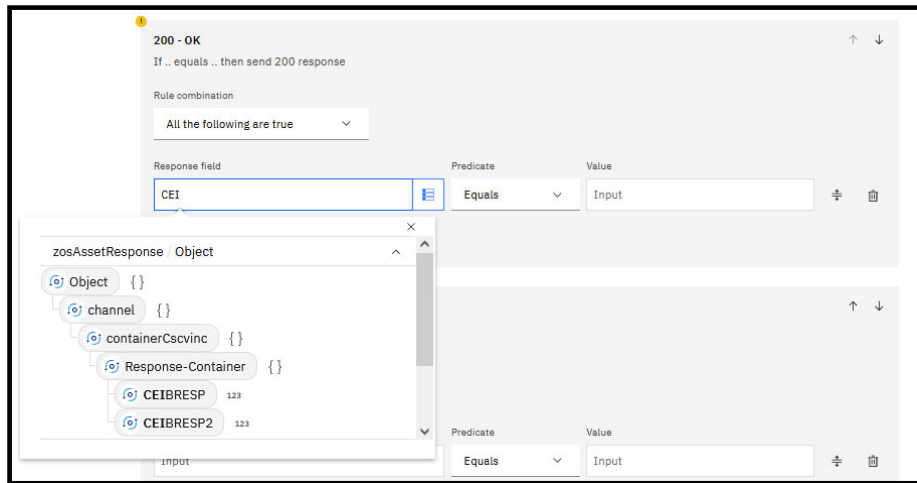
Rule combination: All the following are true

Response field	Predicate	Value
Input	Equals	Input

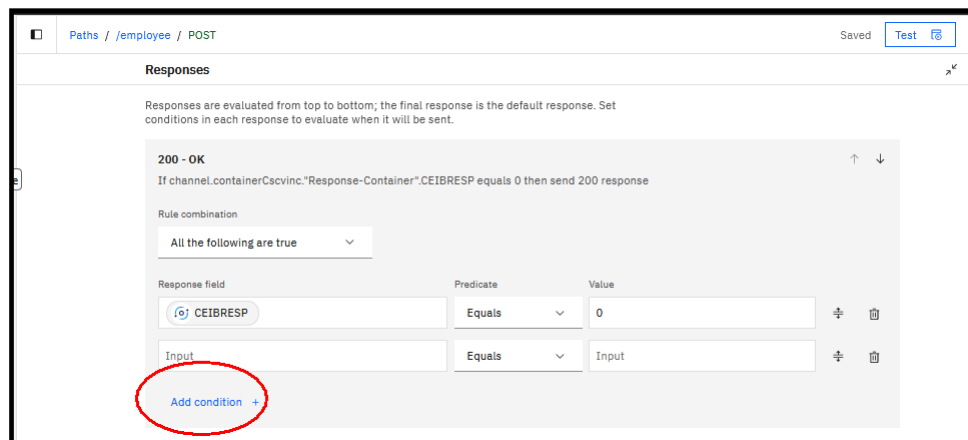
Add condition +

**500 - Internal Server Error**  
Else send default response

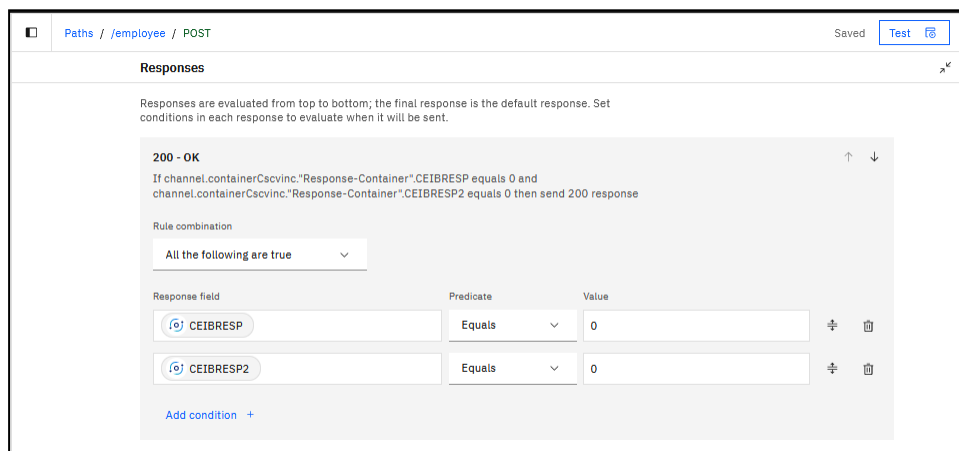
28. Under the **200 – OK** response, Enter the string **CEI** in the **Input** area under **Response field**. This will display all the fields in the CICS response container which match this string (position of the string in the field name does not matter). If the string matches any portion of the field name, that field will be displayed. In this case, map or select the **CEIBRESP** field in the CICS response container. Leave the **Predicate** as **Equals** and enter **0** in the **Input** field for **Value**.



29. Next add a condition check for the value of response container field **CEIBRESP2** by clicking on **Add condition** in the **200 – OK** evaluation.



30. Use the same technique described above to add a check for response container field **CEIBRESP2**. Leave the **Predicate** as **Equals** and set the value to **0** as shown below:



31. For the *400 – Bad Request* check, add a check for container field **CEIBRESP** equaling **14** and response container **CEIBRESP2** equaling **150**.

**400 - Bad Request**

If channel.containerCscvnc.'Response-Container'.CEIBRESP equals 14 and channel.containerCscvnc.'Response-Container'.CEIBRESP2 equals 150 then send 400 response

Rule combination

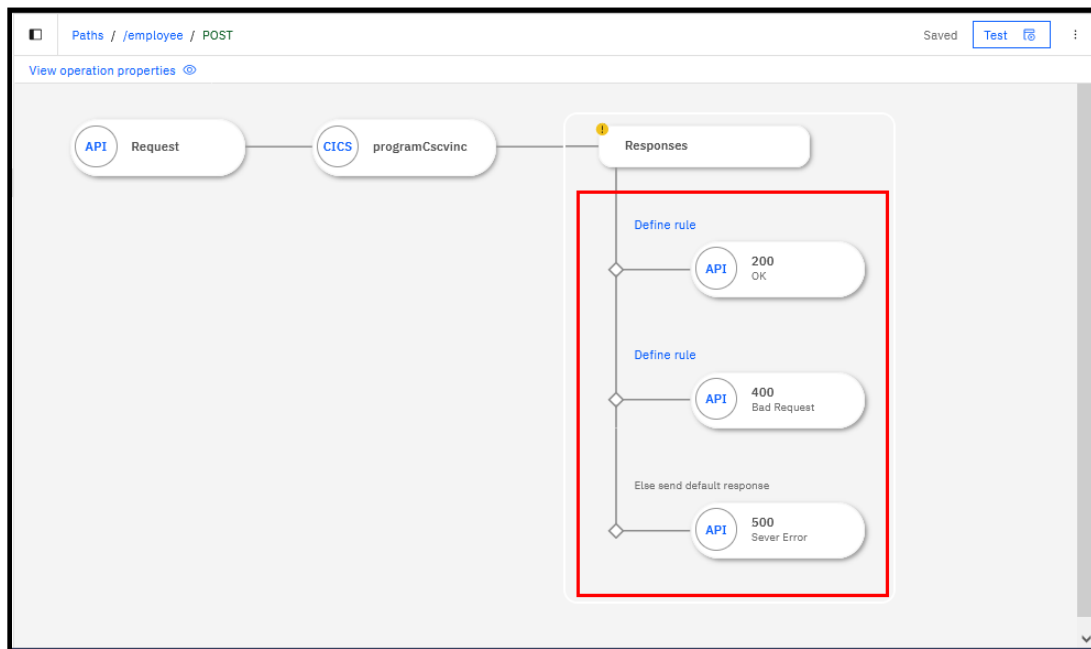
All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	14
CEIBRESP2	Equals	150

Add condition +

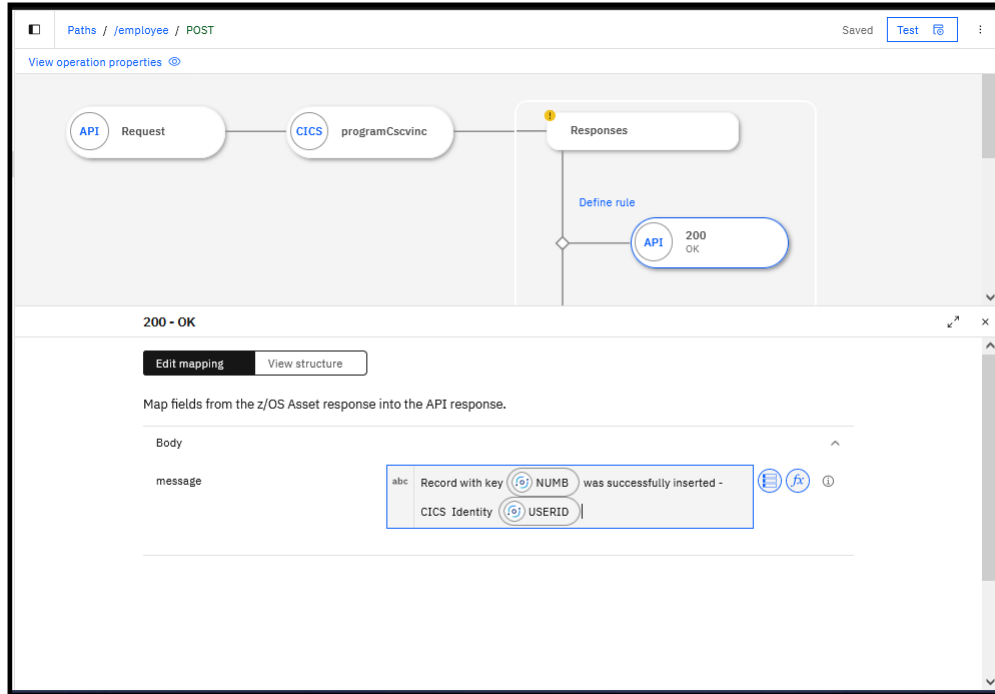
32. If neither of these connections are met, simply return with a HTTP 500 status code.

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



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

**Record with key {{\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEAREA".NUMB}} was successfully inserted - CICS Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**



The same techniques used to map API response with the CICS request container can be used to insert CICS response container fields 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 CICS response container.

**Tech-Tip:** The response messages for contingencies were defined in the YAML document as a single message field.

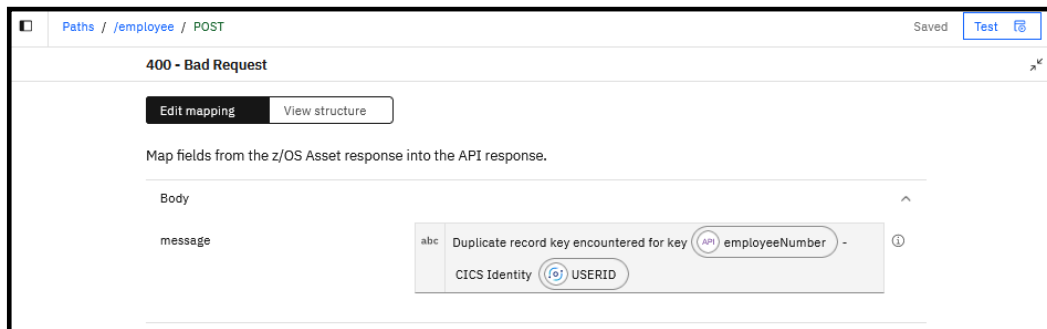
```

cscvinc.yaml - Notepad
File Edit View
- employeeData
- postEmployeeInsertService_response_400:
  type: object
  properties:
    message:
      type: string
  example:
    message: Employee insert failed
- postEmployeeInsertService_response_500:
  type: object
  properties:
    message:
      type: string
  example:
    message: Severe Error Occurred
- postEmployeeInsertService_response_200:
  type: object
  properties:
    message:
      type: string
  example:
    message: Record successfully inserted
Ln 366, Col 44 100% Windows (CRLF) UTF-8

```

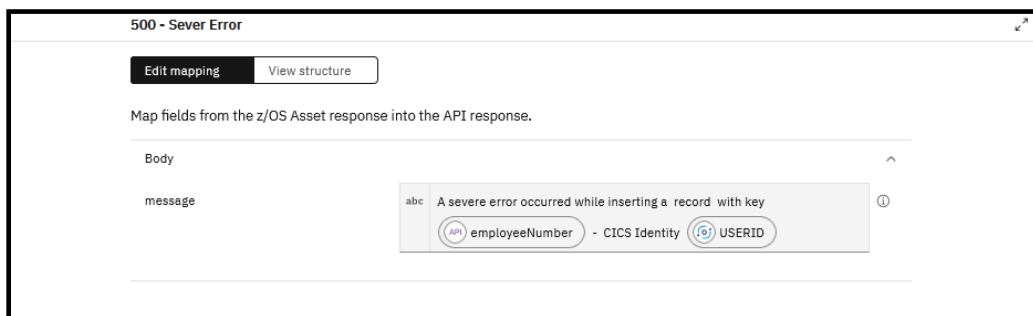
35. Select the response for *400 Add failed* response mapping and paste the text below in the *message* area.

**Duplicate record key encountered for key  
 {{\$apiRequest.body.EmployeeInsertServiceOperation.employeeData.request.employeeDetails.employeeNumber}} - CICS Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**



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

**A severe error occurred while inserting a record with key  
 {{\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} - CICS  
 Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}  
 {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}:  
 {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}**

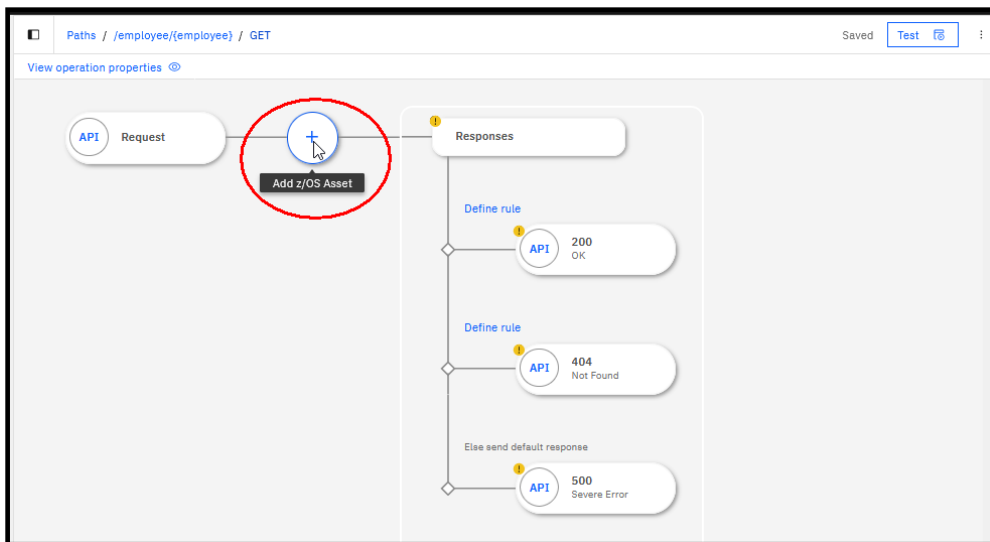


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

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

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

1. Start by adding an existing z/OS Asset for CICS program *CSCVINC* to the GET method by using the same steps as performed before.



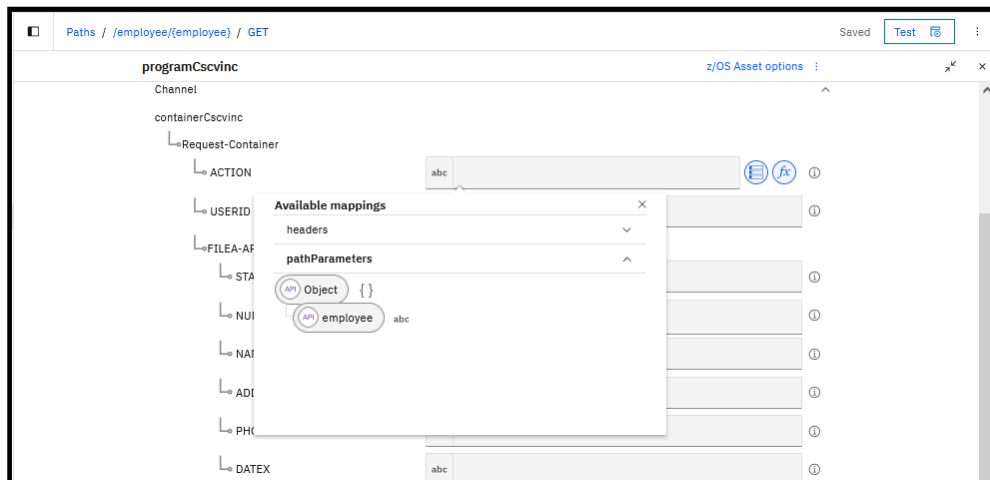
2. On the *Step 1 of 2* page, select *Use existing asset* and click **Next**.



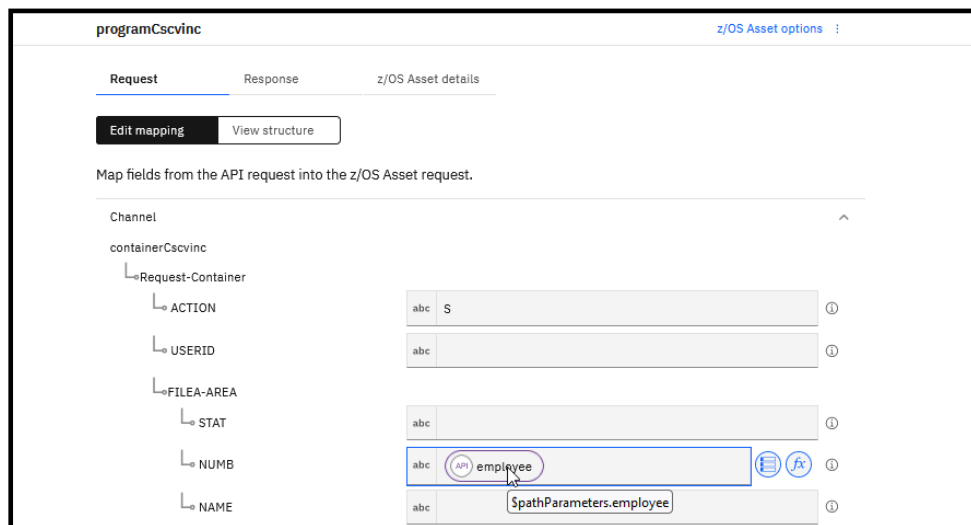
3. On the *Step 2 of 2* page, select the radio button beside *programCSCVINC* and click the **Add z/OS Asset** button.



4. Use the *Insert a mapping tool* to display the API request mapping for this method. Notice that there is no Body mapping. There is only mappings for headers properties and path parameters.



5. So, the only mapping from the API request for this method is to map the path parameter *employee* to the container field *NUMB*. Use your preferred technique to do this mapping and set the value of *ACTION* to *S*.



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

The contents of the *CEIBRESP* and *CEIBRESP2* fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was returned as intended. The application returns the values of the *EIBRESP* and *EIBRESP2* from the EXECI CICS API in the container *CEIBRESP* and *CEIBRESP2* fields. Their values will be zero if a record was successfully read from the VSAM data set. Otherwise, the record did not exist, e.g., *CEIBRESP*=13 and *CEIBRESP2*=80 or some other error occurred will be returned.

Paths / `/employee{employee}` / GET

Responses

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

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

Rule combination  
All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**404 - Not Found**  
If .. equals .. then send 404 response

Rule combination  
All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**500 - Internal Server Error**  
Else send default response

7. Under the *200 – OK* response, use your preferred mapping technique to check the **CEIBRESP** and **CEIBRESP2** response fields are equal to **0**.

**200 - OK**

If channel.containerCscvinc."Response-Container".CEIBRESP equals 0 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 0 then send 200 response

Rule combination

All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	0
CEIBRESP2	Equals	0

Add condition +

8. For the *404 – Not Found* check, use your preferred mapping technique to check the **CEIBRESP** container response field is equal to a value of **13** and the **CEIBRESP2** container response field is equal to a value of **80**.

**404 - Not Found**

If channel.containerCscvinc."Response-Container".CEIBRESP equals 13 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 80 then send 404 response

Rule combination

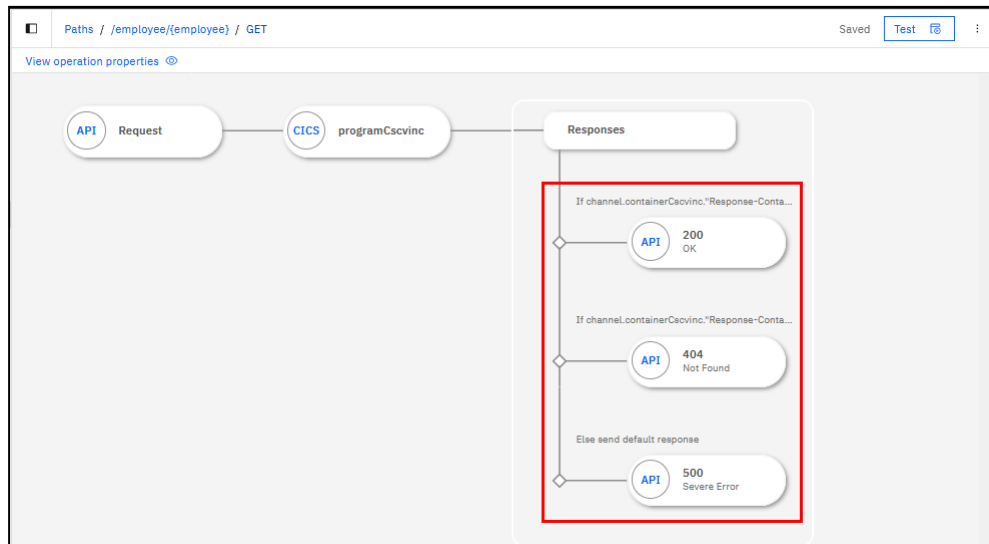
All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	13
CEIBRESP2	Equals	80

Add condition +

9. If none of these connections are met, simply return with a HTTP 500 status code.

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



11. Select the response for *200 OK* and map the fields from the CICS response container. Start by mapping the message API response field the message below:

**Record with key {{\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEA-AREA".NUMB}} was successfully retrieved - CICS Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**

Paths / /employee/{employee} / GET

200 - OK

Edit mapping View structure

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

Body

summary

message

Record with key (6) NUMB was successfully retrieved - CICS Identity (6) USERID

detail

EmployeeSelectServiceOperationResponse

employeeData

response

employeeDetails

employeeNumber

name

address

phoneNumber

date

amount

comment

12. Complete the mapping for the other container fields to the corresponding API response message fields.

Paths / /employee/{employee} / GET

200 - OK

Edit mapping View structure

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

Body

summary

message

Record with key (6) NUMB was successfully retrieved - CICS Identity (6) USERID

detail

EmployeeSelectServiceOperationResponse

employeeData

response

employeeDetails

employeeNumber

name

address

phoneNumber

date

amount

comment

(6) NUMB (6) NAME (6) ADDR (6) PHONE (6) DATEX (6) AMOUNT (6) COMMENT

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

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

404 - Not Found

Edit mapping View structure

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

Body

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

Notice that the mapping for the property in the message was from the API request message and not the response container field. Since the record was not found, the response container field for NUMB is empty.

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

***A severe error occurred while retrieving a record with key {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".FILEA-AREA.NUMB}} - CICS Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}} {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP}}: {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2}}***

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 occurred while retrieving a record with key NUMB - CICS Identity USERID : CEIBRESP : CEIBRESP2 ⓘ

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.

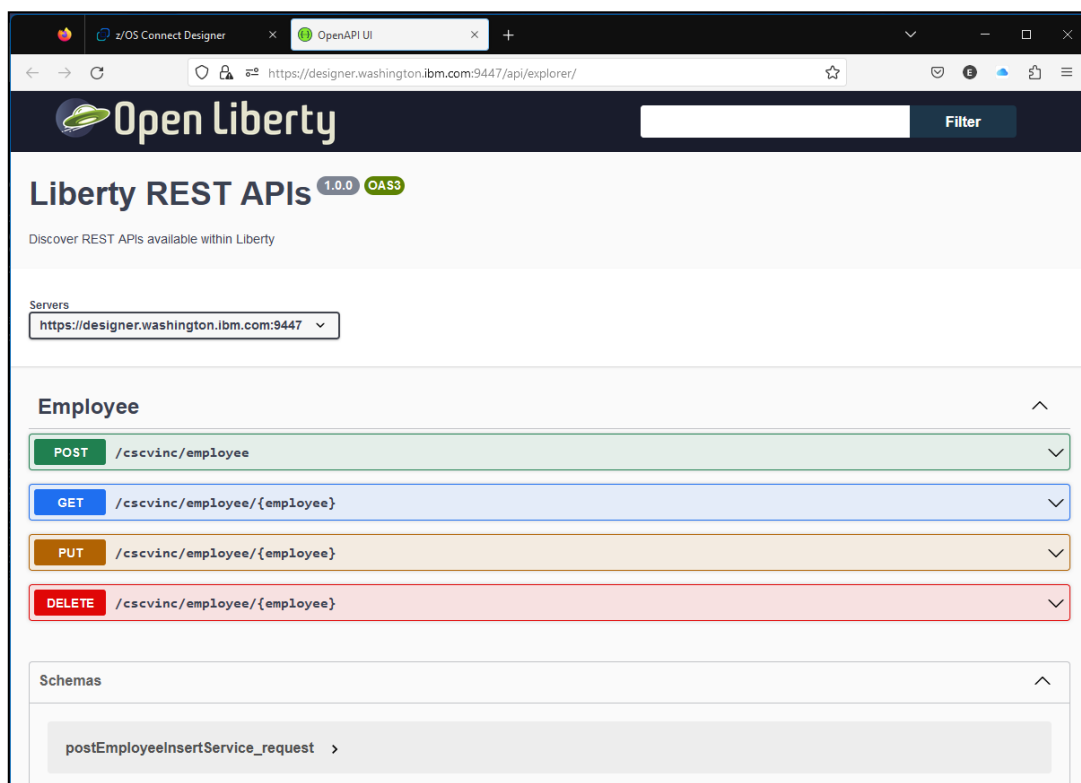
**Tech-Tip:** Note that the exclamation mark has disappeared from *programCscvinc* on the operation page.

## 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 */cscvinc/employee* and the *GET* for */cscvinc/employee/{employee}* have been created. Executing one of the other methods will return an HTTP 404 because the components required to execute these methods cannot be found in the WAR.

Let's test what has been developed so far.

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

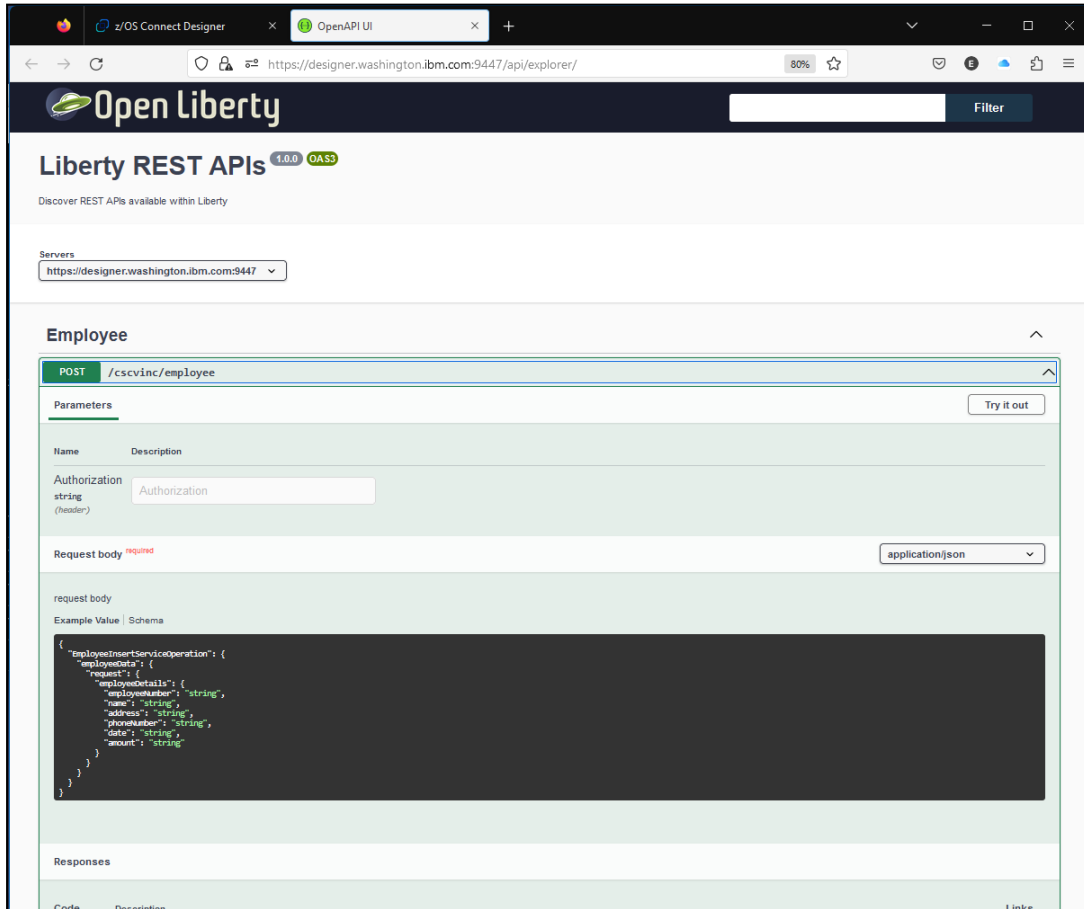


**Tech Tip:** You may be challenged by browser because the digital certificate used by the *Designer* is self-signed Click the **Advanced** button to continue. Scroll down and then click on the **Accept the Risk and Continue** button. Next you may see a prompt you for a userid and password. If you do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

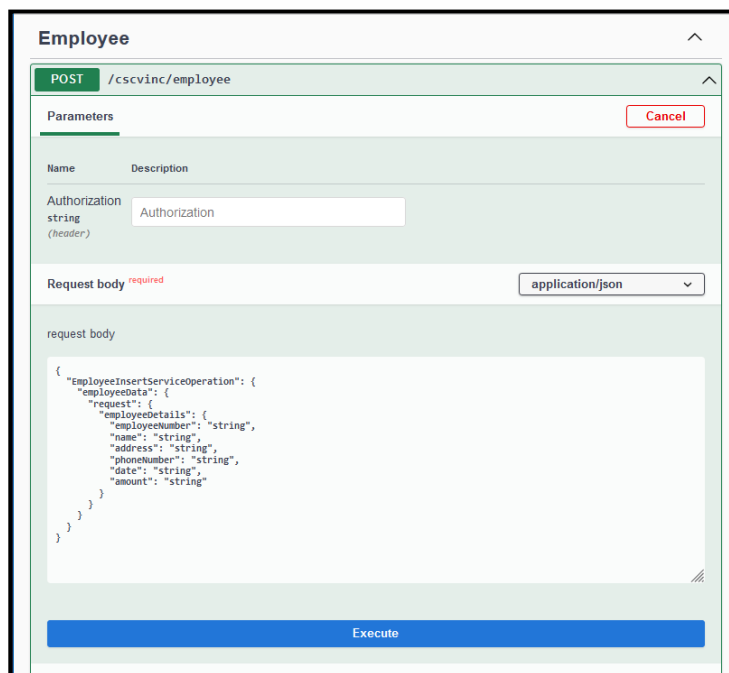
2. Use the pull-down arrow in the *Servers* box at the top of the page and select

<https://designer.washington.ibm.com:9447>

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



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



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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948488",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

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 HTTP 200 - success message.

The screenshot displays the IBM z/OS Connect API client interface. At the top, there are 'Execute' and 'Clear' buttons. Below this, the 'Responses' section is visible. The 'Curl' section shows the command used to execute the request. The 'Request URL' is 'https://designer.ibm.com:9447/cscvinc/employee'. The 'Server response' section shows a '200' status code. The 'Response body' contains a JSON message: '{"message": "Record with key 948478 was successfully inserted - CICS Identity CICSUSER"}'. The 'Response headers' section lists various headers including 'content-language: en-US', 'content-length: 89', 'content-type: application/json', 'date: Thu, 22 Sep 2022 19:31:59 GMT', 'x-firefox-spdy: h2', and 'x-powered-by: Servlet/4.0'.

```
curl -X 'POST' \
  'https://designer.ibm.com:9447/cscvinc/employee' \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
    "EmployeeInsertServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "date": "12/31/22",
            "amount": "$100.00",
            "address": "RTP NC",
            "phoneNumber": "0065",
            "name": "M Johnson"
          }
        }
      }
    }
  }'
```

Request URL

```
https://designer.ibm.com:9447/cscvinc/employee
```

Server response

Code Details

200

Response body

```
{
  "message": "Record with key 948478 was successfully inserted - CICS Identity CICSUSER"
}
```

Response headers

```
content-language: en-US
content-length: 89
content-type: application/json
date: Thu, 22 Sep 2022 19:31:59 GMT
x-firefox-spdy: h2
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 VSAM data set so the request failed with an HTTP 400 – bad request.

Responses

Curl

```
curl -X 'POST' \
  'https://designer.ibm.com:9447/cscvinc/employee' \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
    "EmployeeInsertServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "date": "12/31/22",
            "amount": "$100.00",
            "address": "RTP NC",
            "phoneNumber": "0065",
            "name": "M Johnson"
          }
        }
      }
    }
  }'
```

Request URL

https://designer.ibm.com:9447/cscvinc/employee

Server response

Code	Details
400	<p>Error: Bad Request</p> <p>Response body</p> <pre>{   "message": "Duplicate record key encountered for key 948478 - CICS Identity CICSUSER" }</pre> <p>Download</p>

Response headers

8. Scroll down and click on *GET /cscvinc/employee/{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.

Responses

Curl

```
curl -X 'GET' \
  'https://designer.ibm.com:9447/cscvinc/employee/948478' \
  -H 'accept: application/json'
```

Request URL

https://designer.ibm.com:9447/cscvinc/employee/948478

Server response

Code	Details
200	<p>Response body</p> <pre>{   "summary": {     "message": "Record with key 948478 was successfully retrieved - CICS identity CICSUSER"   },   "detail": {     "EmployeeSelectServiceOperationResponse": {       "employeeData": {         "response": {           "employeeDetails": {             "employeeNumber": "948478",             "name": "name",             "address": "RTP NC",             "phoneNumber": "0065",             "date": "12/31/22",             "amount": "\$100.00",             "comment": "22-09-22"           }         }       }     }   } }</pre> <p>Download</p> <p>Response headers</p> <pre>content-language: en-US content-length: 348 content-type: application/json date: Thu, 22 Sep 2022 19:36:19 GMT x-firefox-spdy: h2 x-powered-by: Servlet/4.0</pre>

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

The screenshot displays a REST client interface with the following sections:

- Responses**: A header for the response details.
- Curl**: A text box containing the command:

```
curl -X 'GET' \
'https://designer.ibm.com:9447/cscvinc/employee/121212' \
-H 'accept: application/json'
```
- Request URL**: A text box containing the URL:

```
https://designer.ibm.com:9447/cscvinc/employee/121212
```
- Server response**: A section showing the response details.
- Code**: A table with one row:

Code	Details
404	Error: Not Found
- Response body**: A text box containing the JSON response:

```
{
  "message": "Record for employee number 121212 was not found"
}
```
- Response headers**: A text box containing the headers:

```
content-language: en-US
content-length: 62
content-type: application/json
date: Thu, 22 Sep 2022 19:37:28 GMT
x-firefox-spdy: h2
x-powered-by: Servlet/4.0
```

See the table at page 82 for a list of the employees in the VSAM dataset.

## Complete 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 /employee/{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 /employee/{employee}

1. Start by adding the existing z/OS Asset for *programCscvinc* to this method.
2. Now map the API path parameter field *employee* to the CICS container field *Numb* as shown below. Next set the ACTION container field to *U*.

3. Map the API request message body fields *status*, *name*, *address*, *phoneNumber*, *date*, *amount*, and *COMMENT* to the CICS container fields *STAT*, *NAME*, *ADDRX*, *PHONE*, *DATEX*, *AMOUNT* and *COMMENT* as shown below.

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

The contents of the *CEIBRESP* and *CEIBRESP2* fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was updated as intended. The application returns the values of the EIBRESP and EIBRESP2 from the EXECI CICS API in the container *CEIBRESP* and *CEIBRESP2* fields. Their values will be zero if a record was successfully updated in the VSAM data set. Otherwise, the record did not exist, e.g., *CEIBRESP*=13 and *CEIBRESP2*=80 or some other error occurred will be returned.

**Responses**

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

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

Rule combination: All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**404 - Not Found**  
If .. equals .. then send 404 response

Rule combination: All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**500 - Internal Server Error**  
Else send default response

5. Under the *200 – OK* response, use your preferred mapping technique to check the **CEIBRESP** and **CEIBRESP2** response fields are equal to **0**.

**200 - OK**

If channel.containerCscvinc."Response-Container".CEIBRESP equals 0 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 0 then send 200 response

Rule combination

All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	0
CEIBRESP2	Equals	0

Add condition +

6. For the *404 – Not Found* check, use your preferred mapping technique to check the **CEIBRESP** container response field is equal to a value of **13** and the **CEIBRESP2** container response field is equal to a value of **80**.

**404 - Not Found**

If channel.containerCscvinc."Response-Container".CEIBRESP equals 13 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 80 then send 404 response

Rule combination

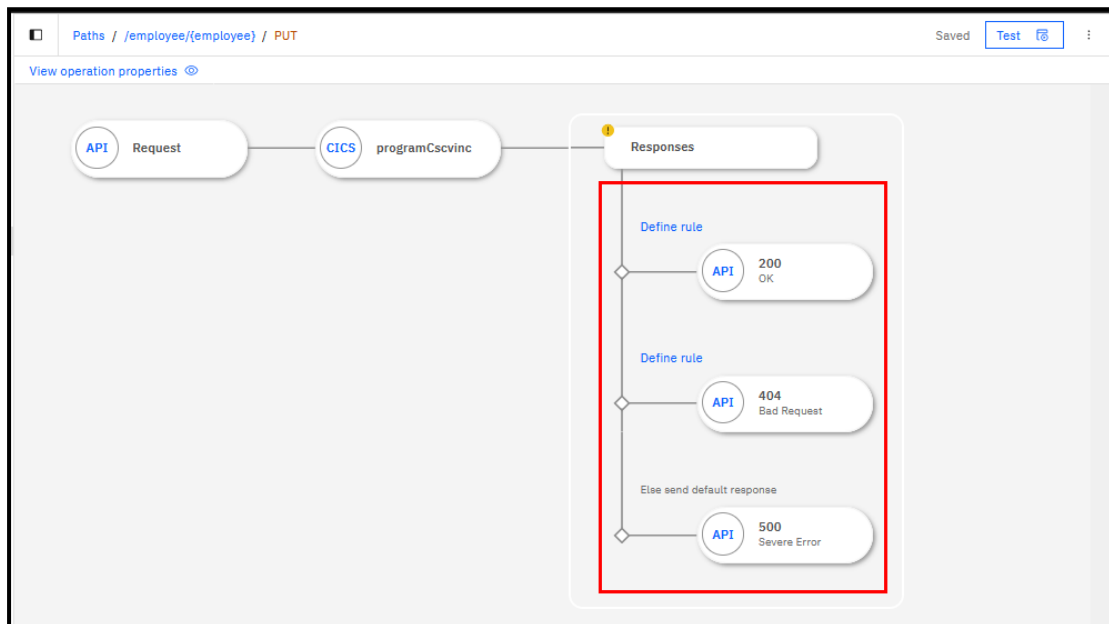
All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	13
CEIBRESP2	Equals	80

Add condition +

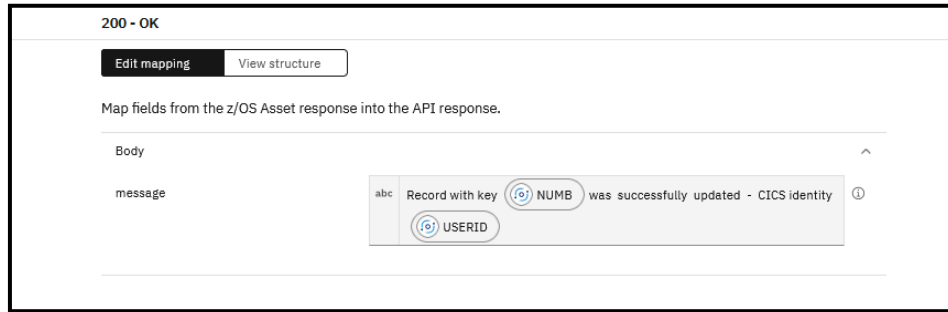
7. If none of these connections are met, simply return with a HTTP 500 status code.

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



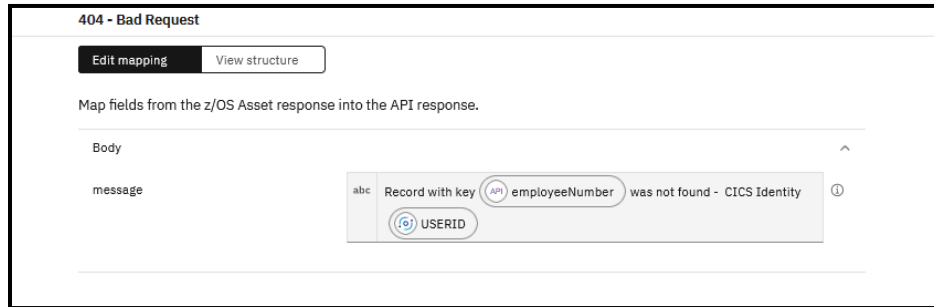
9. Select the response for *200 OK* paste the text below in the *message* area.

**Record with key {{\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILE-AREA".NUMB}} was successfully updated - CICS identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**



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

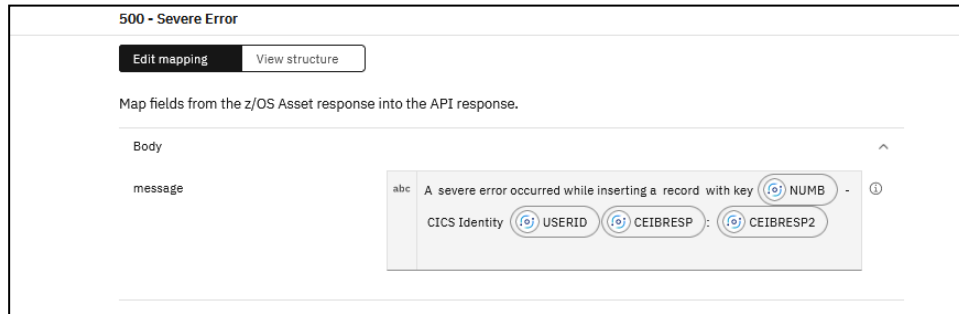
**Record with key {{\$apiRequest.body.EmployeeUpdateServiceOperation.employeeData.request.employeeDetails.employeeNumber}} was not found - CICS Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**



Notice that the mapping for the *employeeNumber* in the message was from the API request message and not the CICS response container.

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

```
A severe error occurred while inserting a record with key  
{{ $zosAssetResponse.channel.containerCscvinc."Response-Container".FILEA-AREA.NUMB }} -  
CICS Identity {{ $zosAssetResponse.channel.containerCscvinc."Response-Container".USERID }}  
{{ $zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP }} :  
{{ $zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2 }}
```

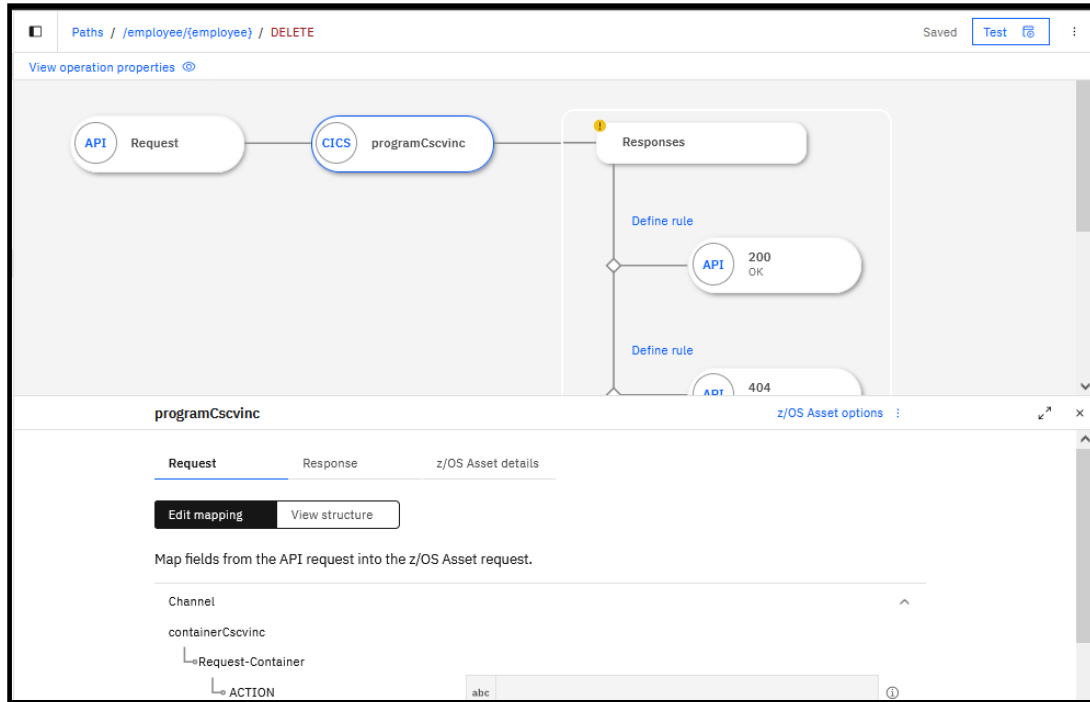


The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate which subcomponent still has an exclamation mark and resolve the issue.

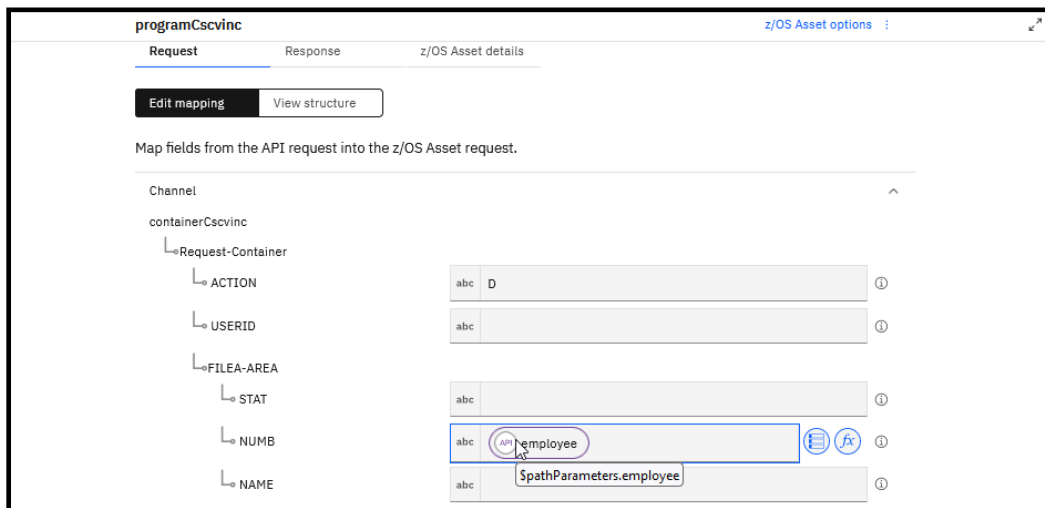
## Configure the *DELETE* method for URI path */employee/{employee}*

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

1. Start by adding the existing *programCscvinc* z/OS Asset to this method.



2. Now map the API path parameter field *employee* to the CICS container field *Numb* as shown below and enter a *D* in the *ACTION* field of the request container.





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

The contents of the *CEIBRESP* and *CEIBRESP2* fields in the CICS response container from the CICS application determine whether the request was successful or not. The first check is to see if the record was deleted as intended. The application returns the values of the EIBRESP and EIBRESP2 from the EXECI CICS API in the container *CEIBRESP* and *CEIBRESP2* fields. Their values will be zero if a record was successfully deleted from the VSAM data set. Otherwise, the record did not exist, e.g., CEIBRESP=13 and CEIBRESP2=80 or some other error occurred will be returned.

Paths / `/employee/{employee}` / DELETE

Responses

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

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

Rule combination: All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**404 - Not Found**  
If .. equals .. then send 404 response

Rule combination: All the following are true

Response field	Predicate	Value
Input	Equals	Input

Add condition +

**500 - Internal Server Error**  
Else send default response

4. Under the *200 – OK* response, use your preferred mapping technique to check the **CEIBRESP** and **CEIBRESP2** response fields are equal to **0**.

**200 - OK**

If channel.containerCscvinc."Response-Container".CEIBRESP equals 0 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 0 then send 200 response

Rule combination

All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	0
CEIBRESP2	Equals	0

Add condition +

5. For the *404 – Not Found* check, use your preferred mapping technique to check the **CEIBRESP** container response field is equal to a value of **13** and the **CEIBRESP2** container response field is equal to a value of **80**.

**404 - Not Found**

If channel.containerCscvinc."Response-Container".CEIBRESP equals 13 and channel.containerCscvinc."Response-Container".CEIBRESP2 equals 80 then send 404 response

Rule combination

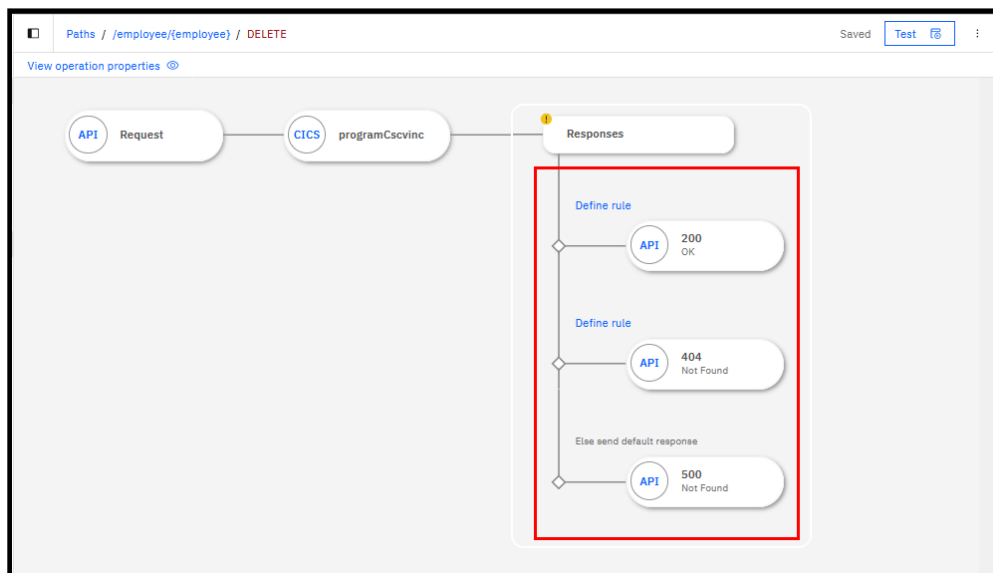
All the following are true

Response field	Predicate	Value
CEIBRESP	Equals	13
CEIBRESP2	Equals	80

Add condition +

6. If none of these connections are met, simply return with a HTTP 500 status code.

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



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

**Record with key {{\$zosAssetResponse.channel.containerCscvinc."Response-Container"."FILEAREA".NUMB}} was successfully deleted - CICS identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**

200 - OK

Edit mapping View structure

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

Body

message

abc Record with key (NUMB) was successfully deleted - CICS identity (USERID)

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

**Record with key {{\$apiRequest.pathParameters.employee}} was not found - CICS Identity {{\$zosAssetResponse.channel.containerCscvinc."Response-Container".USERID}}**

404 - Not Found

Edit mapping View structure

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

Body

message

abc Record with key (API) employee was not found - CICS Identity (USERID)

Notice that the mapping for the employeeNumber in the message was from the API request message and not the CICS response container.

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

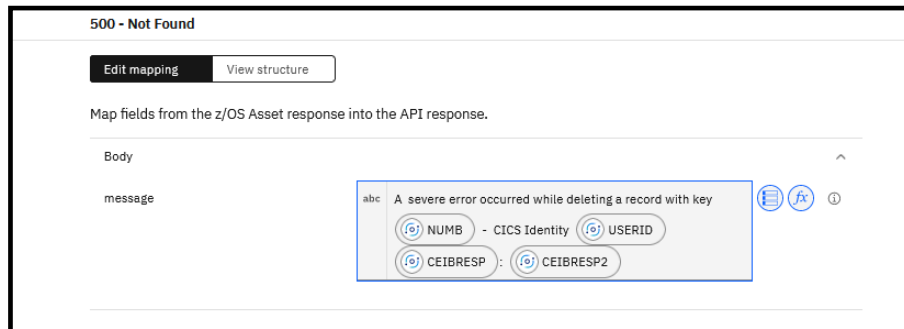
```
A severe error occurred while deleting a record with key  

{{ $zosAssetResponse.channel.containerCscvinc."Response-Container".FILEA-AREA.NUMB }} -  

CICS Identity {{ $zosAssetResponse.channel.containerCscvinc."Response-Container".USERID }}  

{{ $zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP }} :  

{{ $zosAssetResponse.channel.containerCscvinc."Response-Container".CEIBRESP2 }}
```



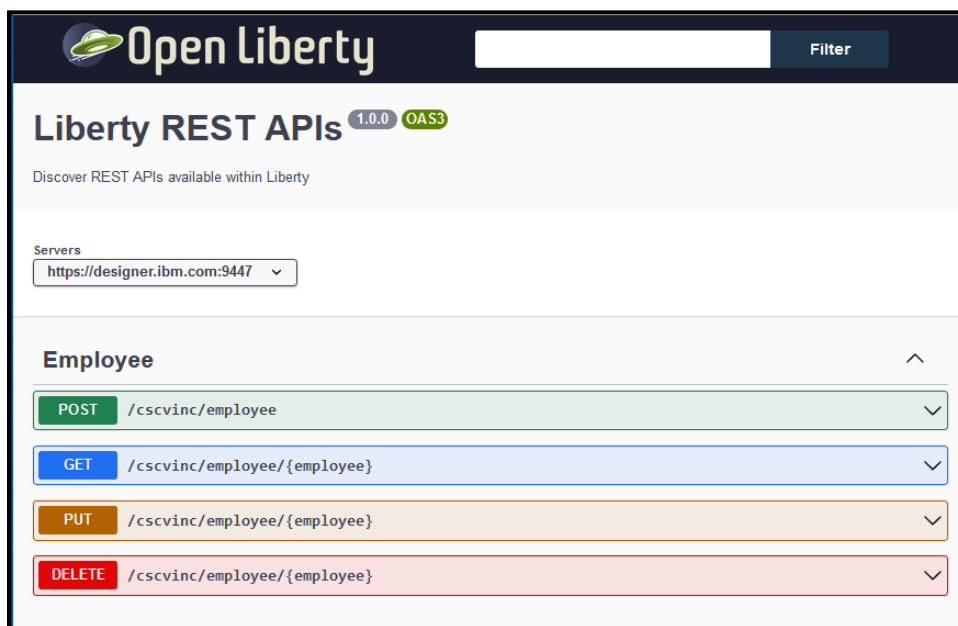
The completes the configuration of this method of this URI path of the API. All the exclamation marks for this method should now have disappeared. If not investigate the subcomponent that still has an exclamation mark and resolve the issue.

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

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

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

We know the URI paths of the API from the initial page of the *API Explorer* displayed when testing in the *Designer*. From this page the first part of the URL can be determined, e.g., <https://designer.washington.ibm.com:9447>. 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:9447/cscvinc/employee/{employee}>

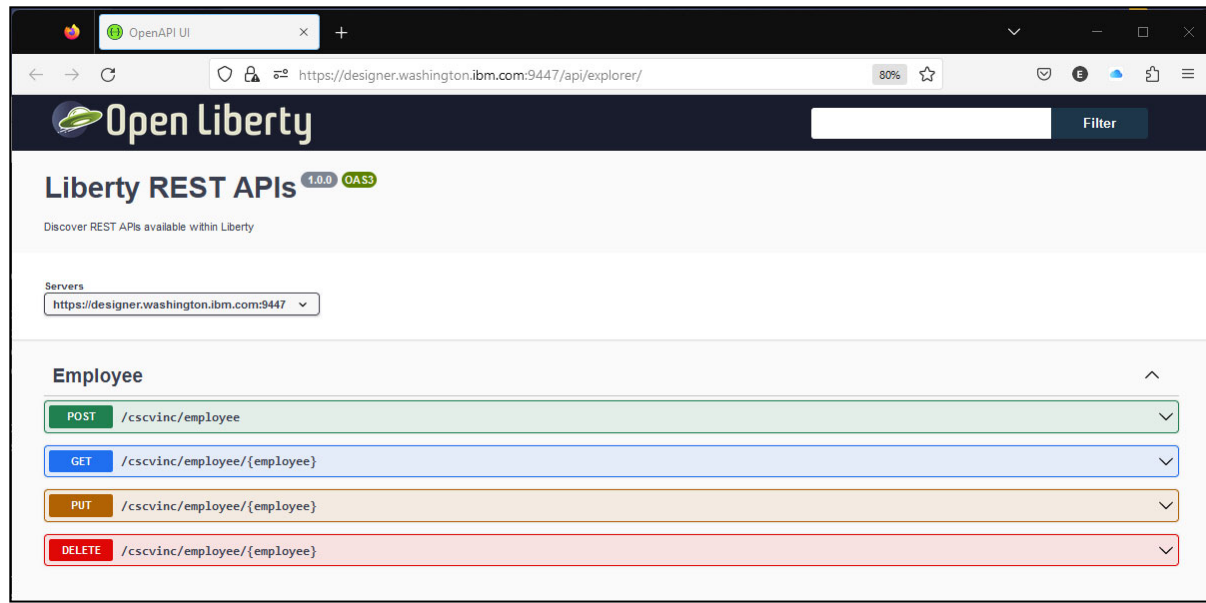


From this display, the methods and URLs required to access the API deployed in this container are:

- POST <https://designer.washington.ibm.com:9447/cscvinc/employee>
- GET <https://designer.washington.ibm.com:9447/cscvinc/employee/{employee}>
- PUT <https://designer.washington.ibm.com:9447/cscvinc/employee/{employee}>
- DELETE <https://designer.washington.ibm.com:9447/cscvinc/employee/{employee}>

This section primarily covers the testing of the optional *DELETE* and *PUT* methods. The tests for the POST and GET methods have already been covered in section *Testing the API's POST and GET methods* on page 38 and can be repeated using the *API Explorer* as described in this section.

1. Using the Firefox browser, go to URL <https://designer.washington.ibm.com:9447/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 **fredpwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.

2. Click on *DELETE* */cscvinc/employee/{employee}* URI path to display the request parameters.

The screenshot shows the OpenAPI interface for the **DELETE** endpoint `/cscvinc/employee/{employee}`. The **Parameters** tab is active, displaying a table with two parameters:

Name	Description
Authorization string (header)	Authorization
<b>employee</b> * required string (path)	employee

A "Try it out" button is located in the top right corner.

3. Press the **Try it out** button.

4. Enter a value of **948488** and press the **Execute** button.

5. If the record exists, a message like the one below will appear.

The screenshot shows the **Responses** tab for the **DELETE** endpoint. It displays the following information:

- Curl:** `curl -X 'DELETE' \ 'https://designer.ibm.com:9447/cscvinc/employee/948488' \ -H 'accept: application/json'`
- Request URL:** `https://designer.ibm.com:9447/cscvinc/employee/948488`
- Server response:**
  - Code:** 200
  - Response body:** `{ "message": "Record with key 948488 was successfully deleted - CICS identity CICSUSER" }`
  - Response headers:**

```
cache-control: no-cache=set-cookie,set-cookie2
content-language: en-US
content-length: 92
content-type: application/json
date: Mon,26 Sep 2022 17:33:02 GMT
expires: Thu,01 Dec 1994 16:00:00 GMT
x-firefox-spdy: h2
x-powered-by: Servlet/4.0
```

6. Otherwise, a message that the record was not found will appear.

The screenshot shows the **Responses** tab for the **DELETE** endpoint, displaying an error response:

- Code:** 404
- Error:** Not Found
- Response body:** `{ "message": "Record with key 948488 was not found - CICS Identity CICSUSER" }`
- Response headers:**

```
content-language: en-US
content-length: 77
content-type: application/json
date: Sun,17 Jul 2022 18:47:16 GMT
x-firefox-spdy: h2
x-powered-by: Servlet/4.0
```

7. Click on *POST /cscvinc/employee* URI path to display the request body view of the URI path.

**Employee**

**POST** /cscvinc/employee

Parameters Try it out

Name	Description
Authorization	

string (header)

Request body **required** application/json

request body

Example Value | Schema

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "string",
          "name": "string",
          "address": "string",
          "phoneNumber": "string",
          "date": "string",
          "amount": "string"
        }
      }
    }
  }
}
```

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

**Employee**

**POST** /cscvinc/employee

Parameters Cancel

Name	Description
Authorization	

string (header)

Request body **required** application/json

request body

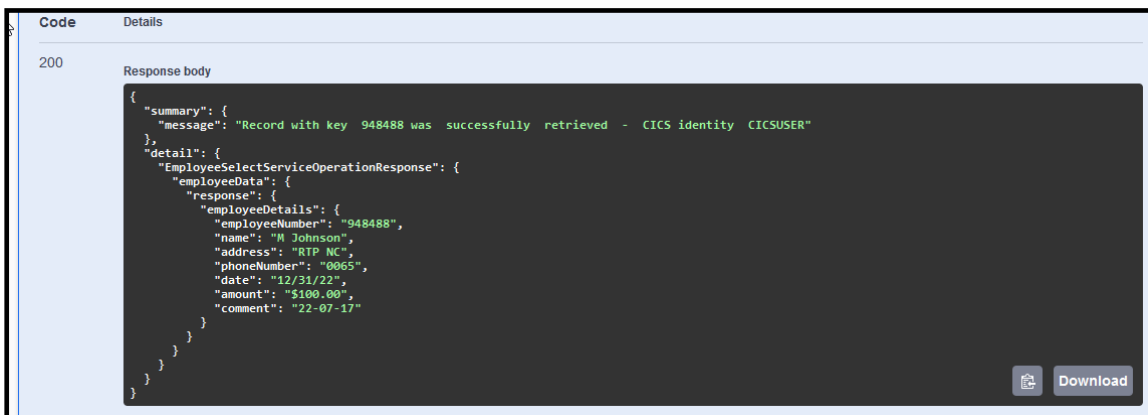
```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "string",
          "name": "string",
          "address": "string",
          "phoneNumber": "string",
          "date": "string",
          "amount": "string"
        }
      }
    }
  }
}
```



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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948488",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

10. Use a Click on *GET* for URI path */employee/{employee}* to display the GET method parameters. Press the **Try it out** button Enter **948488** for the employee and press the **Execute** button to display this record.



The screenshot shows a REST client interface with a 'Code' tab selected. The status is 200. The response body is displayed in a dark-themed editor. The JSON response contains a summary message and detailed employee information.

```
{
  "summary": {
    "message": "Record with key 948488 was successfully retrieved - CICS identity CICSUSER"
  },
  "detail": {
    "EmployeeSelectServiceOperationResponse": {
      "employeeData": {
        "response": {
          "employeeDetails": {
            "employeeNumber": "948488",
            "name": "M Johnson",
            "address": "RTP NC",
            "phoneNumber": "0065",
            "date": "12/31/22",
            "amount": "$100.00",
            "comment": "22-07-17"
          }
        }
      }
    }
  }
}
```

At the bottom right of the editor, there are icons for a file and a 'Download' button.

Security was enabled in the original specification document, so you will be required to sign in with a RACF identity. Use **Fred** for the *Username* and **fred** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

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

12. Expand the *PUT* method and press the **Try it out** button.

13. Enter **948488** in the area beside *employee* and enter the JSON below in the request body area and press the **Execute** button.

```
{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
```

Curl

```
curl -X 'PUT' \
  'https://designer.washington.ibm.com:9447/employee/948488' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "EmployeeUpdateServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "status": "s",
            "name": "A Johnson",
            "address": "Apex NC",
            "phoneNumber": "0065",
            "date": "01/31/23",
            "amount": "500",
            "COMMENT": "updated"
          }
        }
      }
    }
  }'
```

Request URL

https://designer.washington.ibm.com:9447/employee/948488

Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Record with key 948488 was successfully updated - CICS identity CICSUSER" }</pre> <p>Response headers</p> <pre>content-language: en-US content-length: 92 content-type: application/json date: Sun, 17 Jul 2022 19:05:33 GMT x-firefox-spdy: h2 x-powered-by: Servlet/4.0</pre>

14. Use the *GET* method to confirm the updates have taken place.

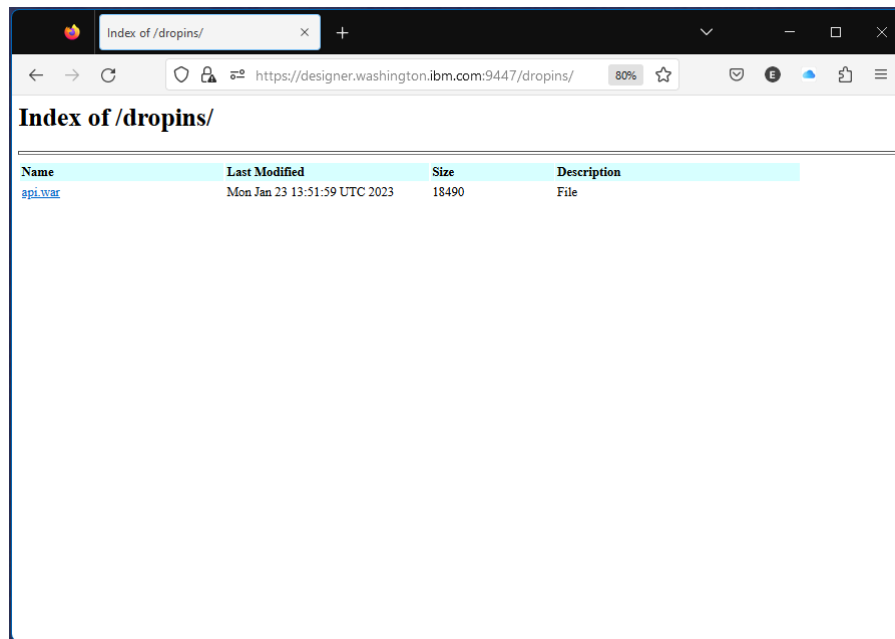
15. Use the *DELETE* method to delete employee **948488** and confirm the record has been deleted.

## 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:9447/dropins/>.



Double click the *api.war* file and save the file in local directory, e.g., *c:/z/openApi3/wars*. Specify a *File* name of *cscvinc.war*.

2. Open a DOS command prompt and use the change directory command to go to directory *C:\z\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 passwrd 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 cscvinc.war
200 Port request OK.
125 Storing data set /var/zcee/openApi3/apps/employee.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="cics" contextRoot="/cscvinc" name="cicsAPI"
location="${server.config.dir}apps/cscvinc.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 */cics* 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 **cicsApi**. The final source is the role name provided in the specification document, **Manager** or **Staff**. So, two EJBRoles need to be defined, **ATSZDFLT.cicsApi.Manager** and **ATSZDFLT.cicsApi.Staff**.

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

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

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

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

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

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

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

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

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

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

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

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

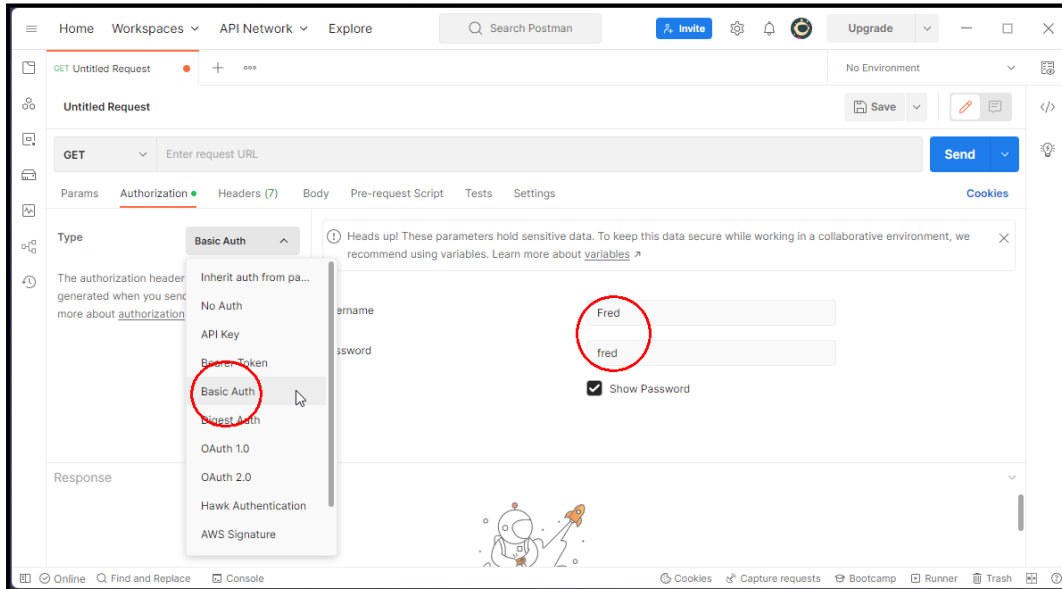
Now we are ready to test the invoking of the methods of this API.

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

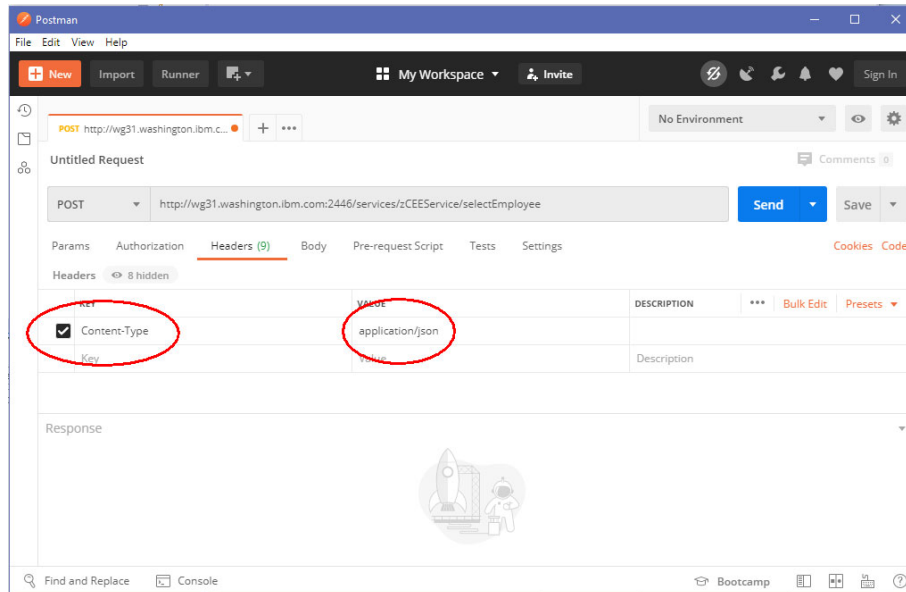
### Using Postman

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 and select the *Authorization* tab to enter an authorization identity and password. Use the pull down arrow to select *Basic Auth* and enter **Fred** as the *Username* and **fred** as the *Password*.

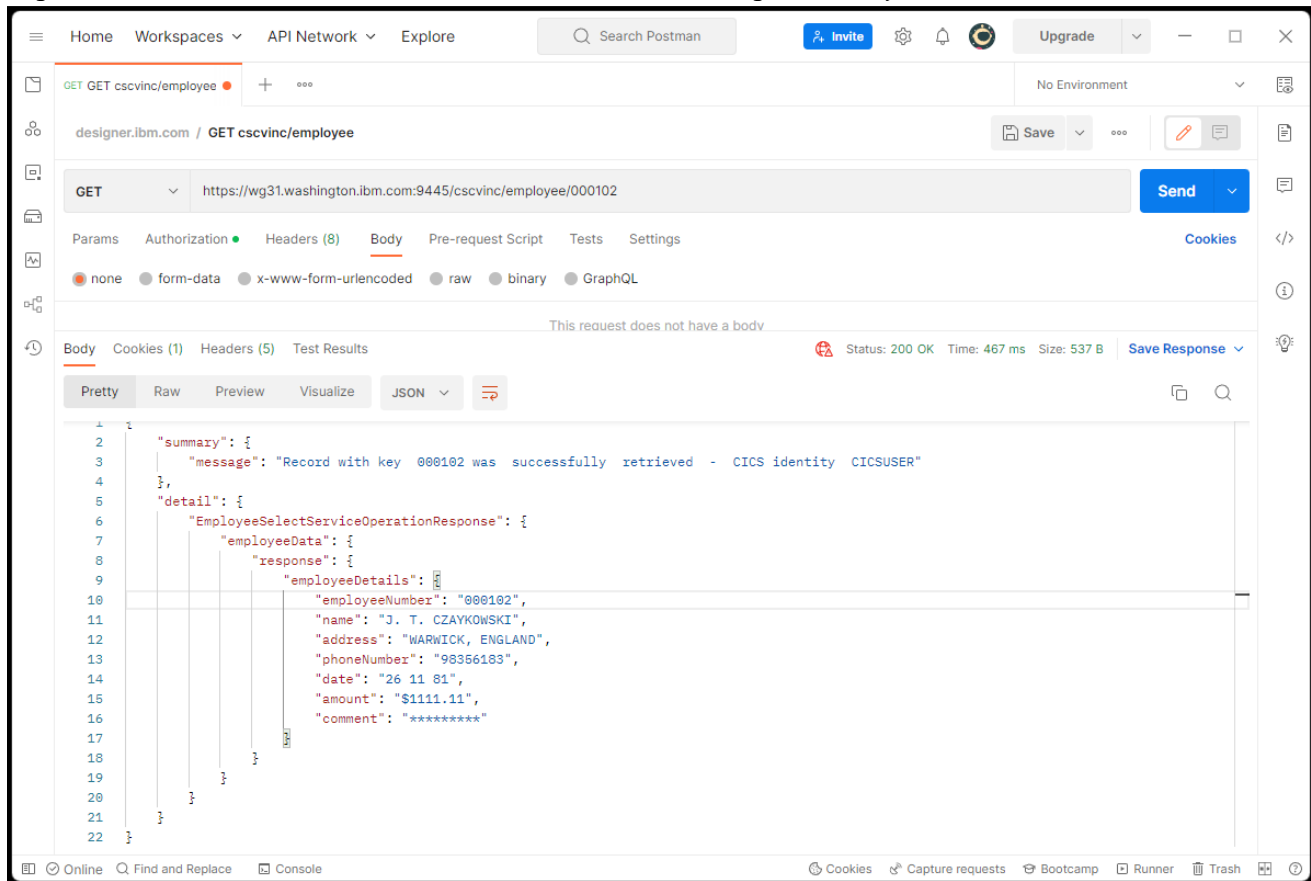


2. Next select the *Headers* tab and under *KEY* use the code assist feature to enter **Content-Type** and under *VALUE* use the code assist feature to enter **application/json**.



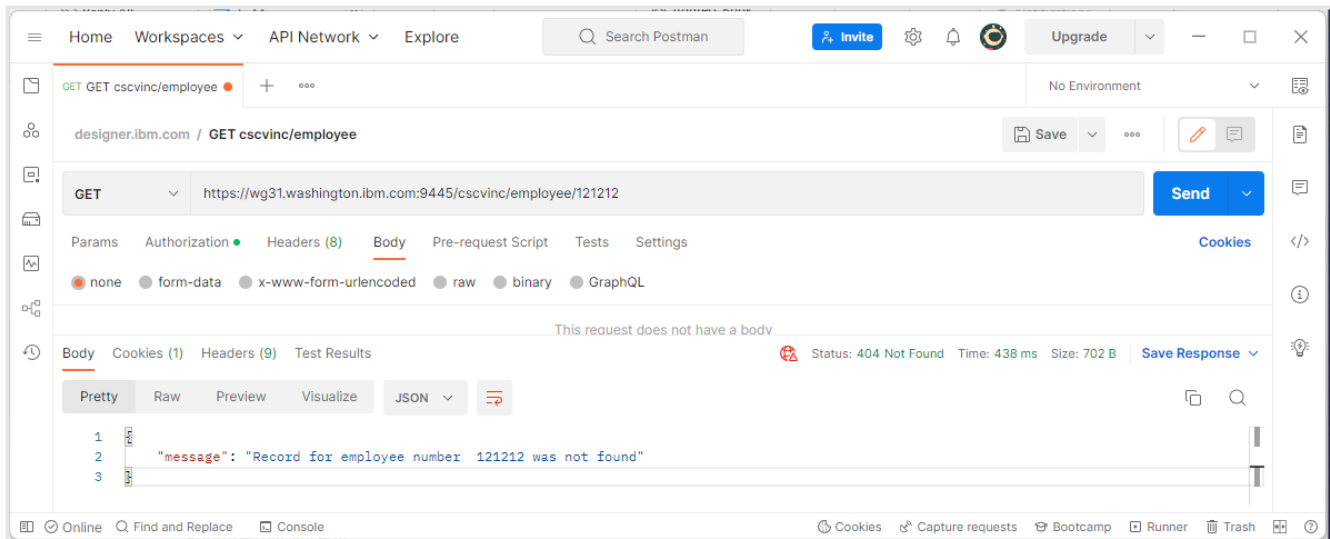
**Tech-Tip:** Code assist simply means that when text is entered in field, all the valid values for that field that match the typed text will be displayed. You can select the desired value for the field from the list displayed and that value will populate that field.

3. Next select the *Body* tab and select the *raw* radio button. Select **GET** and enter <https://wg31.washington.ibm.com:9445/cscvinc/employee/000102> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.



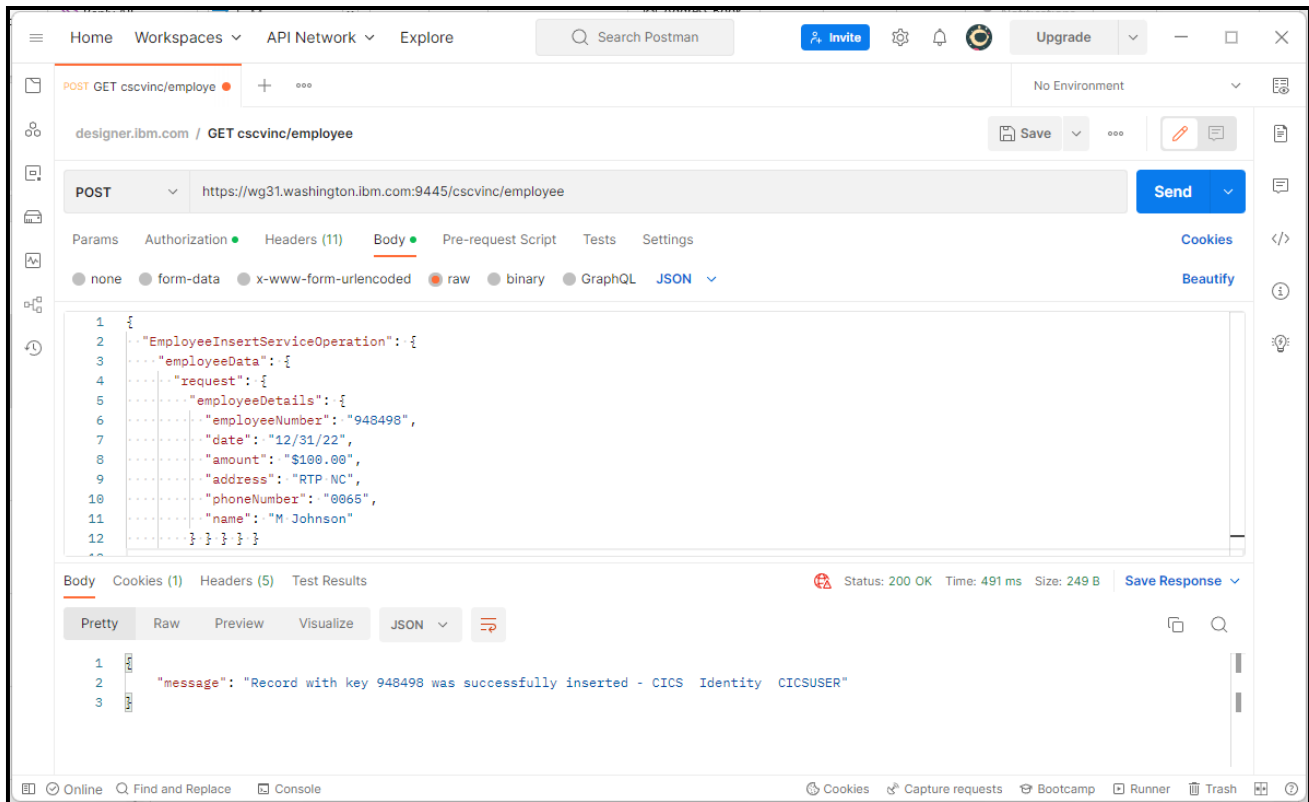


4. Next enter an invalid employee number such as 121212, <https://wg31.washington.ibm.com:9445/cscvinc/employee/121212> in the URL area (see below) and press **Send**. You should see results like below in the response *Body* area.



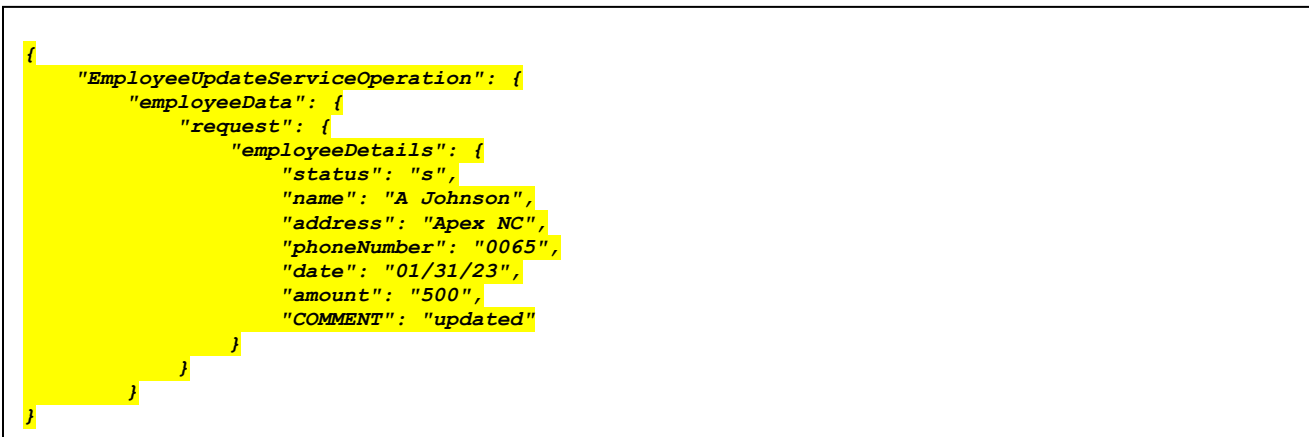
5. Invoke a *POST* with URI path `/cscvinc/employee`, use the JSON below for the request message.

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948498",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

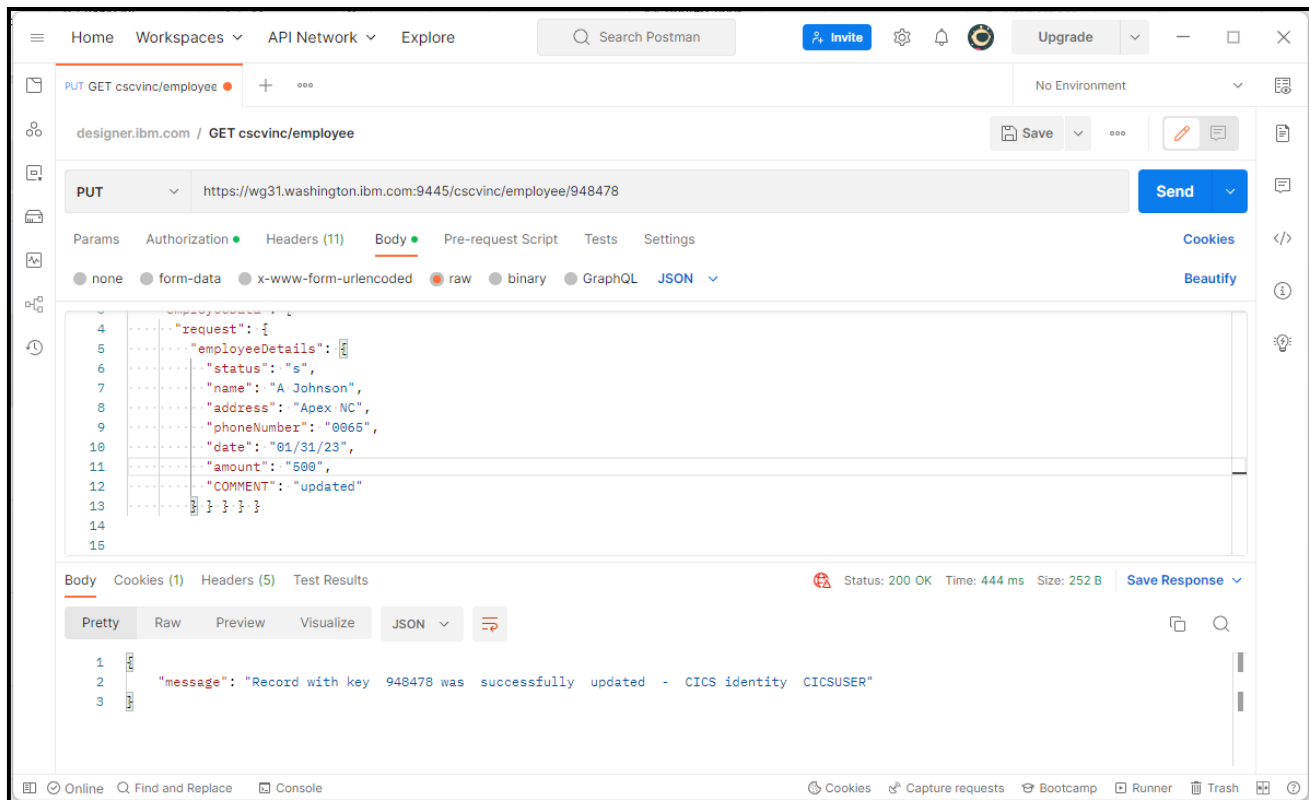


6. Perform a *PUT* with URI path `/cscvinc/employee/{employee}` use this JSON request message.

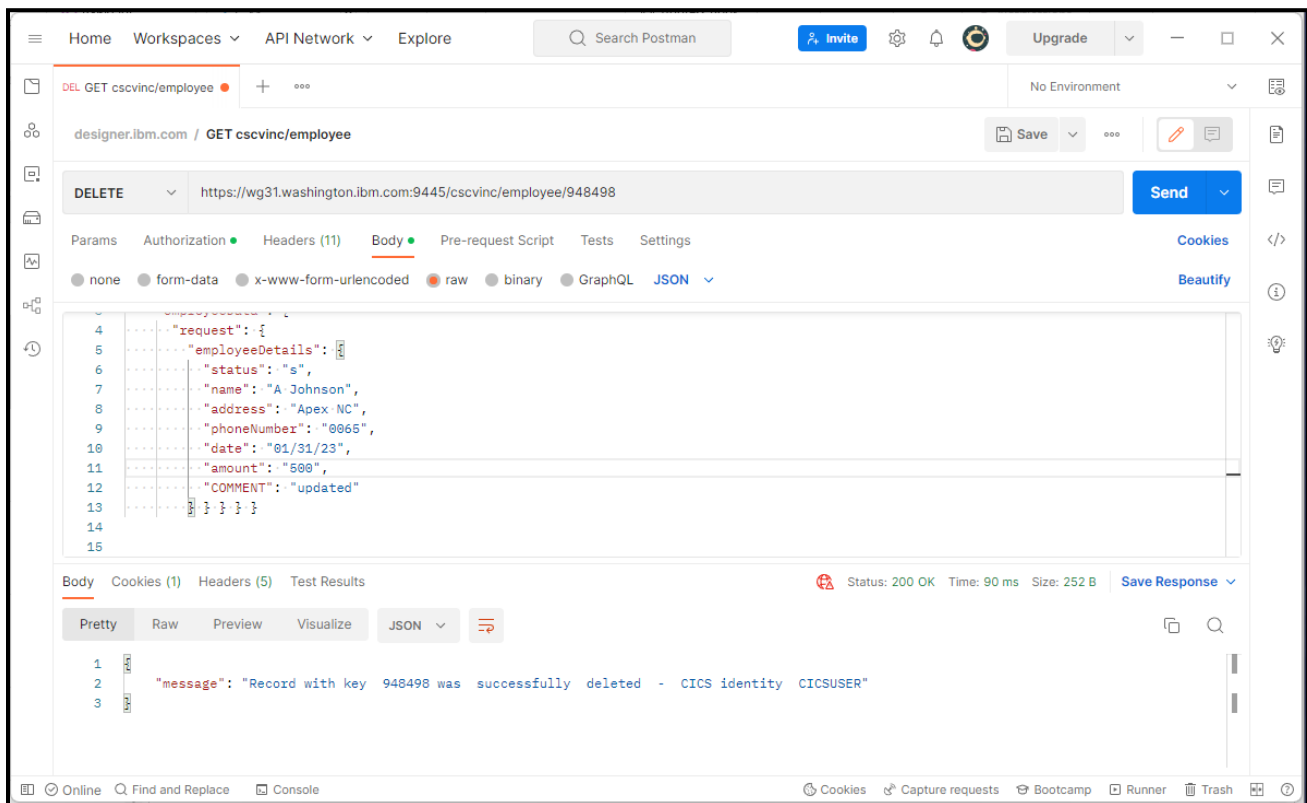
<https://wg31.washington.ibm.com:9445/cscvinc/employee/948498>



## IBM z/OS Connect (OpenAPI 3.0)

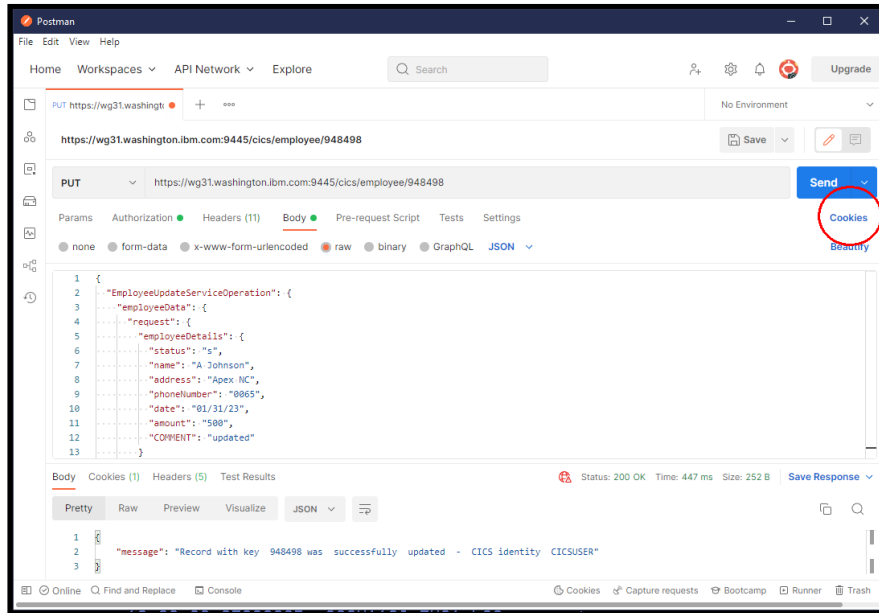


7. Perform a *DELETE* with URI path `/cscvinc/employee/948498`.

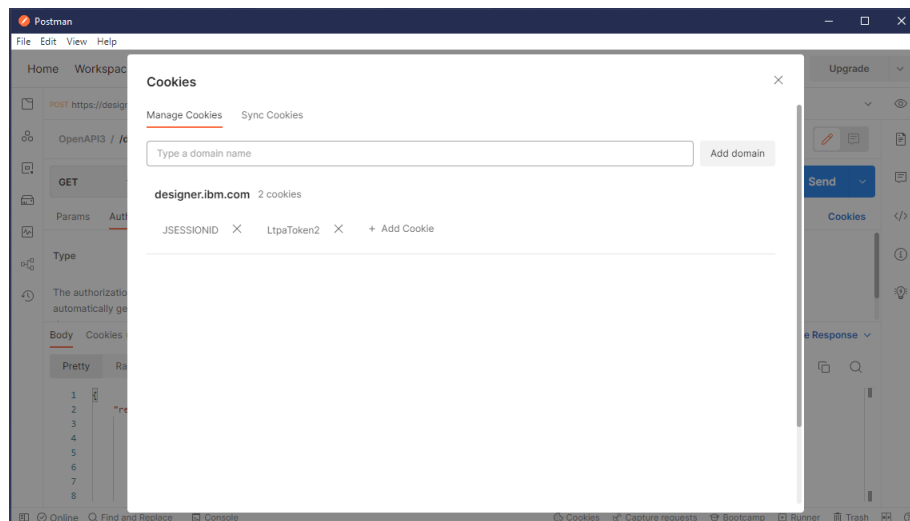


8. Up until this point you have been using the role assigned to user *Fred*. Now experiment using user *user1* and/or *user2*. Before we can use other credentials we have to clear the credentials that cached by *Postman*. Unless this is done, *Postman* will continue to use the credentials for *Fred*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

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



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



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

## Using cURL

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

1. Start a DOS command prompt session and go to directory *c:\z\openapi3*, e.g., *cd \z\openap3*.
2. Enter the *curl* command below and observe the response.

```
curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cscvinc/employee/222222
```

```
c:\z\openApi3>curl -X GET -w " - HTTP CODE %{http_code}" --user Fred:fred --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cscvinc/employee/222222
{"summary":{"message":"Record with key 222222 was successfully retrieved - CICS identity CICSUSER"},"detail":{"EmployeeSelectServiceOperationResponse":{"employeeData":{"response":{"employeeDetails":{"employeeNumber":"222222","name":"DR E. GRIFFITHS","address":"FRANKFURT, GERMANY","phoneNumber":"20034151","date":"26 11 81","amount":"$0022.00","comment":"*****"}}}}}} - HTTP CODE 200
```

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

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

```
curl -X GET -w " - HTTP CODE %{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cscvinc/employee/222222
```

```
c:\z\openApi3>curl -X GET -w " - HTTP CODE %{http_code}" --user user2:user2 --header "Content-Type: application/json" --insecure https://wg31.washington.ibm.com:9445/cscvinc/employee/222222
- HTTP CODE 403
```

A review of the SYSLOG will show that RACF access to the EJBRole protecting this method prevented USER2 from accessing this method.

```
ICH408I USER(USER2 ) GROUP(SYS1 ) NAME(WORKSHOP USER2 )
  ATSZDFLT.cicsAPI.Staff CL(EJBROLE )
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
```

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 @insertCscvinc.json https://wg31.washington.ibm.com:9445/cscvinc/employee/
```

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948478",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

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

```
c:\z\openApi3>curl -X POST -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred --data @insertCscvinc.json https://wg31.washington.ibm.com:9445/cscvinc/employee/ {"message":"Record with key 948478 was successfully inserted - CICS Identity CICSUSER"} - HTTP CODE 200
```

5. Enter the curl command below to invoke the *PUT* method with URI /employee/{employee} to update a record.

```
curl -X PUT -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred --data @updateCscvinc.json https://wg31.washington.ibm.com:9445/cscvinc/employee/948478
```

```
c:\z\openApi3>curl -X PUT -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fred --data @updateCscvinc.json https://wg31.washington.ibm.com:9445/cscvinc/employee/948478 {"message":"Record with key 948478 was successfully updated - CICS identity CICSUSER"} - HTTP CODE 200
```

In the above command, file *updateCscvinc.json* contains this content.

```
{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
```

6. Repeat the GET request to observe that the updates have been applied.

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

```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" -
-insecure --user Fred:fred https://wg31.washington.ibm.com:9445/cscvinc/employee/948478
```

```
c:\z\openApi3>curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json"
--insecure --user Fred:fred https://wg31.washington.ibm.com:9445/cscvinc/employee/948478
{"message":"Record with key 948478 was successfully deleted - CICS identity CICSUSER"} - HTTP CODE
200
```

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

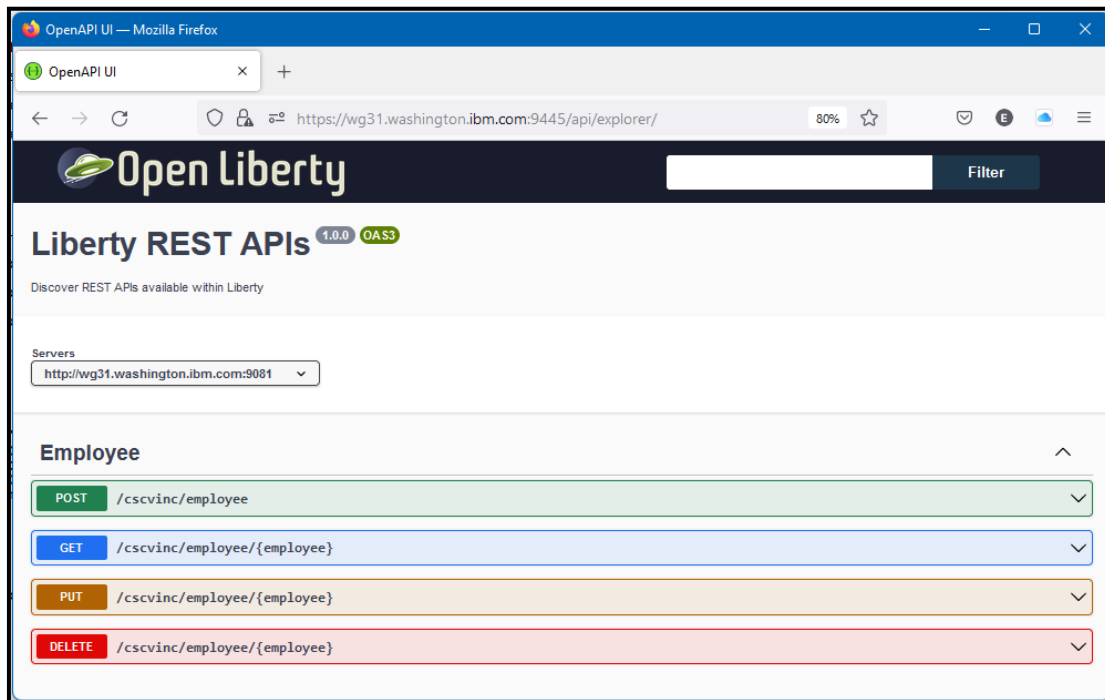
```
curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" -
-insecure --user user1:user1
https://wg31.washington.ibm.com:9445/cscvinc/employee/948478
```

What HTTP status code occurred and why did it occur?

## Using the API Explorer

The API Explorer was used in the Designer to test the APIs as they were being developed. The API Explorer can also be used to test APIs once they have been deployed to the native server.

1. Using the Firefox browser, go to URL <https://wg31.washington.ibm.com:9445/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 **fredpwd** (case matters) and click **OK**. Remember we are using basic security, and this is the user identity and password defined in the server.xml file.



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

<https://wg31.washington.ibm.com:9445>

4. Click on *Post* */cscvinc/employee* URI path to display the request body view of the URI path.

The screenshot displays the Open Liberty REST API Explorer interface. At the top, the 'Liberty REST APIs' header includes version '1.0.0' and 'OAS3' tags. Below this, a 'Servers' dropdown menu is set to 'https://wg31.washington.ibm.com:9445'. The main section is titled 'Employee' and shows the 'POST /cscvinc/employee' endpoint. Under the 'Parameters' tab, an 'Authorization' header is defined as a string. The 'Request body' section is marked as 'required' and set to 'application/json'. It displays an example JSON request body for an 'EmployeeInsertServiceOperation'.

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "string",
          "name": "string",
          "address": "string",
          "phoneNumber": "string",
          "date": "string",
          "amount": "string"
        }
      }
    }
  }
}
```

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

**Employee**

POST /cscvnc/employee

Parameters

Cancel

Name Description

Authorization string (header) Authorization

Request body required application/json

request body

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "string",
          "name": "string",
          "address": "string",
          "phoneNumber": "string",
          "date": "string",
          "amount": "string"
        }
      }
    }
  }
}
```

Execute

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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948478",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

7. 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 **fred** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

8. Scroll down the view and you should see the *Response body* with the expected HTTP 200 - success message.

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  'https://w31.washington.ibm.com:9445/cscvinc/employee' \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
    "employeeInsertServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "date": "12/31/22",
            "amount": "$100.00",
            "address": "RTP NC",
            "phoneNumber": "0000",
            "name": "M Johnson"
          }
        }
      }
    }
  }'
```

Request URL

```
https://w31.washington.ibm.com:9445/cscvinc/employee
```

Server response

Code	Details
200	<div>Response body</div> <pre>{   "message": "Record with key 948478 was successfully inserted - CICS Identity CICSUSER" }</pre> <div>Download</div>
	<div>Response headers</div> <pre>cache-control: no-cache="set-cookie,set-cookie2" content-language: en-US content-length: 89 content-type: application/json date: Tue, 27 Sep 2022 20:05:15 GMT expires: Thu, 01 Dec 1994 16:00:00 GMT x-firefox-spy: h2 x-powered-by: Servlet/4.0</pre>

9. Press the **Execute** button again and observe the results. A row for this employee number already existed in the VSAM data set so the request failed with an HTTP 400 – bad request.

Responses

Curl

```
curl -X 'POST' \
  'https://wg31.washington.ibm.com:9445/cscvinc/employee' \
  -H 'accept: application/json' \
  -H 'content-type: application/json' \
  -d '{
    "employeeInsertServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "date": "12/31/22",
            "amount": "$100.00",
            "address": "RTP NC",
            "phoneNumber": "0065",
            "name": "M Johnson"
          }
        }
      }
    }
  }'
```

Request URL

https://wg31.washington.ibm.com:9445/cscvinc/employee

Server response

Code	Details
400	Error: Bad Request

Response body

```
{
  "message": "Duplicate record key encountered for key 948478 - CICS Identity CICSUSER"
}
```

Download

10. Scroll down and click on *GET /cscvinc/employee/{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.

Responses

Curl

```
curl -X 'GET' \
  'https://wg31.washington.ibm.com:9445/cscvinc/employee/948478' \
  -H 'accept: application/json'
```

Request URL

https://wg31.washington.ibm.com:9445/cscvinc/employee/948478

Server response

Code	Details
200	

Response body

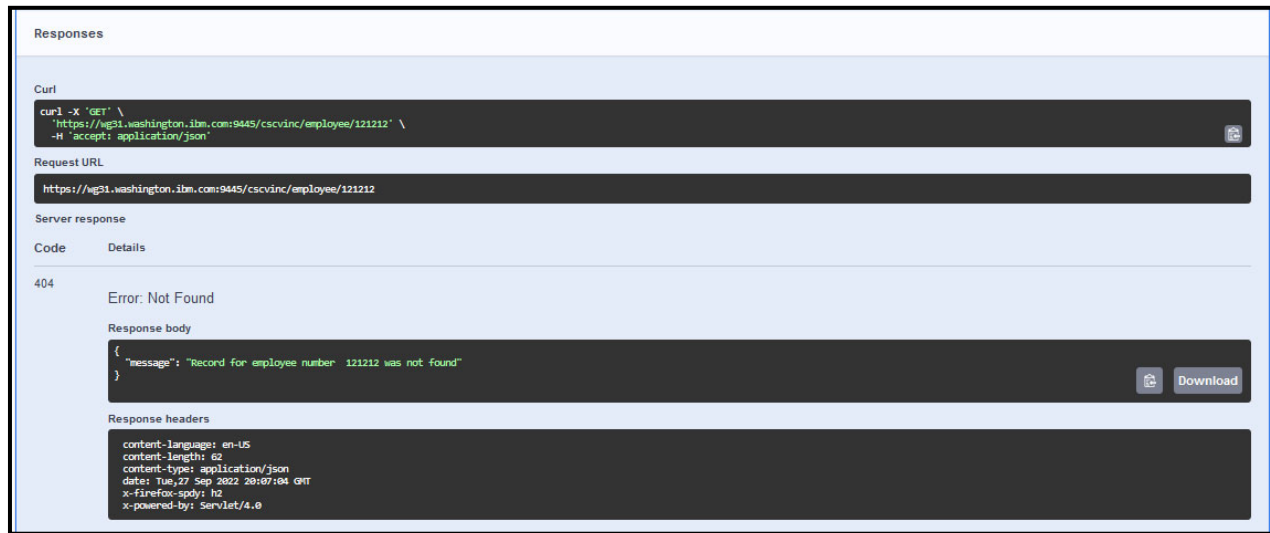
```
{
  "summary": {
    "message": "Record with key 948478 was successfully retrieved - CICS identity CICSUSER"
  },
  "detail": {
    "employeeSelectServiceOperationResponse": {
      "employeeData": {
        "response": {
          "employeeDetails": {
            "employeeNumber": "948478",
            "name": "name",
            "address": "RTP NC",
            "phoneNumber": "0065",
            "date": "12/31/22",
            "amount": "$100.00",
            "comment": "22-09-27"
          }
        }
      }
    }
  }
}
```

Response headers

```
content-language: en-US
content-length: 348
content-type: application/json
date: Tue, 27 Sep 2022 20:05:57 GMT
x-firefox-spy: h2
x-powered-by: Servlet/4.0
```

Download

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



18. Click on **DELETE** `/cscvinc/employee/{employee}` URI path to display the request parameters.



19. Press the **Try it out** button.

20. Enter a value of **948488** and press the **Execute** button.

21. If the record exists, a message like the one below will appear.

Responses

Curl

```
curl -X 'DELETE' \
  'https://designer.ibm.com:9447/cscvinc/employee/948488' \
  -H 'accept: application/json'
```

Request URL

```
https://designer.ibm.com:9447/cscvinc/employee/948488
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Record with key 948488 was successfully deleted - CICS identity CICSUSER" }</pre> <p>Response headers</p> <pre>cache-control: no-cache="set-cookie,set-cookie2" content-language: en-US content-length: 92 content-type: application/json date: Mon, 26 Sep 2022 17:33:02 GMT expires: Thu, 01 Dec 1994 16:00:00 GMT x-firefox-spdy: h2 x-powered-by: Servlet/4.0</pre>

22. Otherwise, a message that the record was not found will appear.

Responses

Curl

```
curl -X 'DELETE' \
  'https://vg31.washington.ibm.com:9445/cscvinc/employee/948488' \
  -H 'accept: application/json'
```

Request URL

```
https://vg31.washington.ibm.com:9445/cscvinc/employee/948488
```

Server response

Code	Details
404	<p>Error: Not Found</p> <p>Response body</p> <pre>{   "message": "Record with key 948488 was not found - CICS Identity CICSUSER" }</pre>

23. Click on *POST* */cscvinc/employee* URI path to display the request body view of the URI path.

The screenshot shows the API Explorer interface for the **Employee** endpoint. The **POST** method and **/cscvinc/employee** path are selected. The **Parameters** tab is active, showing an **Authorization** header of type **string**. The **Request body** is marked as **required** and the media type is set to **application/json**. The **request body** section is expanded, displaying a JSON schema example:

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "string",
          "name": "string",
          "address": "string",
          "phoneNumber": "string",
          "date": "string",
          "amount": "string"
        }
      }
    }
  }
}
```

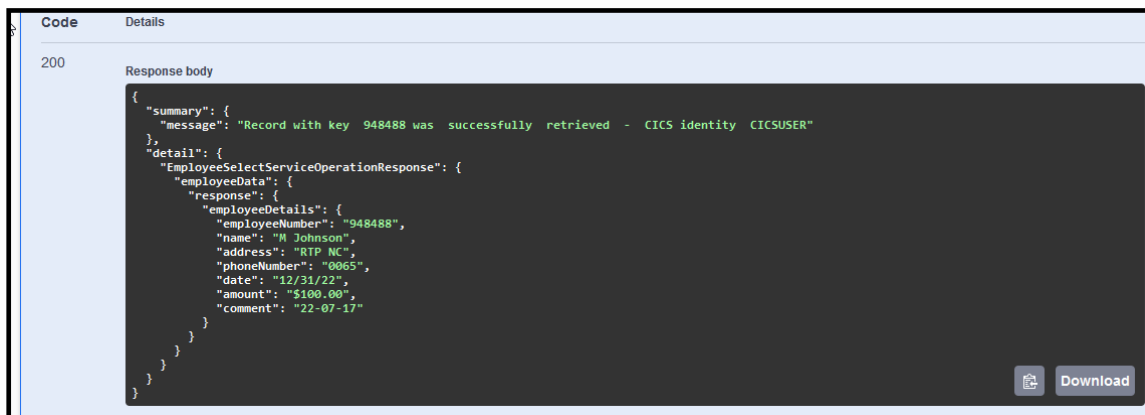
24. Next press the *Try it out* button to enable the entry of a request message body

The screenshot shows the same API Explorer interface, but the **Try it out** button has been pressed, resulting in a **Cancel** button appearing in the top right corner of the **Parameters** section. The **Request body** section remains expanded, showing the same JSON schema example as before.

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

```
{
  "EmployeeInsertServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "employeeNumber": "948488",
          "date": "12/31/22",
          "amount": "$100.00",
          "address": "RTP NC",
          "phoneNumber": "0065",
          "name": "M Johnson"
        }
      }
    }
  }
}
```

26. Use a Click on *GET* for URI path */employee/{employee}* to display the GET method parameters. Press the **Try it out** button Enter **948488** for the employee and press the **Execute** button to display this record.



The screenshot shows a REST client interface with a 'Code' tab selected. The status is 200. The response body is displayed in a dark-themed editor. The JSON response contains a summary message and detailed employee information.

```
{
  "summary": {
    "message": "Record with key 948488 was successfully retrieved - CICS identity CICSUSER"
  },
  "detail": {
    "EmployeeSelectServiceOperationResponse": {
      "employeeData": {
        "response": {
          "employeeDetails": {
            "employeeNumber": "948488",
            "name": "M Johnson",
            "address": "RTP NC",
            "phoneNumber": "0065",
            "date": "12/31/22",
            "amount": "$100.00",
            "comment": "22-07-17"
          }
        }
      }
    }
  }
}
```

At the bottom right of the editor, there are icons for 'Copy' and a 'Download' button.

Security was enabled in the original specification document, so you will be required to sign in with a RACF identity. Use **Fred** for the *Username* and **fred** for the *Password*. Please note that this identity can be changed unless all browser sessions are stopped.

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



28. Expand the *PUT* method and press the **Try it out** button.

29. Enter **948488** in the area beside *employee* and enter the JSON below in the request body area and press the **Execute** button.

```
{
  "EmployeeUpdateServiceOperation": {
    "employeeData": {
      "request": {
        "employeeDetails": {
          "status": "s",
          "name": "A Johnson",
          "address": "Apex NC",
          "phoneNumber": "0065",
          "date": "01/31/23",
          "amount": "500",
          "COMMENT": "updated"
        }
      }
    }
  }
}
```

Responses

Curl

```
curl -X 'PUT' \
  'https://w31.washington.ibm.com:9445/cscvinc/employee/948488' \
  -H 'accept: application/json' \
  -H 'Content-type: application/json' \
  -d '{
    "EmployeeUpdateServiceOperation": {
      "employeeData": {
        "request": {
          "employeeDetails": {
            "status": "s",
            "name": "A Johnson",
            "address": "Apex NC",
            "phoneNumber": "0065",
            "date": "01/31/23",
            "amount": "500",
            "COMMENT": "updated"
          }
        }
      }
    }
  }'
```

Request URL

https://w31.washington.ibm.com:9445/cscvinc/employee/948488

Server response

Code Details

200

Response body

```
{
  "message": "Record with key 948488 was successfully updated - CICS identity CICSUSER"
}
```

Response headers

```
content-language: en-US
content-length: 92
content-type: application/json
date: Thu, 29 Sep 2022 18:53:06 GMT
x-firefox-spy: h2
x-powered-by: Servlet/4.0
```

30. Use the *GET* method to confirm the updates have taken place.

31. Use the *DELETE* method to delete employee **948488** and confirm the record has been deleted.

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

## Additional information and samples

This section shows the contents of the VSAM file accessed by the CICS program. There is an introduction to performing problem determination while developing APIs.

*The table below list the contents of the VSAM data set.*

stat	numb	name	addrx	Phone	datex	amount	comment
Y	000100	S. D. BORMAN	SURREY, ENGLAND	32156778	26 11 81	\$0100.11	*****
Y	000102	J. T. CZAYKOWSI	WARWICK, ENGLAND	98356183	26 11 81	\$1111.11	*****
Y	000104	M. B. DOMBEY	LONDON, ENGLAND	12846293	26 11 81	\$0999.99	*****
Y	000106	A. I. HICKSON	CROYDON, ENGLAND	19485673	26 11 81	\$0087.71	*****
Y	000111	ALAN TULIP	SARATOGA, CALIFORNIA	46120753	01 02 74	\$0111.11	*****
Y	000762	SUSAN MALAIKA	SAN JOSE, CALIFORNIA	22312121	01 06 74	\$0000.00	*****
Y	000983	J. S. TILLING	WASHINGTON, DC	34512120	21 04 75	\$9999.99	*****
Y	001222	D.J.VOWLES	BOBLINGEN, GERMANY	70315551	10 04 73	\$3349.99	*****
Y	001781	TINA J YOUNG	SINDELFINGEN, GERMANY	70319990	21 06 77	\$0009.99	*****
Y	003210	B.A. WALKER	NICE, FRANCE	12345670	26 11 81	\$3349.99	*****
N	003214	PHIL CONWAY	SUNNYVALE, CAL.	34112120	00 06 73	\$0009.99	*****
N	003890	BRIAN HARDER	NICE FRANCE	00000000	28 05 74	\$0009.99	*****
N	004004	JANET FOUCHE	DUBLIN, IRELAND	71112121	02 11 73	\$1259.99	*****
N	004445	DR. P. JOHNSON	SOUTH BEND, S.DAK.	61212120	26 11 81	\$0009.99	*****
N	004878	ADRIAN JONES	SUNNYVALE, CALIF.	32212120	10 06 73	\$5399.99	*****
N	005005	A. E. DALTON	SAN FRANCISCO, CA.	00000001	01 08 73	\$0009.99	*****
N	005444	ROS READER	SARATOGA, CALIF.	67712120	20 10 74	\$0809.99	*****
N	005581	PETE ROBBINS	BOSTON, MASS.	41312120	11 04 74	\$0259.99	*****
Y	006016	SIR MICHAEL ROBERTS	NEW DELHI, INDIA	70331211	21 05 74	\$0009.88	*****
N	006670	IAN HALL	NEW YORK, N.Y.	21212120	31 01 75	\$3509.88	*****
Y	006968	J.A.L. STAINFORTH	WARWICK, ENGLAND	56713821	26 11 81	\$0009.88	*****
N	007007	ANDREW WHARMBY	STUTTGART, GERMANY	70311000	10 10 75	\$5009.88	*****
N	007248	M. J. AYRES	REDWOOD CITY, CALF.	33312121	11 10 75	\$0009.88	*****
Y	007779	MRS. A. STEWART	SAN JOSE, CALIF.	41512120	03 01 75	\$0009.88	*****
Y	009000	P. E. HAVERCAN	WATERLOO, ONTARIO	09876543	21 01 75	\$9000.00	*****
Y	100000	M. ADAMS	TORONTO, ONTARIO	03415121	26 11 81	\$0010.00	*****
Y	111111	C. BAKER	OTTAWA, ONTARIO	51212003	26 11 81	\$0011.00	*****
Y	200000	S. P. RUSSELL	GLASGOW, SCOTLAND	63738290	26 11 81	\$0020.00	*****
Y	222222	DR E. GRIFFITHS	FRANKFURT, GERMANY	20034151	26 11 81	\$0022.00	*****
Y	300000	V. J. HARRIS	NEW YORK, U.S.	64739801	26 11 81	\$0030.00	*****
Y	333333	J.D. HENRY	CARDIFF, WALES	78493020	26 11 81	\$0033.00	*****
Y	400000	C. HUNT	MILAN, ITALY	25363738	26 11 81	\$0040.00	*****
Y	444444	D. JACOBS	CALGARY, ALBERTA	77889820	26 11 81	\$0044.00	*****
Y	500000	P. KINGSLEY	MADRID, SPAIN	44454640	26 11 81	\$0000.00	*****
Y	555555	S.J. LAZENBY	KINGSTON, N.Y.	39944420	26 11 81	\$0005.00	*****
Y	600000	M.F. MASON	DUBLIN, IRELAND	12398780	26 11 81	\$0010.00	*****
Y	666666	R. F. WALLER	LA HULPE, BRUSSELS	42983840	26 11 81	\$0016.00	*****
Y	700000	M. BRANDON	DALLAS, TEXAS	57984320	26 11 81	\$0002.00	*****
Y	777777	L.A. FARMER	WILLIAMSBURG, VIRG.	91876131	26 11 81	\$0027.00	*****
Y	800000	P. LUPTON	WESTEND, LONDON	24233389	26 11 81	\$0030.00	*****
Y	888888	P. MUNDY	NORTHAMPTON, ENG.	23691639	26 11 81	\$0038.00	*****
Y	900000	D.S. RENSHAW	TAMPA, FLA.	35668120	26 11 81	\$0040.00	*****
Y	999999	ANJI STEVENS	RALEIGH, N.Y.	84591639	26 11 81	\$0049.00	*****

## 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 500 with no EIB response code or identity. A review of the trace shows that the CICS transaction abended with an abend code of MIJO.

The screenshot displays the 'designer.ibm.com:9447/logs/trace.log' file in a web browser. The log contains a series of entries for a CICS transaction. A red box highlights the following key entries:

- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.QueueingInputStream 3 copyFromBuffer Entry, offset=0, length=40
- [7/15/22 18:42:02:060 UTC] 00000000 id=52dd7d68 om.ibm.zosconnect.cics.internal.conn.isc.QueueingInputStream < getBytes Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=52dd7d68 om.ibm.zosconnect.cics.internal.conn.isc.QueueingInputStream < read Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.headers.IS44Header 3 40 bytes read, 40 bytes requested
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.headers.IS44Header 3 Dump: IS44 Inbound
- [7/15/22 18:42:02:060 UTC] 00000000 id=76714563 com.ibm.zosconnect.cics.internal.conn.isc.headers.IS44Header < readReply Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=29d48583 com.ibm.zosconnect.cics.Channel > setCCSID Entry
- [7/15/22 18:42:02:060 UTC] 00000000 id=29d48583 com.ibm.zosconnect.cics.Channel < setCCSID Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest < readChannelData Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest > decodeAbendCode Entry
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.ISCECIRequest 3 decodeAbendCode
- [7/15/22 18:42:02:060 UTC] 00000000 id=eab3c3c4 com.ibm.zosconnect.cics.CicsIpicConnection > getID Entry
- [7/15/22 18:42:02:060 UTC] 00000000 id=eab3c3c4 com.ibm.zosconnect.cics.CicsIpicConnection < getID Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.ISCECIRequest E BAQR0657E: Transaction abend MIJO occurred in CICS while using CICS connection cicsConn and service program Cscvnc.
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest < decodeAbendCode Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.ISCECIRequest 3 ISC Return code set to 500
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.ISCECIRequest 3 Decoded IS43: eibrc=0, abend=MIJO, mapped to isrc=500
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest < parseDPLReply Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest > setTimeSorReceivedAndDriveInterceptors Entry
- [7/15/22 18:42:02:060 UTC] 00000000 id=750bdef5 om.ibm.zosconnect.zosasset.cics.internal.hockResponseDataExit > setTimeSorReceived Entry
- [7/15/22 18:42:02:060 UTC] 00000000 id=750bdef5 om.ibm.zosconnect.zosasset.cics.internal.hockResponseDataExit > setTimeSorReceived Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=750bdef5 om.ibm.zosconnect.zosasset.cics.internal.hockResponseDataExit > driveServiceProviderInterceptors Entry
- [7/15/22 18:42:02:060 UTC] 00000000 id=750bdef5 om.ibm.zosconnect.zosasset.cics.internal.hockResponseDataExit < driveServiceProviderInterceptors Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest < setTimeSorReceivedAndDriveInterceptors Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=324cef4e com.ibm.zosconnect.cics.internal.conn.ISCECIRequest < readReply Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.Session 3 Current Session state is RECEIVING
- [7/15/22 18:42:02:060 UTC] 00000000 id=00000000 com.ibm.zosconnect.cics.internal.conn.isc.Session 3 Session state updated from RECEIVING to WAITING
- [7/15/22 18:42:02:060 UTC] 00000000 id=2087cc8d com.ibm.zosconnect.cics.internal.conn.isc.Session < readReply Exit
- [7/15/22 18:42:02:060 UTC] 00000000 id=cab3e99c com.ibm.zosconnect.cics.internal.conn.isc.Conversation < readReply Exit

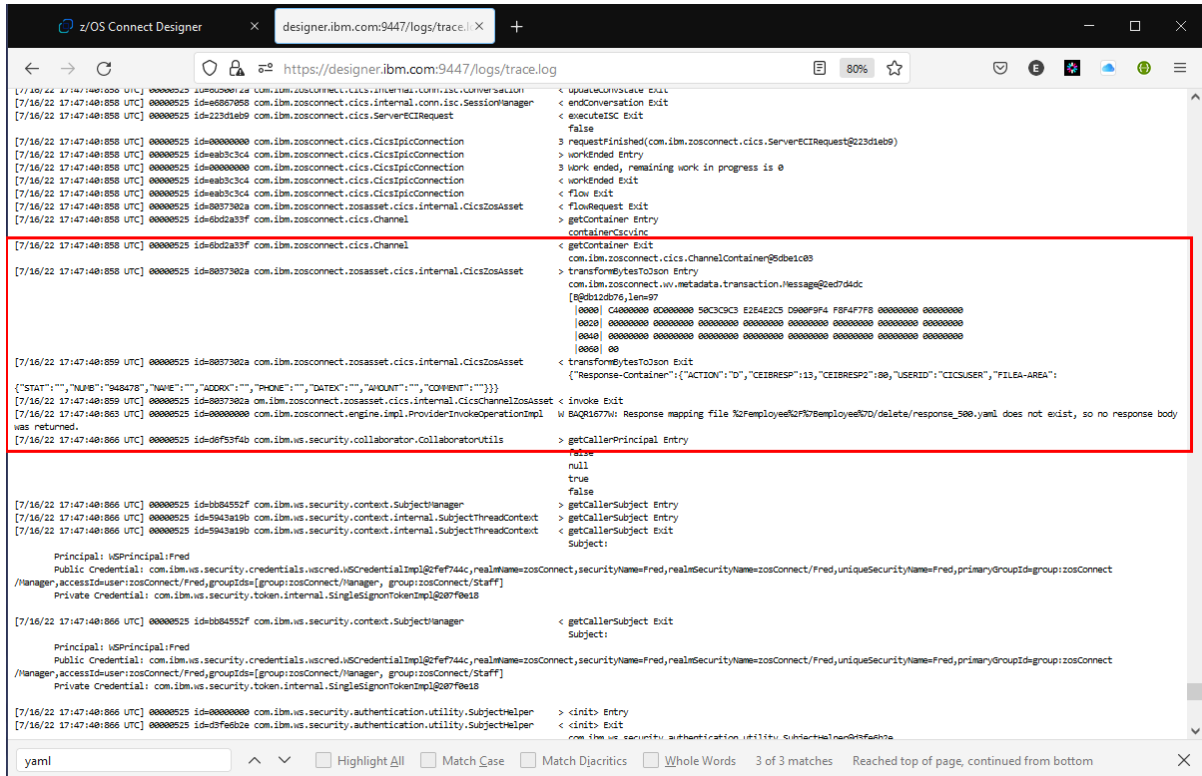
The search bar at the bottom shows the term 'mijo' with 4 of 5 matches.

2. Invoking a method returned an HTTP 500 with no other information, for example using the curl as shown below.

```
c:\z\openApi3>curl -X DELETE -w " - HTTP CODE %{http_code}" --header "Content-Type: application/json" --insecure --user Fred:fredpwd https://designer.washington.ibm.com:9447/employee/948478 - HTTP CODE 500
```

d

The trace showed a response came back from CICS with non-zero EIB response codes. But the mapping for this situation was not present, i.e., the *response\_500.yaml* did not exist.



The point of this section is to use the *trace.log* and *messages.out* file when developing an API identify issues with the API and the interactions with the CICS region.