

IBM z/OS Connect (OpenAPI 2.0)

Developing MVS Batch API Requesters Applications



Lab Version Date: February 2, 2023

Table of Contents

Overview	4
<i>MVS Batch RESTful API Requester (part 1)</i>	5
<i>Generate the API requester artifacts from a Swagger document.....</i>	5
<i>Review the application programs</i>	12
<i>Compile and link-edit the application programs</i>	15
<i>Test the API requester application programs</i>	17
<i>MVS Batch RESTful API Requester (part 2)</i>	20
<i>Generate the API requester artifacts from a Swagger document.....</i>	20
<i>Review the application programs</i>	27
<i>Compile and link-edit the application programs</i>	29
<i>Test the API requester application programs</i>	30
Summary	32

General Exercise Information and Guidelines

- ✓ The Windows artifacts for this exercise are in directory *c:\z\apiRequester*. Open a DOS command prompt Window and go to this directory using a change directory command, e.g., *cd \z\apiRequester*
- ✓ Viewing or changing the contents of a file can easily be done by opening the Windows Explorer and going to directory *c:\z\apiRequester*. On this folder's display you can select a file and right mouse button click and select the *Edit* option to open a file.
- ✓ This exercise requires using TSO user *USER1* and the RACF password for this user is *USER1*.
- ✓ This exercise primarily uses data sets *USER1.ZCEE.CNTL* and *USER.ZCEE.SOURCE*.
- ✓ 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.
- ✓ Text in **bold** and highlighted in **yellow** in this document should be available for copying and pasting in a file named *OpenAPI2 developing API Requesters CopyPaste* file on the desktop.
- ✓ Any time you have any questions about the use of IBM z/OS Explorer, 3270 screens, features or tools do not hesitate to ask the instructor for assistance.
- ✓ For information regarding the use of the Personal Communication 3270 emulator, see the *Personal Communications Tips* PDF in the exercise folder.

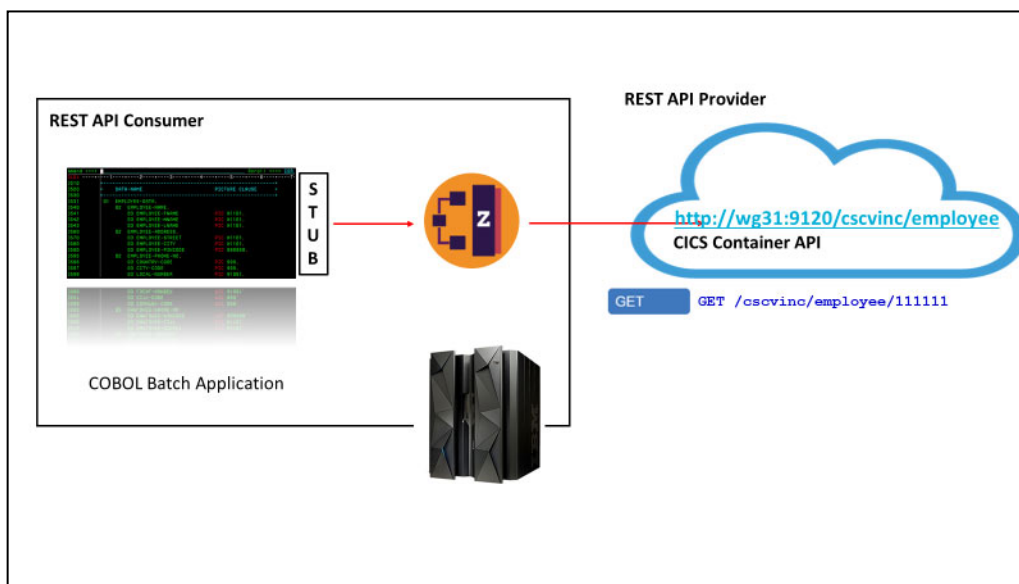
Important: The REST API provider applications used in this exercise access a CICS applications and MQ. These APIs were developed using the instructions provided in other exercises in this workshop. The REST API providers could have just as easily have been anywhere in the cloud on any remote system as long as the Swagger document for the REST API was available.

Overview

These exercises demonstrate the steps required to enable COBOL programs to invoke RESTful APIs. There are two samples provided to do this. One sample uses RESTful APIs to access a CICS program and the other uses RESTful APIs to interact with an MQ queue. Note that even though these APIs are hosted in z/OS Connect server these same steps could be followed to access APIs hosted anywhere in the cloud.

The process starts with the Swagger document that describes the API which the COBOL program will invoke. The Swagger document is used by the z/OS Connect build toolkit to (1) generate the COBOL copy books included in the COBOL program to invoke the API (2) as well as an API requester archive file (ARA) which is deployed to a z/OS Connect server.

The diagram shows this process at a high level. The copy books generated by the z/OS Connect build toolkit will be integrated into a COBOL program to provide (1) the COBOL representation of the request and response messages and to identify (2) the HTTP method to be used in the RESTful request. The API requester archive file will be deployed to a z/OS Connect server and used to convert (1) the COBOL request and response messages to JSON messages and to invoke (2) the outbound RESTful request and wait for the response.



In this exercise the COBOL program will be compiled and executed in a batch environment. The only differences between batch and CICS and IMS is the configuration required for connecting to the z/OS Connect server. Consult the z/OS Connect Knowledge Center for information how this is done for CICS and IMS.

If you have completed either the developing APIs exercise for MVS Batch, DB2 or MQ you can start with section *MVS Batch RESTful API Requester* on page 5.

MVS Batch RESTful API Requester to a VSAM API Provider

In this section an API requester will be developed and tested where the target API accesses a VSAM data set. The batch programs will invoke the API to insert records (POST), retrieve records (GET), update record (PUT) and delete records (DELETE) in a VSAM data set.

Generate the API requester artifacts from a Swagger document

The first step is to use the API's Swagger document to generate the artifacts required for the COBOL API client application and the API requester archive file. The Swagger document was obtained from the server where the API is hosted. All the files mention in this section are in directory `c:/z/apiRequester/cscvinc` on your workstation.

The z/OS Connect build toolkit (ZCONBT) is a Java application shipped with z/OS Connect as a zip file and can be installed on a workstation or in OMVS. Anywhere a Java runtime is available. In this exercise the build toolkit will on Windows.

1. Begin by reviewing the API's Swagger document for the VSAM API provider. This document identifies the types of HTTP methods supported as well as the path of each HTTP method. It also the contents each method's request JSON message and the response JSON message field contents and field types. This is the blueprint for invoking the API.

```
{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "cscvinc"
  },
  "basePath": "/cscvinc",
  "schemes": [
    "https",
    "http"
  ],
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/employee": {
      "post": {
        "operationId": "postCscvincService",
        "parameters": [
          {
            "in": "body",
            "name": "postCscvincService_request",
            "description": "request body",
            "required": true,
            "schema": {
              "$ref": "#/definitions/postCscvincService_request"
            }
          }
        ]
      }
    }
  }
}
```

Tech-Tip: Instructions for installing the z/OS Connect build toolkit can be found at URL https://www.ibm.com/support/knowledgecenter/en/SS4SVW_3.0.0/com.ibm.zosconnect.doc/installing/bt_install.html

2. Next review the input parameters that will be passed to the z/OS Connect build toolkit in a properties file. This file is *cscvinc.properties* see its below:

```
apiDescriptionFile=./swagger.json 1
dataStructuresLocation=./syslib 2
apiInfoFileLocation=./syslib 3
logFileDirectory=./logs 4
language=COBOL 5
connectionRef=cscvincAPI 6
requesterPrefix=csc 7
```

1. Identifies the Swagger document that describes the API
2. Provides the directory where the generated COBOL copybooks representing the request and response messages will be written.
3. Provides the directory where the generated COBOL copybook with invocation details of the API will be written.
4. Provides the directory where a log file will be written.
5. Identifies the target language.
6. Identifies the corresponding *zosconnect_endpointConnection* element define in the z/OS Connect server's server.xml file.
7. Provides a 3-character prefix to be used for all generated copybooks

3. View file *cscvinc.bat* which contains this one line.

```
c:\z\zconbt\bin\zconbt.bat -p=./cscvinc.properties -f=./cscvinc.ara
```

This is batch command file that when executed invokes the z/OS Connect build toolkit (*zconbt*). The *-p* switch identifies the input properties file and the *-f* switch identifies the name of the generate API requester archive file. The API requester archive (ARA) file will be deployed and made available to the z/OS Connect server in directory *.../resources/zosconnect/apiRequesters*.

4. On the workstation desktop, locate the *Command Prompt* icon and double click on it to open a DOS session that will allow commands to be entered.

5. Use the change directory command (*cd*) to change to directory *C:\z\api\Requester\cscvinc*, e.g.

```
cd c:\z\apiRequester\cscvinc
```

6. Enter command **cscvinc** in the DOS command prompt window and you should see output like the below:

```

BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.2 (20200408-1120).
BAQB0008I: Creating API requester archive from configuration file ./cscvinc.properties.
BAQB0040I: The generated API requester is automatically named cscvincapi_1.0.0 based on the title
cscvincapi and version 1.0.0 of the API to be called.
BAQB0015I: Start processing operation (operationId: getCscvincSelectService, relativePath:
/employee/{employee}, method: GET).
DFHPI9586W A reserved word "address" has been detected in the input document, it has been changed
to "Xaddress".
DFHPI9586W A reserved word "date" has been detected in the input document, it has been changed to
"Xdate".
BAQB0016I: Successfully processed operation (operationId: getCscvincSelectService, relativePath:
/employee/{employee}, method: GET).
BAQB0015I: Start processing operation (operationId: putCscvincUpdateService, relativePath:
/employee/{employee}, method: PUT).
DFHPI9586W A reserved word "address" has been detected in the input document, it has been changed
to "Xaddress".
DFHPI9586W A reserved word "date" has been detected in the input document, it has been changed to
"Xdate".
BAQB0016I: Successfully processed operation (operationId: putCscvincUpdateService, relativePath:
/employee/{employee}, method: PUT).
BAQB0015I: Start processing operation (operationId: postCscvincInsertService, relativePath:
/employee/{employee}, method: POST).
DFHPI9586W A reserved word "address" has been detected in the input document, it has been changed
to "Xaddress".
BAQB0016I: Successfully processed operation (operationId: postCscvincInsertService, relativePath:
/employee/{employee}, method: POST).
BAQB0015I: Start processing operation (operationId: deleteCscvincDeleteService, relativePath:
/employee/{employee}, method: DELETE).
BAQB0016I: Successfully processed operation (operationId: deleteCscvincDeleteService, relativePath:
/employee/{employee}, method: DELETE).

Total 4 operation(s) (success: 4, ignored: 0) defined in api description file: ./swagger.json
----- Successfully processed operation(s) -----
operationId: postCscvincInsertService, basePath: /cscvinc, relativePath: /employee, method: POST
- request data structure   : CSC00Q01
- response data structure  : CSC00P01
- api info file            : CSC00I01

operationId: getCscvincSelectService, basePath: /cscvinc, relativePath: /employee/{employee},
method: GET
- request data structure   : CSC01Q01
- response data structure  : CSC01P01
- api info file            : CSC01I01

operationId: putCscvincUpdateService, basePath: /cscvinc, relativePath: /employee/{employee},
method: PUT
- request data structure   : CSC02Q01
- response data structure  : CSC02P01
- api info file            : CSC02I01

operationId: deleteCscvincDeleteService, basePath: /cscvinc, relativePath: /employee/{employee},
method: DELETE
- request data structure   : CSC03Q01
- response data structure  : CSC03P01
- api info file            : CSC03I01

```

Note that each HTTP method defined in the Swagger document will cause the generation of up to three copy books. One for the request message(Q01), one for the response message(P01) and one for the API details(I01). The copy book names are based on the *requesterPrefix* property (e.g. *csc*) and use an ascending sequence number sequence to differentiate between methods. Also note that there may be fewer than three copy books generated. For example, if there is no response message or no request message for a specific method a copy book may not be generated.

Important Note: The COBOL programs used in this exercise include these generated copy books with the names as generated above, e.g. the GET COBOL program includes CSC01 prefixed copy books. Occasionally (probably after applying service) the sequence of the methods will change causing the corresponding sequence number of the copy book to also change. Review the sequence of the processing of the methods and change the COBOL applications so they include the correct copy book. Otherwise the compilation of the COBOL program may fail or unexpected results may be observed.

7. Now explore a sample snippet of the copy book for the request message of a PUT method. Use *EditPad Lite* to open file `c:\z\apiRequester\cscvinc\syslib\CSC02Q01`. Scroll down until you see something like the code below.

```

06 ReqPathParameters.
  09 employee-length          PIC S9999 COMP-5 SYNC.
  09 employee                 PIC X(6) .
06 ReqBody.
  09 cscvincUpdateServiceOp-num PIC S9(9) COMP-5 SYNC.
  09 cscvincUpdateServiceOperatio.
    12 cscvincContainer.
      15 request2-num          PIC S9(9) COMP-5 SYNC.
      15 request.
        18 filea2-num          PIC S9(9) COMP-5 SYNC.
        18 filea.
          21 name-num          PIC S9(9) COMP-5 SYNC.
          21 name.
            24 name2-length     PIC S9999 COMP-5 SYNC.
            24 name2           PIC X(20) .
          21 Xaddress-num       PIC S9(9) COMP-5 SYNC.
          21 Xaddress.
            24 Xaddress2-length PIC S9999 COMP-5 SYNC.
            24 Xaddress2       PIC X(20) .
          21 phoneNumber-num    PIC S9(9) COMP-5 SYNC.
          21 phoneNumber.
            24 phoneNumber2-length PIC S9999 COMP-5 SYNC.
            24 phoneNumber2     PIC X(8) .
          21 Xdate-num          PIC S9(9) COMP-5 SYNC.
          21 Xdate.
            24 Xdate2-length     PIC S9999 COMP-5 SYNC.
            24 Xdate2           PIC X(8) .
          21 amount-num         PIC S9(9) COMP-5 SYNC.
          21 amount.
            24 amount2-length    PIC S9999 COMP-5 SYNC.
            24 amount2          PIC X(8)

```


Note that each variable has an associated length (e.g. *name2-length*). The length of a variable is required since there is no delimiter between the variables in storage. The runtime needs to know the size of a variable at execution time so the request and response messages can be properly converted to and from JSON name/value pairs.

For this API, the Swagger document included an additional variable for the number of occurrences of a variable, (e.g. *name-num*). In this exercise we will set the number of each variable to one.

Tech-Tip: This additional variable was included for this API because the target program is a CICS container enabled program and a CICS channel can have multiple occurrences of the same container. The z/OS Connect API requester support needs to know number of each variable is being sent (and returned). This behavior is not unique to CICS.

The corresponding response message (*c:\z\apiRequester\cscvinc\syslib\CSC02P01*) will have a similar layout.

8. The corresponding API information copy book (*c:\z\apiRequester\cscvinc\syslib\CSC02I01*) has contents providing the API name and the path to be used for the method:

```
03 BAQ-APINAME                PIC X(255)
   VALUE 'cscvinc_1.0.0'.
03 BAQ-APINAME-LEN            PIC S9(9) COMP-5 SYNC
   VALUE 13.
03 BAQ-APIPATH                PIC X(255)
   VALUE '%2Fcscvinc%2Femployee%2F%7Bemployee%7D'.
03 BAQ-APIPATH-LEN            PIC S9(9) COMP-5 SYNC
   VALUE 38.
03 BAQ-APIMETHOD              PIC X(255)
   VALUE 'PUT'.
03 BAQ-APIMETHOD-LEN          PIC S9(9) COMP-5 SYNC
   VALUE 3.
```

Deploy the API Requester Archive file to the z/OS Connect server

Next make the API requester archive file available to the z/OS Connect server.

The API requester archive (ARA) file needs to be deployed to the API requester directory. In this case the target directory is */var/ats/zosconnect/servers/zceapir/resources/zosconnect/apiRequesters*.

1. Open a DOS command session and change to directory *c:\z\apiRequester\cscvinc*, e.g **cd c:\z\apiRequester\cscvinc**
2. Use the z/OS Connect RESTful administrative interface to deploy the ARA files by using the cURL command embedded in command file *deploy.cmd*. Invoke this command file to deploy the cscvinc API archive file.

```
c:\z\apiRequester\cscvinc>curl -X POST --user Fred:fredpwd --data-binary @cscvinc
c.ara --header "Content-Type: application/zip" --insecure https://wg31.washingt
on.ibm.com:9483/zosConnect/apiRequesters
{"name":"cscvinc_1.0.0","version":"1.0.0","description":"","status":"Started","api
RequesterUrl":"https://wg31.washington.ibm.com:9483/zosConnect/apiRequeste
rs/cscvinc_1.0.0","connection":"cscvincAPI"}
```

Tech-Tip: If a REST client tool like cURL or Postman was not available then ARA file could have been deployed using FTP to upload the file in binary mode to the *apiRequesters* directory.

Tech-Tip: If an ARA needs to be redeployed, the cURL command with a PUT method can be used to stop the API requester

curl -X PUT --user Fred:fredpwd --insecure

https://wg31.washington.ibm.com:9483/zosConnect/apiRequesters/cscvinc_1.0.0?status=stopped

And the cURL command with a DELETE method can be used to delete the API requester.

curl -X DELETE --user Fred:fredpwd --insecure

https://wg31.washington.ibm.com:9483/zosConnect/apiRequesters/cscvinc_1.0.0

otherwise the redeployment of the ARA file will fail.

3. The z/OS Connect server has the configuration below already included.

```
<server description="API Requester">
  <!-- Enable features -->
  <featureManager>
    <feature>zosconnect:apiRequester-1.0</feature>
  </featureManager>

  <zosconnect_endpointConnection id="cscvincAPI" 1
    host="http://wg31.washington.ibm.com"
    port="9120"
    basicAuthRef="myBasicAuth"
    connectionTimeout="10s"
    receiveTimeout="20s" />

  <zosconnect_authData id="myBasicAuth" 2
    user="Fred"
    password="fredpwd" />

</server>
```

1. Identifies the system which hosts the target API as well as any security information and time out parameters. Note that the ID of this element matches the value *connectionRef* property used when the ARA was generated by the z/OS Connect build toolkit.
2. The target server uses basic authentication, so we are simply provided a user identity and password. The password can be encrypted in the server.xml file or TLS used for security.

Tech-Tip: The *endpointConnection* element specifies the host and port of the target API provider. This z/OS Connect server performs the conversion of the COBOL data structure to JSON and then acts as a JSON REST client to the target API provider. Which for this example, is the same z/OS Connect server. The REST client loops back to itself when invoking the API. This would rarely happen in a real-world environment. This is why this host and port is also used by the API requester client in its environment variables.

Move the COBOL copy books to an MVS data set

Next make the generated COBOL copy books available for including into the COBOL program.

The generated COBOL copy books need to be available in an MVS data set which is included in the compiler SYSLIB concatenation sequence. In the data set is *USER1.ZCEE.SOURCE*. Use a DOS command prompt session and the commands below to move the copy books file to this data set. Authenticate as user USER1 (password USER1).

1. Open a DOS command prompt and use the change directory command to go to directory C:\z\apiRequester\cscvinc\syslib, e.g. **cd \z\apiRequester\cscvinc\syslib**
2. Start a file transfer session with the WG31 host using the *ftp* command, e.g. **ftp wg31**
3. Logon as USER1 and then use the *cd* command to change to directory to data set *USER1.ZCEE.SOURCE*, e.g. **cd zcee.source**
4. Toggle prompting off by entering command **prompt**
5. Perform multiple put requests by using the multiple put command, **mput csc***
6. When the last transfer has completed enter the **quit** command.

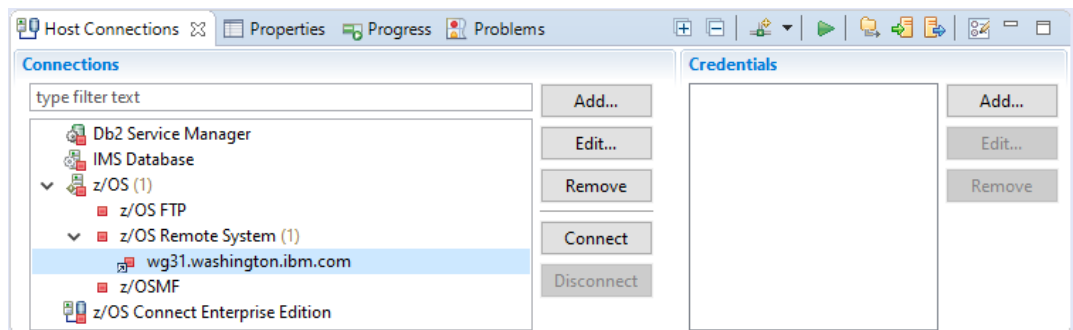
```
c:\z\apiRequester\cscvinc>cd syslib
c:\z\apiRequester\cscvinc\syslib>ftp wg31.washington.ibm.com
Connected to wg31.washington.ibm.com.
220-FTP 16:26:23 on 2018-02-15.
220 Connection will close if idle for more than 200 minutes.
User (wg31.washington.ibm.com:(none)): user1
331 Send password please. user1
Password:
230 USER1 is logged on. Working directory is "USER1.".
ftp> cd zcee.source
250 The working directory "USER1.ZCEE.SOURCE" is a partitioned data set
ftp> prompt
Interactive mode Off .
ftp> mput CSC*
200 Port request OK.
125 Storing data set USER1.ZCEE.SOURCE(CSC00I01)
250 Transfer completed successfully.
ftp: 533 bytes sent in 0.32Seconds 1.66Kbytes/sec.
200 Port request OK.
sent in 0.10Seconds 110.89Kbytes/sec.
200 Port request OK.
.....
125 Storing data set USER1.ZCEE.SOURCE(CSC03P01)
250 Transfer completed successfully.
ftp: 7393 bytes sent in 0.32Seconds 22.89Kbytes/sec.
200 Port request OK.
125 Storing data set USER1.ZCEE.SOURCE(CSC03Q01)
250 Transfer completed successfully.
ftp: 1132 bytes sent in 0.40Seconds 2.80Kbytes/sec.
ftp> quit
```

Review the application programs

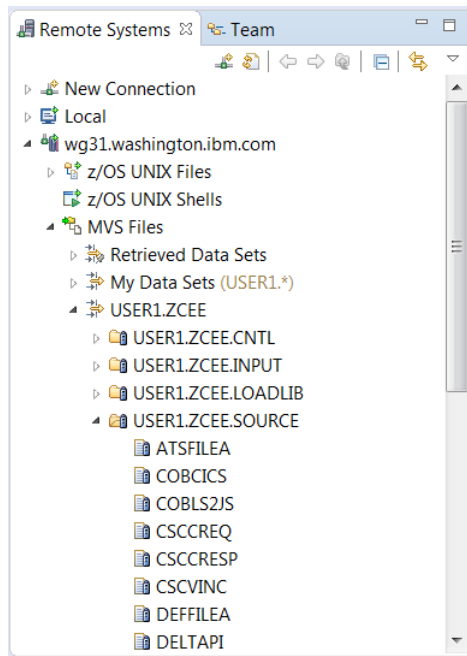
Now that the copy books have been generated and are available in a data set the next step is compile and link edit the application programs. The source for the application programs is in `USER1.ZCEE.SOURCE`. Member `DELTAPI` invokes the `DELETE` method, member `GETAPI` invokes the `GET` method, `POSTAPI` invokes the `POST` method and `PUTAPI` invokes the `PUT` method.

Tech-Tip: Data set `USER1.ZCEE.SOURCE` can be accessed either by using the IBM z/OS Explorer or by using the Personal Communication icon on the desktop and logging on to TSO and use ISPF option 3.4 to access this data set. If you have any questions about either method, ask the instructor.

1. Back in the IBM z/OS Explorer session switch to the *Remote System Explorer* perspective and select `wg31.washington.ibm.com` under *z/OS Remote System* and use the **Connect** button to connect to the remote system explorer daemon running on z/OS.



2. Expand the `USER1.ZCEE` filter and select data set `USER1.ZCEE.SOURCE` in the *Remote System view* and double click to display a list of its members.



Tech-Tip: Data set USER1.ZCEE.SOURCE can be accessed either by using the IBM z/OS Explorer or by using the Personal Communication icon on the desktop and logging on to TSO and use ISPF option 3.4 to access this data set. If you have any questions about either method, ask the instructor.

3. Begin by reviewing the application programs GETAPI. Open member *GETAPI* in *USER1.ZCEE.SOURCE* and review its contents by selecting the member and right- mouse button clicking and selecting the *Open* option. Scroll down until you see these lines of code.

These lines of code copy into the source the required z/OS Connect copy book (BAQRINFO) and the copy books generated by the z/OS Connect build tool kit. If when you generated the copy book earlier in the exercise and the sequence number used for the GET method was not 00, e.g. the CSC00 prefix, then you need to change the COBOL source to match sequence generated when you invoke the ZCONBT tool.

```
* Copy API Requester required copybook
COPY BAQRINFO.

* Request and Response
01 GET-REQUEST.
   COPY CSC01Q01.
01 GET-RESPONSE.
   COPY CSC01P01.
* Structure with the API information
01 GET-INFO-OPER1.
   COPY CSC01I01.
```

4. Scroll down further until you find this code which provides the contents of the request message.

```
*-----*
* Set up the data for the API Requester call *
*-----*

  MOVE employee of PARM-DATA TO employee IN GET-REQUEST.
  MOVE LENGTH of employee in GET-REQUEST to
    employee-length IN GET-REQUEST.

*-----*
* Initialize API Requester PTRs & LENS *
*-----*

* Use pointer and length to specify the location of
* request and response segment.
* This procedure is general and necessary.
  SET BAQ-REQUEST-PTR TO ADDRESS OF GET-REQUEST.
  MOVE LENGTH OF GET-REQUEST TO BAQ-REQUEST-LEN.
  SET BAQ-RESPONSE-PTR TO ADDRESS OF GET-RESPONSE.
  MOVE LENGTH OF PUT-RESPONSE TO BAQ-RESPONSE-LEN.
```

The *GET* method only requires one variable and it is provided as a path parameter, e.g. /cscvinc/employee/{numb}.

5. Scroll down further and you will the call to the z/OS Connect API requester stub module.

```

*-----*
* Call the communication stub                                     *
*-----*
* Call the subsystem-supplied stub code to send
* API request to zCEE
    CALL COMM-STUB-PGM-NAME USING
        BY REFERENCE    GET-INFO-OPER1
        BY REFERENCE    BAQ-REQUEST-INFO
        BY REFERENCE    BAQ-REQUEST-PTR
        BY REFERENCE    BAQ-REQUEST-LEN
        BY REFERENCE    BAQ-RESPONSE-INFO
        BY REFERENCE    BAQ-RESPONSE-PTR
        BY REFERENCE    BAQ-RESPONSE-LEN.
* The BAQ-RETURN-CODE field in 'BAQRINFO' indicates whether this
* API call is successful.

```

Note that the parameters are being passed by reference. This means that the z/OS Connect API requester stub will have direct access to the program storage for both accessing data and making changes (i.e. the response message).

6. Scroll down further and you will see the code where the application displays the results in the response message.

```

IF BAQ-SUCCESS THEN
  DISPLAY "EmployeeNumber: " employeeNumber2 of GET-RESPONSE
  DISPLAY "EmployeeName:   " employeeName2 of GET-RESPONSE
  DISPLAY "Address:        " Xaddress2 of GET-RESPONSE
  DISPLAY "Phone:          " phoneNumber2 of GET-RESPONSE
  DISPLAY "Date:           " Xdate2 of GET-RESPONSE
  DISPLAY "Amount:         " amount2 of GET-RESPONSE
  MOVE CEIBRESP of GET-RESPONSE to EIBRESP
  MOVE CEIBRESP2 of GET-RESPONSE to EIBRESP2
  DISPLAY "EIBRESP:        " EIBRESP
  DISPLAY "EIBRESP2:       " EIBRESP2
  DISPLAY "USERID:         " USERID2

```

Close member *GETAPI* and open member *POSTAPI*. The code for this program is very similar to the code for *GETAPI* with the exception that all variables are provided in a JSON request message. Scroll down and find the code below.

```

*-----*
* Set up the data for the API Requester call                                *
*-----*
      MOVE 1 to cscvincInsertServiceOp-num in POST-REQUEST
      REQUEST2-num in POST-REQUEST
      filea2-num in POST-REQUEST
      employeeNumber-num in POST-REQUEST
      employeeName-num in POST-REQUEST
      Xaddress-num in POST-REQUEST
      phoneNumber-num in POST-REQUEST
      startDate-num in POST-REQUEST
      amount-num in POST-REQUEST.

      MOVE employeeNumber of PARM-DATA TO employeeNumber2
      IN POST-REQUEST.
      MOVE LENGTH of employeeNumber2 in POST-REQUEST to
      employeeNumber2-length IN POST-REQUEST.

      MOVE "John" TO employeeName2 IN POST-REQUEST.
      MOVE LENGTH of employeeName2 in POST-REQUEST to
      employeeName2-length IN POST-REQUEST.

```

In this case the number of occurrences of a property

must be provided and the property value moved to the request copy book.

7. Explore members *DELTAPI* and *PUTAPI* and their corresponding copy books to see the code which is common (variable housekeeping, calls to the z/OS Connect stub, etc.) regardless of the method being invoked.

Compile and link-edit the application programs

The applications programs now are ready to be compiled and link edited.

1. Submit the job in member **CSCVINC** in data set *USER1.ZCEE.CNTL*.

```
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,PARM.COBOL='NODYNAM
//COBOL.SYSIN DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(POSTAPI)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
// DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(POSTAPI)
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,PARM.COBOL='NODYNAM
//COBOL.SYSIN DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(GETAPI)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
// DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(GETAPI)
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,PARM.COBOL='NODYNAM'
//COBOL.SYSIN DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(PUTAPI)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
// DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(PUTAPI)
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,PARM.COBOL='NODYNAM'
//COBOL.SYSIN DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(DELTAPI)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
// DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(DELTAPI)
```

Note that the CALL statement in each of this program uses a variable for the program name z/OS Connect stub program (BAQCSTUB). This means that this program will be dynamically linked at execution. The load module library containing BAQSCTUB (ZCEE30.SBAQLIB) must be available in the STEPLIB concatenation sequence when the program is executed.

This job should complete with a zero-return code.

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

Test the API requester application programs

The JCL to execute these programs can be found in USER1.ZCEE.CNTL, the member names in this data set match the program names. The members are DELTAPI, GETAPI, POSTAPI and PUTAPI.

1. Submit the job in member **GETAPI** in *USER1.ZCEE.CNTL*.

```
//GETAPI EXEC PGM=GETAPI,PARM='111111'
//STEPLIB DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
// DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//SYSPRINT DD SYSOUT=*
//CEEOPPTS DD *
    POSIX(ON),
    ENVAR("BAQURI=wg3l.washington.ibm.com",
    "BAQPORT=9120")
```

2. The job should complete with a condition code of 200 and have the following output for SYSPRINT.

```
EmployeeNumber: 111111
EmployeeName: C. BAKER
Address: OTTAWA, ONTARIO
Phone: 51212003
Date: 26 11 81
Amount: $0011.00
EIBRESP: 00000000
EIBRESP2: 00000000
USERID: CICSUSER
HTTP CODE: 000000200
```

Tech-Tip: An HTTP code of 200 indicates success. For an explanation of HTTP codes see URL https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

3. Change the PARM='111111' to PARM='000000' and resubmit. This time should receive a return code of 404 and the output should look like this:

```
Error code: 0000000404
Error
msg:{"cscvincSelectServiceOperationResponse":{"Container1":{"RESPONSE_CONTAINER":{"CEIBRESP2":80,"fileArea":{"date":"","employeeName":"","amount":"","address":"","phone":"","employeeNumber":"000000"},"userIdentity":"CICSUSER","CEIBRESP":13}}}}
```

The EIBRESP and EIBRESP2 values are from a CICS program and the response codes received when an EXEC CICS READ fails with a *Not Found* condition.

The target API was developed using z/OS Connect and had a response code mapping that checked the values of EIBRESP and EIBRESP2 and if this error occurred, the API was to return an HTTP 404 return code.

Edit Response 404

Response code: **404 - Not Found** Description: **Not Found**

Define rules that indicate whether to use this response code and apply its response mapping, if defined.

Rule	Path	Operator	Value	Logic	Buttons
Rule 1	:/Container1/RESPONSE_CONTAINER/CEIBRESP	=	13	AND	Up, Down, X
Rule 2	:/Container1/RESPONSE_CONTAINER/CEIBRESP2	=	80		Up, Down, X

Add Rule

Summary
Rule 1 AND Rule 2

OK **Cancel**

4. Submit the job in member **POSTAPI** in *USER1.ZCEE.CNTL*.

5. The job should complete with a condition code of 200 and have the following output for SYSPRINT.

```
EIBRESP:      00000000
EIBRESP2:     00000000
USERID:       CICSUSER
HTTP CODE:    0000000200
```

This job added a record for employee 323232.

6. Edit member **GETAPI** and change the *PARM* value to **323232** and submit the job.

```
EmployeeNumber: 323232
EmployeeName:   John
Address:        Apex
Phone:          0065
Date:           11 22 65
Amount:         $1000.65
EIBRESP:        00000000
EIBRESP2:       00000000
USERID:         CICSUSER
HTTP CODE:      0000000200
```

7. Member **DELTAPI** in *USER.ZCEE.CNTL* will delete a record and **PUTAPI** will update an existing record. Submit these jobs and observe the results. Or if you want to modify the request messages in POSTAPI and PUTAPI and recompile and relink these programs and observe the results when different values are provided.

The available records are listed below:

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

Summary

You have use the z/OS Connect build toolkit to generate an API requester archive file and the COBOL copy books that must be included in the COBOL application program when accessing the API requester archive file in a z/OS Connect server. The COBOL applications have been compiled and link-edited and the target API has been tested using various RESTful methods

MVS Batch RESTful API Requester to a MQ API provider

In this section an API requester will be developed and tested where the target API accesses a MQ queue. The batch programs will be put messages (POST) to the queue, do non-destructive gets (GET) and destructive gets (DELETE) of messages from the same queue.

Generate the API requester artifacts from a Swagger document

The first step is to use the API's Swagger document to generate the artifacts required for the COBOL API client application and the API requester archive file. The Swagger document was obtained from the server where the API is hosted. All the files mention in this section are in directory *c:/z/apiRequester/mqapi* on your workstation.

The z/OS Connect build toolkit (ZCONBT) is a Java application shipped with z/OS Connect as a zip file and can be installed on a workstation or in OMVS. Anywhere a Java runtime is available. In this exercise the build toolkit will on Windows.

Tech-Tip: Instructions for installing the z/OS Connect build toolkit can be found at URL https://www.ibm.com/support/knowledgecenter/en/SS4SVW_3.0.0/com.ibm.zosconnect.doc/installing/bt_install.html

1. Begin by reviewing the API's Swagger document for the API. This document identifies the types of HTTP methods supported as well as the path of each HTTP method. It also the contents each method's request JSON message and the response JSON message field contents and field types. This is the blue print for invoking the API.

```
{
  "swagger": "2.0",
  "info": {
    "description": "",
    "version": "1.0.0",
    "title": "mqapi"
  },
  "basePath": "/mqapi",
  "schemes": [
    "https",
    "http"
  ],
  "consumes": [
    "application/json"
  ],
  "produces": [
    "application/json"
  ],
  "paths": {
    "/queue": {
      "get": {
        "tags": [
          "mqapi"
        ],
        "operationId": "getMqGet",
        "parameters": [
```

2. Next review the input parameters that will be passed to the z/OS Connect build toolkit in a properties file. This file is *mqapi.properties* see its below:

```
apiDescriptionFile=./mqapi.json 1
dataStructuresLocation=./syslib 2
apiInfoFileLocation=./syslib 3
logFileDirectory=./logs 4
language=COBOL 5
connectionRef=mqapi 6
requesterPrefix=mq 7
```

1. Identifies the Swagger document that describes the API
2. Provides the directory where the generated COBOL copybooks representing the request and response messages will be written.
3. Provides the directory where the generated COBOL copybook with invocation details of the API will be written.
4. Provides the directory where a log file will be written.
5. Identifies the target language.
6. Identifies the corresponding *zosconnect_endpointConnection* element define in the z/OS Connect server's *server.xml* file.
7. Provides a 3-character prefix to be used for all generated copybooks

3. View file *mqapi.bat* which contains this one line.

```
c:\z\zconbt\bin\zconbt.bat -p=./mqapi.properties -f=./mqapi.ara
```

This is batch command file that when executed invokes the z/OS Connect build toolkit (*zconbt*). The *-p* switch identifies the input properties file and the *-f* switch identifies the name of the generate API requester archive file. The API requester archive (ARA) file will be uploaded and make available to the z/OS Connect server in directory *.../resources/zosconnect/apiRequesters*.

4. On the workstation desktop, locate the *Command Prompt* icon and double click on it to open a DOS session that will allow commands to be entered.

5. Use the change directory command (*cd*) to change to directory *apiRequester/mqapi*, e.g.

```
cd c:\z\apiRequester\mqapi
```

6. Enter command **mqapi** in the DOS command prompt window and you should see output like the below:

```
c:\z\apiRequester\mqapi>c:\z\software\zconbt\bin\zconbt.bat -p=./mqapi.properties -
f=./mqapi.ara
BAQB0000I: z/OS Connect Enterprise Edition Build Toolkit Version 1.2 (20200408-1120).
BAQB0008I: Creating API requester archive from configuration file ./mqapi.properties.
BAQB0040I: The generated API requester is automatically named mqapi_1.0.0 based on the title
mqapi and version 1.0.0 of the API to be called.
BAQB0015I: Start processing operation (operationId: getMqGetService, relativePath: /queue,
method: GET).
BAQB0016I: Successfully processed operation (operationId: getMqGetService, relativePath:
/queue, method: GET).
BAQB0015I: Start processing operation (operationId: postMqPutService, relativePath: /queue,
method: POST).
BAQB0016I: Successfully processed operation (operationId: postMqPutService, relativePath:
/queue, method: POST).

Total 2 operation(s) (success: 2, ignored: 0) defined in api description file: ./mqapi.json
----- Successfully processed operation(s) -----
operationId: getMqGetService, basePath: /mqapi, relativePath: /queue, method: GET
- request data structure : No data structure needs to be generated.
- response data structure : MQ000P01
- api info file : MQ000I01

operationId: postMqPutService, basePath: /mqapi, relativePath: /queue, method: POST
- request data structure : MQ001Q01
- response data structure : No data structure needs to be generated.
- api info file : MQ001I01

BAQB0009I: Successfully created API requester archive file ./mqapi.ara.
```

Note that each HTTP method defined in the Swagger document will normally cause the generation of up to three copy books. One for the request message(Q01), one for the response message(P01) and one for the API details(I01). For this API only two copy books were generated for the messages. One for either the request message or one for the response message. In this case the API did not require both types for any method. Also note that the copy book names are based on the *requesterPrefix* property (e.g. *mq*) and use an ascending sequence number sequence to differentiate between methods.

Important Note: The COBOL programs used in this exercise include these generated copy books with the names as generated above, e.g. the GET COBOL program includes CSC01 prefixed copy books. Occasionally (probably after applying service) the sequence of the methods will change causing the corresponding sequence number of the copy book to also change. Review the sequence of the processing of the methods and change the COBOL applications so they include the correct copy book. Otherwise the compilation of the COBOL program may fail, or unexpected results may be observed.

7. Now explore a sample snippet of the copy book for the request message of a PUT method. Use *EditPad Lite* to open file `c:\z\apiRequester\mqapi\syslib\MQ001Q01`. Scroll down until you see something like the code below.

```

06 ReqBody.
  09 MQMESSAGE2-num          PIC S9(9) COMP-5 SYNC.
  09 MQMESSAGE.
    12 STAT-num              PIC S9(9) COMP-5 SYNC.
    12 STAT.
      15 STAT2-length        PIC S9999 COMP-5 SYNC.
      15 STAT2                PIC X(1).
    12 NUMB-num              PIC S9(9) COMP-5 SYNC.
    12 NUMB.
      15 NUMB2-length        PIC S9999 COMP-5 SYNC.
      15 NUMB2                PIC X(6).
    12 NAME-num              PIC S9(9) COMP-5 SYNC.
    12 NAME.
      15 NAME2-length        PIC S9999 COMP-5 SYNC.

```

Note that each variable has an associated length (e.g. *STAT2-length*). The length of a variable is required since there is no delimiter between the variables in storage. The runtime needs to know the size of a variable at execution time so the request and response messages can be properly converted to and from JSON name/value pairs.

A response message (`c:\z\apiRequester\mqapi\syslib\MQ000P01`) will have a similar layout.

8. An API information copy book (`c:\z\apiRequester\mqapi\syslib\MQ001I01`) has contents providing the API name and the path to be used for the method:

```

03 BAQ-APINAME                PIC X(255)
  VALUE 'mqapi_1.0.0'.
03 BAQ-APINAME-LEN            PIC S9(9) COMP-5 SYNC
  VALUE 11.
03 BAQ-APIPATH                PIC X(255)
  VALUE '%2Fmqapi%2Fqueue'.
03 BAQ-APIPATH-LEN            PIC S9(9) COMP-5 SYNC
  VALUE 16.
03 BAQ-APIMETHOD             PIC X(255)
  VALUE 'GET'.
03 BAQ-APIMETHOD-LEN         PIC S9(9) COMP-5 SYNC
  VALUE 3.

```

Deploy the API Requester Archive file to the z/OS Connect server

Next make the API requester archive file available to the z/OS Connect server.

The API requester archive (ARA) file needs to be deployed to the API requester directory. In this case the target directory is `/var/ats/zosconnect/servers/zceeapir/resources/zosconnect/apiRequesters`.

1. Open a DOS command session and change to directory `C:\z\apiRequester\mqapi`, e.g **cd**
`\z\apiRequester\mqapi`
2. Use the z/OS Connect RESTful administrative interface to deploy the ARA files by using the cURL command embedded in command file `deploy.cmd`. Invoke this command file to deploy the MQ API archive file.

```
c:\z\apiRequester\mqapi>curl -X POST --user Fred:fredpwd --data-binary @mqapi.ara
--header "Content-Type: application/zip" --insecure
https://wg31.washington.ibm.com:9483/zosConnect/apiRequesters
{"name":"mqapi_1.0.0","version":"1.0.0","description":"","status":"Started"
,"apiRequesterUrl":"https://wg31.washington.ibm.com:9483/zosConnect/apiReque
sters/mqapi_1.0.0","connection":"mqapi"}
```

Tech-Tip: If a REST client tool like cURL or Postman was not available then ARA file could have been deployed using FTP to upload the file in binary mode to the `apiRequesters` directory.

Tech-Tip: If an ARA needs to be redeployed, the cURL command with a PUT method can be used to stop the API requester

curl -X PUT --user Fred:fredpwd --insecure

https://wg31.washington.ibm.com:9483/zosConnect/apiRequesters/mqapi_1.0.0?status=stopped

And the cURL command with a DELETE method can be used to delete the API requester.

curl -X DELETE --user Fred:fredpwd --insecure

https://wg31.washington.ibm.com:9483/zosConnect/apiRequesters/mqapi_1.0.0

otherwise the redeployment of the ARA file will fail.

3. The z/OS Connect server where the API requester will be installed has the configuration below in its server.xml file.

```
<server description="API Requester">
  <!-- Enable features -->
  <featureManager>
    <feature>zosconnect:apiRequester-1.0</feature>
  </featureManager>

  <zosconnect_endpointConnection id="mqapi" 1
    host="http://wg31.washington.ibm.com"
    port="9120"
    basicAuthRef="myBasicAuth"
    connectionTimeout="10s"
    receiveTimeout="20s" />

  <zosconnect_authData id="myBasicAuth" 2
    user="Fred"
    password="fredpwd" />

</server>
```

1. Identifies the system which hosts the target API as well as any security information and time out parameters. Note that the ID of this element matches the value *connectionRef* property used when the ARA was generated by the z/OS Connect build toolkit.
2. The target server uses basic authentication, so we are simply provided a user identity and password. The password can be encrypted and/or TLS can be used for security.

Tech-Tip: The host and port in the *endpointConnection* element specifies the host and port of the target API provider. Normally the z/OS Connect server performs the conversion of the COBOL data structure to JSON and then acts as a JSON REST client to the target API provider. In this exercise the target API provider is the same z/OS Connect server. This is why this host and port is also used by the batch API requester client in its environment variables.

Move the COBOL copy books to a MVS data set

Next make the generated COBOL copy books available for including into the COBOL program.

The generated COBOL copy books need to be available in an MVS data set which is included in the compiler SYSLIB concatenation sequence. In the data set is *USER1.ZCEE.SOURCE*. Use a DOS command prompt session and the commands below to move the copy books file to this data set. Authenticate as user USER1 (password USER1).

1. Open a DOS command prompt and use the change directory command to go to directory C:\z\apiRequester\filequeue\syslib, e.g. **cd \z\apiRequester\mqapi\syslib**
2. Start a file transfer session with the WG31 host using the *ftp* command, e.g. **ftp wg31**
3. Logon as USER1 and then use the *cd* command to change to directory to data set *USER1.ZCEE.SOURCE*, e.g. **cd zcee.source**
4. Toggle prompting off by entering command **prompt**
5. Perform multiple put request by using the multiple put command, **mput imq***
6. When the last transfer has completed enter the **quit** command.

```
c:\z\apiRequester\cscvinc>cd syslib
c:\z\apiRequester\cscvinc\syslib>ftp wg31.washington.ibm.com
Connected to wg31.washington.ibm.com.
220-FTP 16:26:23 on 2018-02-15.
220 Connection will close if idle for more than 200 minutes.
User (wg31.washington.ibm.com:(none)): user1
331 Send password please. user1
Password:
230 USER1 is logged on. Working directory is "USER1.".
ftp> cd zcee.source
250 The working directory "USER1.ZCEE.SOURCE" is a partitioned data set
ftp> prompt
Interactive mode Off .
ftp> mput MQ*
200 Port request OK.
125 Storing data set USER1.ZCEE.SOURCE(MQ000I01)
250 Transfer completed successfully.
ftp: 533 bytes sent in 0.32Seconds 1.66Kbytes/sec.
200 Port request OK.
sent in 0.10Seconds 110.89Kbytes/sec.
200 Port request OK.
.....
125 Storing data set USER1.ZCEE.SOURCE(MQ01I01)
250 Transfer completed successfully.
ftp: 7393 bytes sent in 0.32Seconds 22.89Kbytes/sec.
ftp> quit
```

Review the application programs

Now that the copy books have been generated and are available in a data set the next step is compile and link edit the application programs. The source for the application programs are in *USER1.ZCEE.SOURCE*. Member DELMQAPI invokes the DELETE method, member GETMQAPI invokes the GET method and PSTMQAPI invokes the POST method. No changes need to be made to these programs.

Tech-Tip: Data set USER1.ZCEE.SOURCE can be accessed either by using the IBM z/OS Explorer filter or by using the Personal Communication icon on the desktop and logging on to TSO and use ISPF option 3.4 to access this data set. If you have any questions about either method ask the instructor.

1. Begin by reviewing the application programs MQGET. Browse data set *USER1.ZCEE.SOURCE* and open member MQGET. Scroll down until you see these lines of code.

```
* Copy API Requester required copybook
COPY BAQRINFO.

* Request and Response
01 PUT-REQUEST.
   10 FILLER                      PIC X(1) .
01 PUT-RESPONSE.
   COPY MQ000P01.
* Structure with the API information
01 PUT-INFO-OPER1.
   COPY MQ000I01.
```

These lines of code copy into the source the required z/OS Connect copy book (BAQRINFO) and the copy books generated by the z/OS Connect build tool kit. If when you generated the copy book earlier in the exercise and the sequence number used for the GET method was not 02 then you need to change the COBOL source to match sequence generated when you invoke the ZCONBT tool.

2. Scroll down further and you will the call to the z/OS Connect API requester stub module.

```
*-----*
* Call the communication stub                                     *
*-----*
* Call the subsystem-supplied stub code to send
* API request to zCEE
  CALL COMM-STUB-PGM-NAME USING
      BY REFERENCE  PUT-INFO-OPER1
      BY REFERENCE  BAQ-REQUEST-INFO
      BY REFERENCE  BAQ-REQUEST-PTR
      BY REFERENCE  BAQ-REQUEST-LEN
      BY REFERENCE  BAQ-RESPONSE-INFO
      BY REFERENCE  BAQ-RESPONSE-PTR
      BY REFERENCE  BAQ-RESPONSE-LEN.
* The BAQ-RETURN-CODE field in 'BAQRINFO' indicates whether this
* API call is successful.
```

Note that the parameters are being passed by reference. This means that the z/OS Connect API requester stub will have direct access to the program storage for both accessing data and making changes (i.e. the response message).

3. Scroll down further and you will see the code where the application displays the results in the response message.

```

IF BAQ-SUCCESS THEN
  DISPLAY "NUMB:  " numb    of PUT-RESPONSE
  DISPLAY "NAME:  " name    of PUT-RESPONSE
  DISPLAY "ADDRX: " addrx   of PUT-RESPONSE
  DISPLAY "PHONE: " phone   of PUT-RESPONSE
  DISPLAY "DATEX: " datex   of PUT-RESPONSE
  DISPLAY "AMOUNT: " amount  of PUT-RESPONSE
  DISPLAY "HTTP CODE: " BAQ-STATUS-CODE

```

4. Exit member *MQGET* and open member *MQPUT*. The code for this program is very similar to the code for *MQGET* with the exception that all variables are provided in a JSON request message. Scroll down and find the code below.

```

*-----*
* Set up the data for the API Requester call      *
*-----*

MOVE 1 to MQMESSAGE2-num.
MOVE "837367" TO numb2 IN PUT-REQUEST.
MOVE LENGTH of numb2 in PUT-REQUEST to
      numb2-length IN PUT-REQUEST.

MOVE "John" TO name2 IN PUT-REQUEST.
MOVE LENGTH of name2 in PUT-REQUEST to
      name2-length IN PUT-REQUEST.

MOVE "Apex" TO addrx2 IN PUT-REQUEST.
MOVE LENGTH of addrx2 in PUT-REQUEST to
      addrx2-length IN PUT-REQUEST.

MOVE "0065" TO phone2 IN PUT-REQUEST.
MOVE LENGTH of phone2 in PUT-REQUEST to
      phone2-length IN PUT-REQUEST.

MOVE "11 22 65" TO datex2 IN PUT-REQUEST.
MOVE LENGTH of datex2 in PUT-REQUEST to
      datex2-length IN PUT-REQUEST.

MOVE "$1000.65" TO amount2 IN PUT-REQUEST.
MOVE LENGTH of amount2 in PUT-REQUEST to
      amount2-length IN PUT-REQUEST.

```

In this case the length of the property value and finally the property value

moved to the request copy book.

Compile and link-edit the application programs

The applications programs now are ready to be compiled and link-edited.

1. Submit the job in member **APIRMQ** in data set *USER1.ZCEE.CNTL*.

```
//USER1S JOB MSGCLASS=H,NOTIFY=&SYSUID
//* JCLLIB ORDER=SYS1.ECOBOL.SIGYPROC
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,PARM.COBOL='NODYNAM'
//COBOL.SYSIN DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(MQGET)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
//          DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(MQGET)
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,PARM.COBOL='NODYNAM'
//COBOL.SYSIN DD DISP=SHR,DSN=USER1.ZCEE.SOURCE(MQPUT)
//COBOL.SYSLIB DD DISP=SHR,DSN=USER1.ZCEE.SOURCE
//          DD DISP=SHR,DSN=ZCEE30.SBAQCOB
//LKED.SYSLMOD DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB(MQPUT)
```

Note that the CALL statement in each of this program uses a variable for the program name z/OS Connect stub program (BAQCSTUB). This means that this program will be dynamically linked at execution. The load module library containing BAQCTUB (ZCEE30.SBAQLIB) must be available in the STEPLIB concatenation sequence when the program is executed.

The execution of this job should complete with at most a 4-return code.

Test the API requester application programs

The JCL to execute these programs can be found in `USER1.ZCEE.CNTL`, the member names in this data set match the program names. The members are `DLTMQAPI`, `GETMQAPI` and `PSTMQAPI`.

1. Submit the job in member **MQGET** in *USER1.ZCEE.CNTL*.

```
//GETAPI EXEC PGM=MQGET
//STEPLIB DD DISP=SHR,DSN=USER1.ZCEE.LOADLIB
// DD DISP=SHR,DSN=ZCEE30.SBAQLIB
//SYSPRINT DD SYSOUT=*
//CEEOPTS DD *
        POSIX(ON),
        ENVAR("BAQURI=wg31.washington.ibm.com",
        "BAQPORT=9120")
```

2. The job should complete with a condition of 200 (HTTP success) and have the following output for `SYSPRINT`.

```
NUMB: 000100
NAME: S. D. BORMAN
ADDRX: SURREY, ENGLAND
PHONE: 32156778
DATEX: 26 11 81
AMOUNT: $0100.11
HTTP CODE: 0000000200
```

Tech-Tip: An HTTP code of 200 indicates success. For an explanation of HTTP codes see URL https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

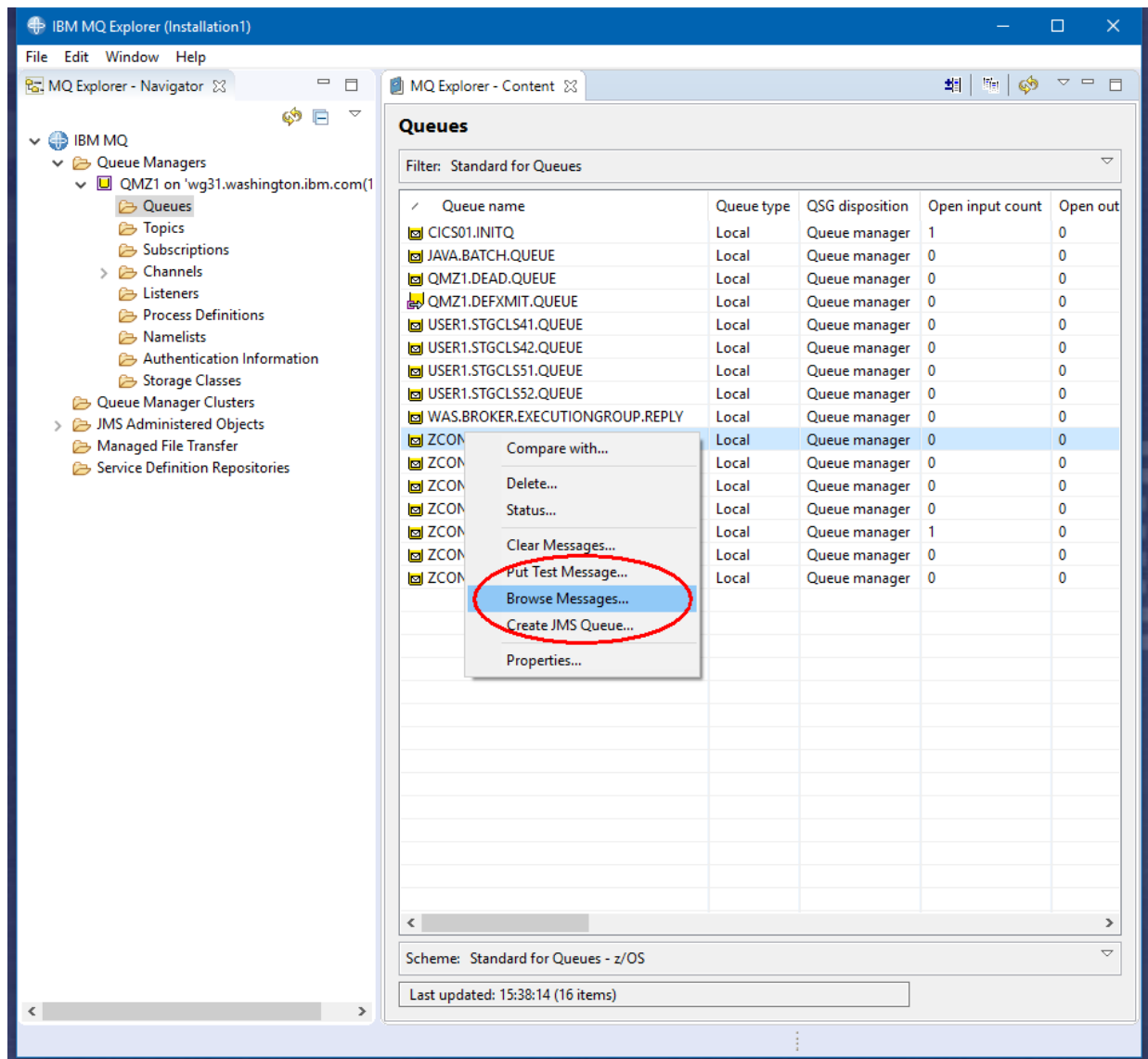
3. Member **MQPUT** in *USER.ZCEE.CNTL* will invokes a POST HTTP method which will put a message on the queue. Submit this job for execution and observe the results. The POST method for this API does not return a response message.

```
HTTP CODE: 0000000204
```

Tech-Tip: An HTTP code of 204 indicates that the server processed the request and is not returning any content. For an explanation of HTTP codes see URL https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

4. Confirm by opening the *MQExplorer* icon on the desktop and browsing the messages in the *ZCONN2.DEFAULT.MQZCEE.QUEUE* queue.

- Select *QMZ1* under **Queue Managers** and right mouse button click
- Select the *Connect* option.
- Once connected, expand the *Queues* folder and select *ZCONN2.DEFAULT.MQZCEE.QUEUE* and right mouse button click and select the *Browse Messages* option.



5. You see a list of the messages currently in the queue, something like the list below:

Message browser

Queue Manager Name: QMZ1
Queue Name: ZCONN2.DEFAULT.MQZCEE.QUEUE

Position	Put date/time	User identifier	Put application name	Format	Total length	Data length	Message data
1	Jul 23, 2019 3:34:46 PM	ATSSERV	ZCEEAPIR		260	80	837367John
2	Feb 2, 2017 6:16:44 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000100S. D. BORMAN
3	Feb 2, 2017 6:17:27 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000102J. T. CZAYKOV
4	Feb 2, 2017 6:17:44 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000104M. B. DOMBEY
5	Feb 2, 2017 6:18:02 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000106A. I. HICKSON
6	Feb 2, 2017 6:18:30 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000111ALAN TULIP
7	Feb 2, 2017 6:18:45 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000762SUSAN MALA
8	Feb 2, 2017 6:19:06 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	000983J. S. TILLING
9	Feb 2, 2017 6:19:22 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	001222D.J.VOWLES
10	Feb 2, 2017 6:19:36 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	001781TINA J YOUNG
11	Feb 2, 2017 6:20:07 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	003210B.A. WALKER
12	Feb 2, 2017 6:29:33 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	003214PHIL CONWA
13	Feb 2, 2017 6:29:49 PM	MITCHJ	MQ Explorer 9.0.0	MQSTR	122	122	003890BRIAN HARDE

Scheme: Standard for Messages

Last updated: 15:40:50 (44 items)

All available messages on the queue have been browsed. Press the refresh button for new messages.

Refresh Close

Summary

You have use the z/OS Connect build toolkit to generate an API requester archive file and the COBOL copy books that must be included in the COBOL application program when accessing the API requester archive file in a z/OS Connect server. The COBOL applications have been compiled and link edited, and the target API has been tested using various RESTful methods