

A case study: The WSC experiences developing a CICS REST client from an OAS3 specification



*Mitch Johnson
John Brefach
October 3, 2025*

Table of Contents

Overview.....	3
Reviewing the API specification's influence on the COBOL code	4
<i>Request and response messages</i>	5
<i>More about the response copy book.....</i>	13
The response copy book and the LINKAGE SECTION	13
Determining the HTTP status code using the BAQBASE structure.....	15
Handling Multiple occurrences (arrays)	17
Deploying the API Requester WAR file	20
<i>Server XML configuration Updates</i>	20
<i>Make the WAR file available to the server</i>	21
Using the OMVS copy (cp) command.....	21
Using the curl command	22
<i>Security Updates.....</i>	22
<i>CICS Updates</i>	23
The CICS API requester client application	24
<i>MAIN-PROCESS SECTION</i>	24
<i>INVOKE-API SECTION</i>	27
<i>GET-EACH-EMPLOYEE SECTION.....</i>	31
<i>BAQINIT SECTION.....</i>	32
<i>BAQEXEC SECTION.....</i>	33
<i>BAQEXEC-GET-DATA-NAME SECTION (BAQGETN)</i>	34
<i>BAQFREE SECTION.....</i>	35
<i>BAQTERM SECTION</i>	35
<i>Compiling and link-editing the application program</i>	37
Testing the CICS API requester application program	38
Gradle on z/OS	42
<i>Installing and enabling the Gradle Build Tool on z/OS</i>	42
<i>Building the API requester artifacts from an OpenAPI3 specification document.....</i>	47
Convert ASCII copy books to EBCDIC copy books.....	51

Overview

This document was written by the Washington Systems Center to describe our experiences when developing a COBOL program that uses z/OS Connect API requester support to invoke a REST API. We started this project with two goals. The first goal was to describe how the details of the API as provided in an OpenAPI3 specification document impact the COBOL code in the API client requester application. And the second goal was to explore installing and using the Gradle build tool (see URL <https://gradle.org>) on z/OS. We know that using Gradle on z/OS is unlikely but we wanted to understand the steps required to install and enable Gradle on z/OS and show how the use of Gradle can be integrated with MVS utilities, MVS data sets, OMVS commands, etc. The lessons learned may be useful in other situations.

We chose to develop a CICS COBOL REST client to demonstrate the required COBOL coding since (1) we thought there are really no significance differences in the COBOL coding required for the other z/OS client environments and (2) because we thought CICS simply offered the more interesting execution environment. We know that there are differences between the environments, but the differences are primarily in the configuring the connection to the z/OS Connect server. For information about these connection differences, consult the z/OS Connect documentation.

For simplicity, we chose to use a BMS-enabled CICS application program. The BMS application interacts with a terminal user and for collecting and displaying information. But rather than developing a single program that included both CICS BMS code and the z/OS Connect related code, we chose a solution that followed the Model-View-Controller (MVC) architectural pattern. That is, the presentation logic (e.g., the View and Controller) would be in a BMS enabled program and the business logic,(e.g., the Model) would be a separate CICS linkable program. Separating the functions in a common program allows the exposure of the z/OS Connect HOST APIs to different clients interfaces. For example, the same CICS linkable program could also be accessed from clients using CICS Web Services support, or using ECI or EXCI protocols, or using JMS for sending and receiving request and response messages via the CICS MQ Bridge, or a Java client using the JCICS classes running in CICS Liberty servers or even other REST client by exposing the CICS linkable program using z/OS Connect API provider support.

An overview of the application used in this document is that once information was entered by the terminal user, the BMS program would do a CICS EXEC LINK to the CICS API requester application passing a channel with a request container. The CICS API requester application would use the information in the request container and the z/OS Connect Host API verbs to invoke the remote API, the results would be returned to the BMS application in a response container. A success container when the API returned an HTTP 200 (success) or an exception container if an HTTP 400 or HTTP 500 status code was returned.

.

Reviewing the API specification's influence on the COBOL code

We started by reviewing the API's specification document. The details of the API's request and response messages and the details of the message payloads as defined in the API's specification directly impact key aspects of the COBOL code in a REST client application. The details of what we learn about these relationships are covered in this section..

Below is a screen shot from using the Liberty API Explorer (</api/explorer>) to discuss the details of the API. This screen shows that the *Employee Roster* API can be accessed using six methods. Three *GET* methods for retrieving employee information (these methods are differentiated by their URI paths) and one *POST* method for adding a new employee, one *PUT* method for updating an employee's information and one *DELETE* method for removing or deleting an employee's information.

The screenshot shows the Open Liberty API Explorer interface. At the top, there is a navigation bar with the Open Liberty logo, a search bar, and a 'Filter' button. Below the header, it says 'Liberty REST APIs 1.0.0 OAS 3.0'. A sub-header indicates 'Discover REST APIs available within Liberty'. A 'Servers' dropdown menu is set to 'https://designer.ibm.com:9449'. The main content area is titled 'employee roster'. It lists six API endpoints:

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

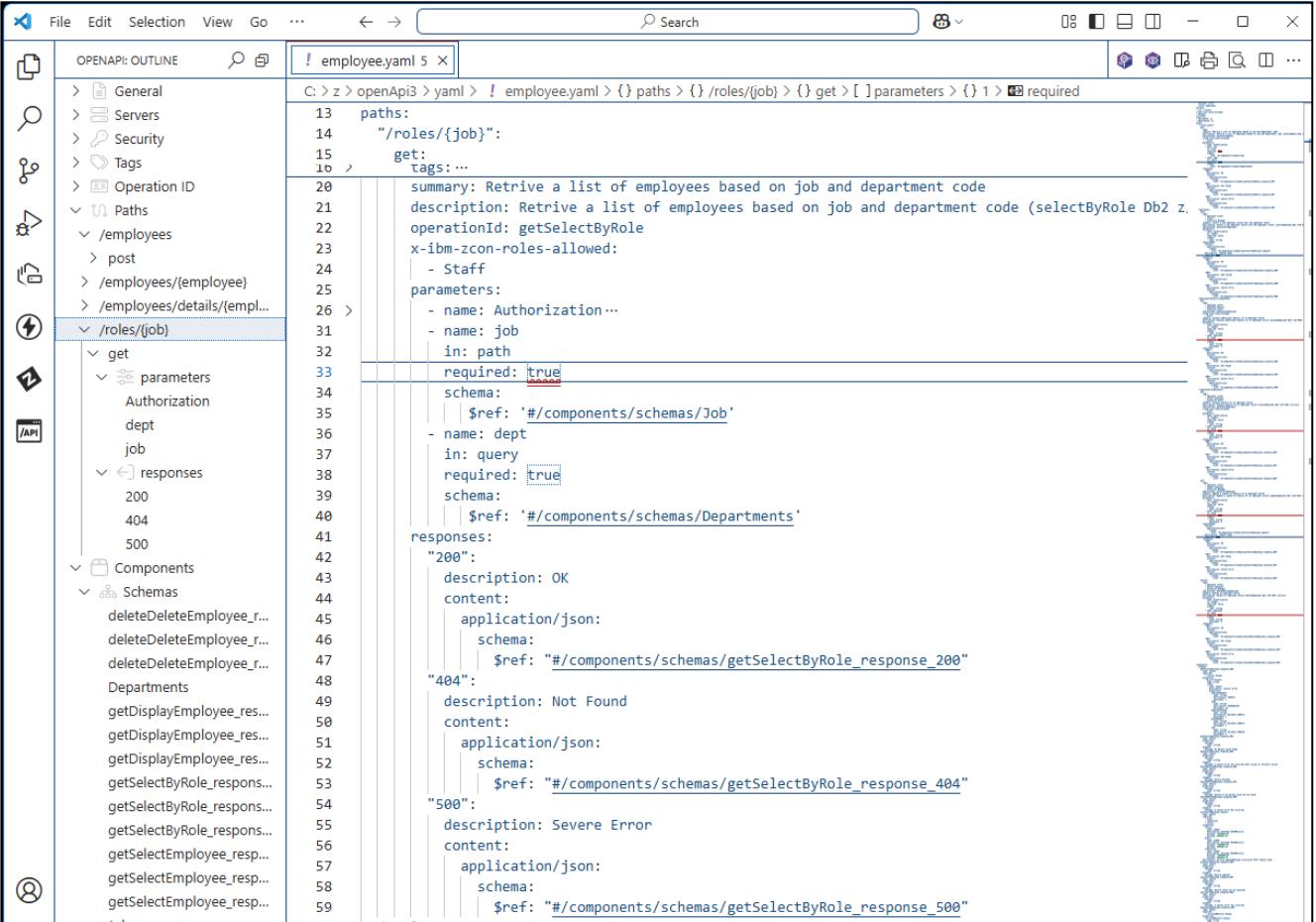
Each endpoint row includes a lock icon and a dropdown arrow.

We chose to use the *GET* method for URI path */roster/roles/{job}* as the subject for exploring. This method was chosen because it has the most interesting response because multiple individual items are returned in the response message. We also knew that the basic process used with this method could be applied to develop REST client applications for the other methods.

Request and response messages

The first part of the specification document we explored were the GET method's request and response messages. Normally the z/OS Connect API requester will parse the specification document and generate the correct COBOL code, but we wanted to understand the reasoning behind the details of the generation process..

The API specification document provided the basic details of this method in the section below. This tells us that the request message includes parameter *job* as a *path* parameter in the URI path and that parameter *dept* is a *query* parameter in the URI path, e.g., */roster/roles/PRES/?dept=A01*. Also, details of the different HTTP response codes (e.g., HTTP 200, HTTP 404, or HTTP 500) to be expected are provided.



The screenshot shows the Visual Studio interface with the "OPENAPI OUTLINE" tool window open. The left pane displays a tree view of the API definition, including sections for General, Servers, Security, Tags, Operation ID, Paths, /employees, /employees/{employee}, /employees/details/{empl...}, and /roles/{job}. The right pane shows the YAML code for the "/roles/{job}" endpoint's "get" method. The code defines parameters for Authorization, dept, and job, and specifies responses for 200, 404, and 500 status codes. The "dept" parameter is marked as required and has a schema pointing to "#/components/schemas/Departments". The "job" parameter also has a schema pointing to "#/components/schemas/Job". The "responses" section provides descriptions and content types for each status code.

```
C: > z > openApi3 > yaml > employee.yaml > {} paths > {} /roles/{job} > {} get > [ ] parameters > {} 1 > required
13   paths:
14     "/roles/{job}":
15       get:
16         tags: ...
17
18         summary: Retrieve a list of employees based on job and department code
19         description: Retrieve a list of employees based on job and department code (selectByRole Db2 z...
20         operationId: getSelectByRole
21         x-ibm-zcon-roles-allowed:
22           - Staff
23
24         parameters:
25           - name: Authorization...
26           - name: job
27             in: path
28             required: true
29             schema:
30               $ref: '#/components/schemas/Job'
31           - name: dept
32             in: query
33             required: true
34             schema:
35               $ref: '#/components/schemas/Departments'
36
37         responses:
38           "200":
39             description: OK
40             content:
41               application/json:
42                 schema:
43                   $ref: '#/components/schemas/getSelectByRole_response_200'
44           "404":
45             description: Not Found
46             content:
47               application/json:
48                 schema:
49                   $ref: '#/components/schemas/getSelectByRole_response_404'
50           "500":
51             description: Severe Error
52             content:
53               application/json:
54                 schema:
55                   $ref: '#/components/schemas/getSelectByRole_response_500'
```

N.B.: The screen shots of YAML files in this document were taken using the OpenAPI (Swagger) Editor extension in Visual Studio.

Relevant information regarding the method of the API, e.g., the URI path and method name is extracted from the specification document and saved in a generated copy book that needs to be included in the COBOL program. The COBOL program passes this information to z/OS Connect where it is used to build the URL for accessing the method and to identify which method is being invoked. Below is how these details are provided in a COBOL copy book. The application should not change the contents of any of these variables.

```
* ++++++  
* This file contains the generated API information structure  
* which is passed to the Host API via the BAQEXEC call.  
* ++++++  
01 BAQ-API-INFO-EMP00I01.  
    03 BAQ-API-INFO-EYE          PIC X(4)  
        VALUE 'BAQA'.  
    03 BAQ-API-INFO-LENGTH      PIC 9(9) COMP-5 SYNC  
        VALUE 1052.  
    03 BAQ-API-INFO-VERSION     PIC 9(9) COMP-5 SYNC  
        VALUE 1.  
    03 BAQ-API-INFO-RESERVED01  PIC 9(9) COMP-5 SYNC  
        VALUE 0.  
    03 BAQ-API-NAME             PIC X(255)  
        VALUE 'roster'.  
    03 BAQ-API-NAME-LEN         PIC 9(9) COMP-5 SYNC  
        VALUE 6.  
    03 BAQ-API-PATH             PIC X(255)  
        VALUE '%2Froles%2F%7Bjob%7D'.  
    03 BAQ-API-PATH-LEN         PIC 9(9) COMP-5 SYNC  
        VALUE 20.  
    03 BAQ-API-METHOD           PIC X(255)  
        VALUE 'GET'.  
    03 BAQ-API-METHOD-LEN       PIC 9(9) COMP-5 SYNC  
        VALUE 3.  
    03 BAQ-API-OPERATION        PIC X(255)  
        VALUE 'getSelectByRole'.  
    03 BAQ-API-OPERATION-LEN    PIC 9(9) COMP-5 SYNC  
        VALUE 15.  
50
```

Elsewhere in the specification document, the URI path fields (*job* and *dept*) are defined as unconstrained strings (*type: string*). Unconstrained in the respect that their respective lengths are not bounded by a *maxLength* attribute.

The screenshot shows the API Platform interface with the 'OPENAPI: OUTLINE' tab selected. The left sidebar contains navigation icons and the outline tree. The main area displays the contents of the 'employee.yaml' file, which defines a schema for 'Job' with an enum type containing values like 'A00', 'B01', etc. The code editor shows line numbers from 270 to 643.

```
C: z > openApi3 > yaml > ! employee.yaml > {} components > {} schemas > {} Job > type
270 components:
271 schemas:
621 Departments:
622 enum:
623 - A00
624 - B01
625 - C01
626 - E01
627 - D11
628 - D21
629 - E11
630 - E21
631
632 Job:
633 type: string
634 enum:
635 - "FIELDREP"
636 - "OPERATOR"
637 - "CLERK"
638 - "DESIGNER"
639 - "ANALYST"
640 - "SALESREP"
641 - "MANAGER"
642 - "PRES"
643
```

The maximum length of the *job* and *dept* variables can be deduced because the valid values for these fields are enumerated and their maximum lengths are constrained by the lengths of the values in the enumerated lists. Note that if the size of a string can not be deduced, the length of the COBOL PIC X specification variable will be set to a default value based on the API generation property *defaultCharacterMaxLength*, which has a default value of 255, e.g., a COBOL PIC X(255).

```
35 * ++++++
36
37      01 BAQBASE-EMP00Q01.
38          03 requestPathParameters.
39              06 job-length           PIC S9999 COMP-5 SYNC.
40                  06 job            PIC X(8).
41          03 requestQueryParameters.
42              06 dept-length        PIC S9999 COMP-5 SYNC.
43                  06 dept          PIC X(3).
44
```

N.B.: The reason for these *-length* variables for containing the actual length of valid data at execution timer is because of how COBOL handles string variables versus how other programming languages manage string variables. Strings in COBOL are in continuous storage and are a finite maximum size as determined by the compiler based on the PIC attribute. The COBOL length function will always return the maximum length of a variable, not the actual length of the data in the variable. This works well for COBOL-to-COBOL calls but presents a problem for interlanguage calls.

Other languages such as Java maintain strings as simple objects and strings lengths are terminated by the presence of a null character, x'00'. So, the size of a string can vary and the length can easily be programmatically determined by Java by using the null character as the termination of a string.

So, if the COBOL program did not provide the length of the variable data, another programming language like Java would look for the first null character in storage and send the variable data plus the content of whatever else was in storage as the value of the variable (aka known as garbage).

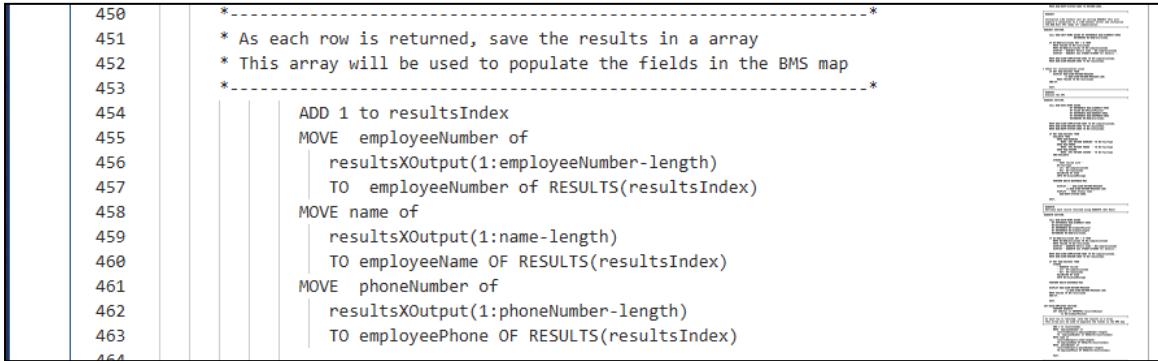
The actual length of the valid data in these variables is not apparent, and this could be a problem since a comparison for job with contents of “PRES ” versus a comparison for “PRES” would provide different results. To address this, additional variables are generated in the COBOL source to hold the actual length of the data in the variables. These fields are identified in the COBOL source by the presence of the *-length* suffix added to the variable name, e.g., *job-length* and *dept-length*. It is the COBOL REST client application’s responsibility to provide the actual length of the data using the corresponding *-length* variable, as shown in the example code as shown below.

```

69      * Edit job code by removing leading blanks
70          MOVE FUNCTION TRIM(job in requestContainer leading)
71          | to job in BAQBASE-EMP00Q01
72      * Move job code to request path parameter
73          MOVE LENGTH of job in BAQBASE-EMP00Q01 to
74          | job-length IN BAQBASE-EMP00Q01
75      * Adjust length to remove trailing spaces
76          MOVE zero to ws-length
77          INSPECT FUNCTION REVERSE (job IN BAQBASE-EMP00Q01)
78          | TALLYING ws-length FOR ALL SPACES
79          SUBTRACT ws-length FROM
80          | job-length in BAQBASE-EMP00Q01
81
82      * Move department code to query parameter
83          MOVE department in requestContainer
84          | to dept IN BAQBASE-EMP00Q01.
85          MOVE LENGTH of dept in BAQBASE-EMP00Q01 to
86          | dept-length IN BAQBASE-EMP00Q01
87      * Adjust length to remove trailing spaces
88          MOVE ZERO TO ws-length
89          INSPECT FUNCTION REVERSE (dept in BAQBASE-EMP00Q01)
90          | TALLYING ws-length FOR ALL SPACES
91          SUBTRACT ws-length FROM
92          | dept-length IN BAQBASE-EMP00Q01

```

The same applies for variables in the response message. In this case, z/OS Connect supplies the actual length of the data in the response's variables in the *-length* variables so the COBOL program would only work with valid data, see an example below.



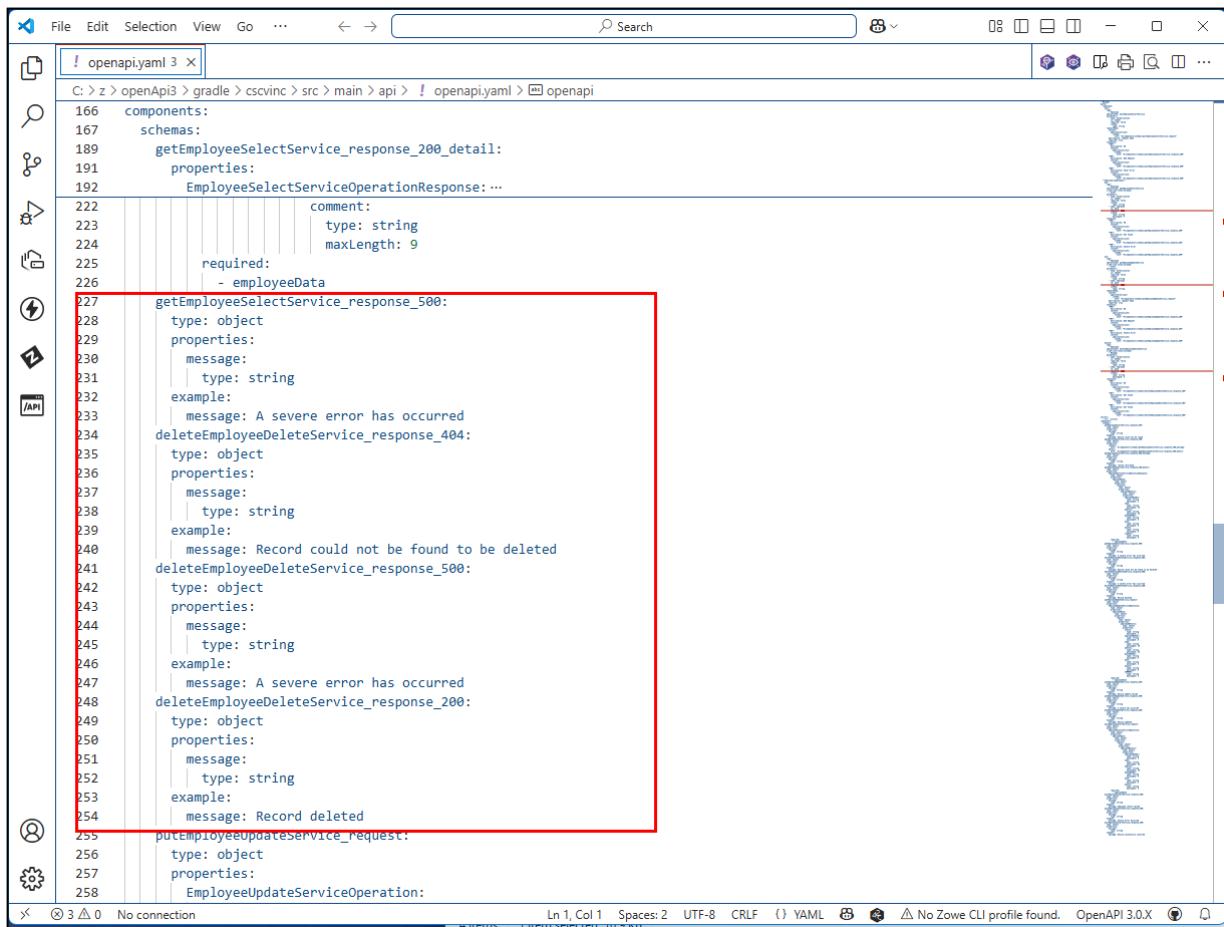
```

450      *-----*
451      * As each row is returned, save the results in a array
452      * This array will be used to populate the fields in the BMS map
453      *
454      | ADD 1 to resultsIndex
455      | MOVE employeeNumber of
456      |   resultsXOutput(1:employeeNumber-length)
457      |   TO employeeNumber of RESULTS(resultsIndex)
458      | MOVE name of
459      |   resultsXOutput(1:name-length)
460      |   TO employeeName OF RESULTS(resultsIndex)
461      | MOVE phoneNumber of
462      |   resultsXOutput(1:phoneNumber-length)
463      |   TO employeePhone OF RESULTS(resultsIndex)
464

```

Remember, for both request and response message, using the *-length* variables prevents the sending and receiving of any unintentional or extraneous characters in these variables.

As stated above, if the length of a variable string is not constrained or can not be deduced, the default maximum character property (*defaultCharacterMaxLength*) is used to control the COBOL being generated.

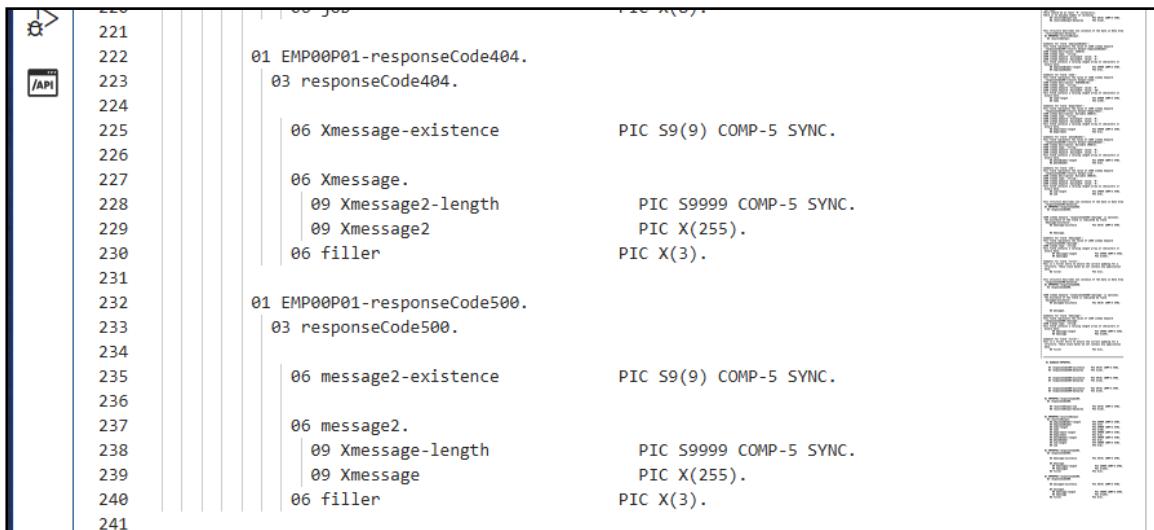


```

openapi: 3.0.0
components:
  schemas:
    getEmployeeSelectService_response_200_detail:
      properties:
        EmployeeSelectServiceOperationResponse:
          comment:
            type: string
            maxLength: 9
          required:
            - employeeData
    getEmployeeSelectService_response_500:
      type: object
      properties:
        message:
          type: string
        example:
          message: A severe error has occurred
    deleteEmployeeDeleteService_response_404:
      type: object
      properties:
        message:
          type: string
        example:
          message: Record could not be found to be deleted
    deleteEmployeeDeleteService_response_500:
      type: object
      properties:
        message:
          type: string
        example:
          message: A severe error has occurred
    deleteEmployeeDeleteService_response_200:
      type: object
      properties:
        message:
          type: string
        example:
          message: Record deleted
    putEmployeeUpdateService_request:
      type: object
      properties:
        EmployeeUpdateServiceOperation:

```

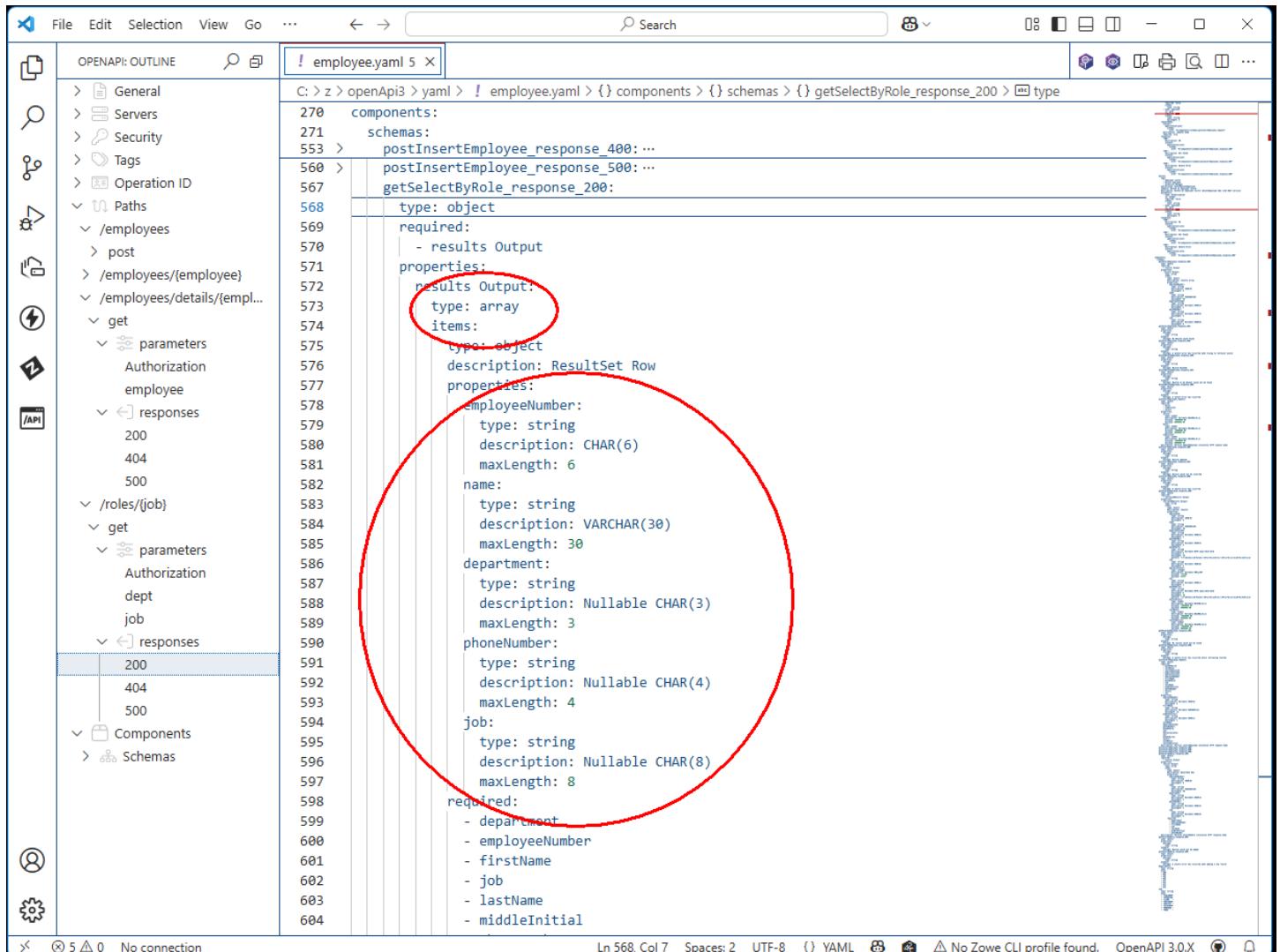
As shown in the example below, the various *message* strings, will be generated as PIC X(255) variables.



The screenshot shows a COBOL editor interface with a toolbar at the top. The main area displays COBOL code. The code includes several 01-level definitions for messages, each with 03-level sub-definitions for response codes and message lengths. The message definitions are annotated with comments indicating their type (e.g., 'Xmessage', 'Xmessage2') and length (e.g., 'S9(9)', 'X(255)'). The code is numbered from 221 to 241. The right side of the editor shows a preview of the generated assembly or machine code.

```
221
222      01 EMP00P01-responseCode404.
223      | 03 responseCode404.
224
225      | 06 Xmmessage-existence          PIC S9(9) COMP-5 SYNC.
226
227      | 06 Xmmessage.
228      | | 09 Xmmessage2-length        PIC S9999 COMP-5 SYNC.
229      | | 09 Xmmessage2              PIC X(255).
230      | | 06 filler                  PIC X(3).
231
232      01 EMP00P01-responseCode500.
233      | 03 responseCode500.
234
235      | 06 message2-existence        PIC S9(9) COMP-5 SYNC.
236
237      | 06 message2.
238      | | 09 Xmmessage-length        PIC S9999 COMP-5 SYNC.
239      | | 09 Xmmessage              PIC X(255).
240      | | 06 filler                  PIC X(3).
241
```

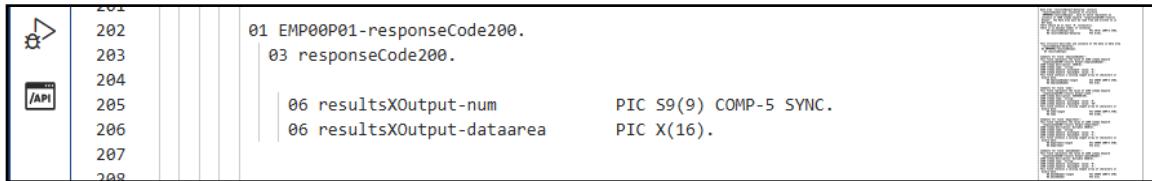
The response message for the GET method for URI path `/roster/roles/{job}` is shown below:



```
C: > z > openApi3 > yaml > ! employee.yaml > {} components > {} schemas > {} getSelectByRole_response_200 > type
270 components:
271   schemas:
553     postInsertEmployee_response_400: ...
560   postInsertEmployee_response_500: ...
567   getSelectByRole_response_200:
568     type: object
569       required:
570         - results Output
571       properties:
572         results Output:
573           type: array
574             items:
575               type: object
576               description: ResultSet Row
577               properties:
578                 employeeNumber:
579                   type: string
580                   description: CHAR(6)
581                   maxLength: 6
582                 name:
583                   type: string
584                   description: VARCHAR(30)
585                   maxLength: 30
586                 department:
587                   type: string
588                   description: Nullable CHAR(3)
589                   maxLength: 3
590                 phoneNumber:
591                   type: string
592                   description: Nullable CHAR(4)
593                   maxLength: 4
594                 job:
595                   type: string
596                   description: Nullable CHAR(8)
597                   maxLength: 8
598             required:
599               - department
600               - employeeNumber
601               - firstName
602               - job
603               - lastName
604               - middleInitial
```

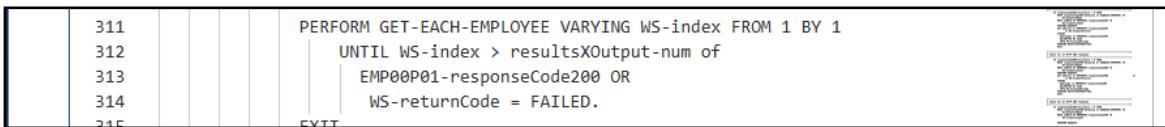
Note that these variables are constrained with `maxLength` attributes. If this attribute were not present, the default lengths for strings would be set to the value of `defaultCharacterMaxLength`.

Notice that the response shown on the previous page will include an array of results (*type : array*). The number of elements in the array will be provided to the COBOL application in a variable with a **-num** suffix, see variable *resultsXOutput-num* in the example below:



```
201
202      01 EMP00P01-responseCode200.
203          03 responseCode200.
204
205          06 resultsXOutput-num          PIC S9(9) COMP-5 SYNC.
206          06 resultsXOutput-dataarea    PIC X(16).
207
208
```

The COBOL program will need to use the z/OS Connect BAQGETN Host API in a loop to process each entry in the array. The variable *resultsXOutput-num* will control how the number of times the loop needs to be repeated.



```
311      PERFORM GET-EACH-EMPLOYEE VARYING WS-index FROM 1 BY 1
312          UNTIL WS-index > resultsXOutput-num of
313              EMP00P01-responseCode200 OR
314                  WS-returnCode = FAILED.
315
316      EXIT.
```

N.B.: In this document we tried to be consistent with terminology. Some readers may use the term 'list' to refer to a set of multiple instance of an element; others may use the term 'table' for the same concept while others may use the term 'array'. In this document, they are equivalent.

More about the response copy book

The z/OS Connect Host APIs is a set of APIs that can be called from a COBOL program to access z/OS Connection REST client functions. The APIs are documented in the section **Understanding the HOST API** at URL <https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=apis-understanding-host-api>

A COBOL REST client application using the Host APIs will usually make one BAQINIT (initialize the API storage and connect to the server) request, one or more BAQEXEC (execute an API) requests, one BAQFREE (free storage) request and one BAQTERM (terminate a connection) request. The number of BAQGETN (get next element) requests and their usage is based on the generated response message copy book. In this section we see how the contents of the generated response copybook determine when and how the BAQGETN requests are used in the COBOL code.

The response copy book and the LINKAGE SECTION

The response copy book needs to be included in the LINKAGE SECTION of the COBOL program. This allows the COBOL structures defined in a LINKAGE SECTION to be used as templates for referencing data in the storage used for inter-program communications. Overlaying the templates over the storage area allows the program to use the variables defined in these structure for referencing data in the storage area. Establishing addressability between the structure and the storage is done by using a SET ADDRESS statement to set the address of template structure to the address of the storage area as shown below.

```
251
252          *-----*
253          * Establish addressability to response message
254          *-----*
255          |   SET address of BAQBASE-EMP00P01 to BAQ-RESP-BASE-ADDRESS.
256
```

In the code above, BAQ-RESP-BASE-ADDRESS is a pointer variable that contains the address of the storage where the initial response message content resides.

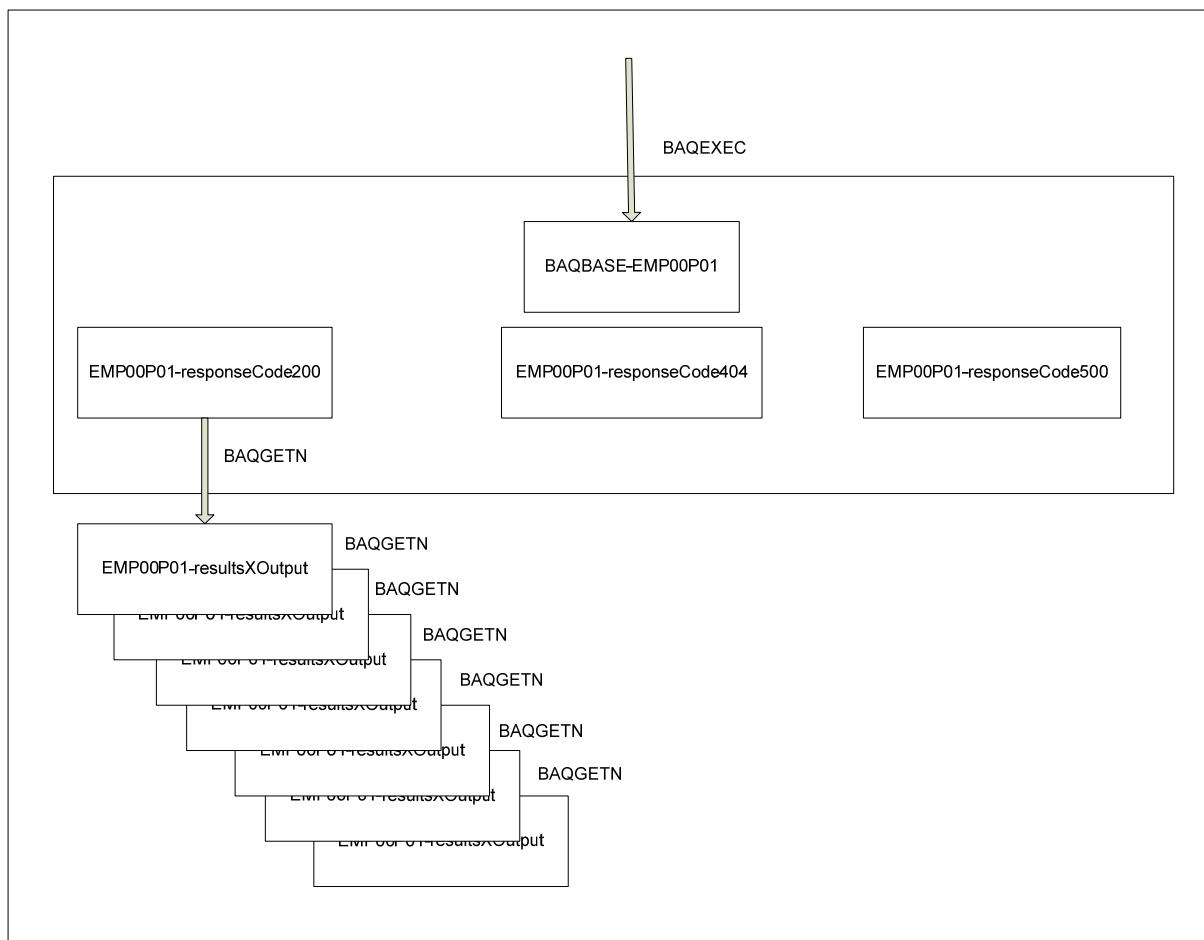
The generated response copy book will begin with a *BAQBASE-copybookName* structure (where the string *copybookName* is the name of the generated copy book, e.g., *EMP00P01* as shown in the example below).

187	188	01 BAQBASE-EMP00P01.	
189			
190	03 responseCode200-existence	PIC S9(9) COMP-5 SYNC.	
191	03 responseCode200-dataarea	PIC X(16).	
192			
193			
194	03 responseCode404-existence	PIC S9(9) COMP-5 SYNC.	
195	03 responseCode404-dataarea	PIC X(16).	
196			
197			
198	03 responseCode500-existence	PIC S9(9) COMP-5 SYNC.	
199	03 responseCode500-dataarea	PIC X(16).	
200			
201			
202	01 EMP00P01-responseCode200.		
203	03 responseCode200.		

In this example, the contents of the storage area represented by the BAQBASE-EMP00P01 structure are populated and a pointer address which was returned when the BAQEXEC host API is called.

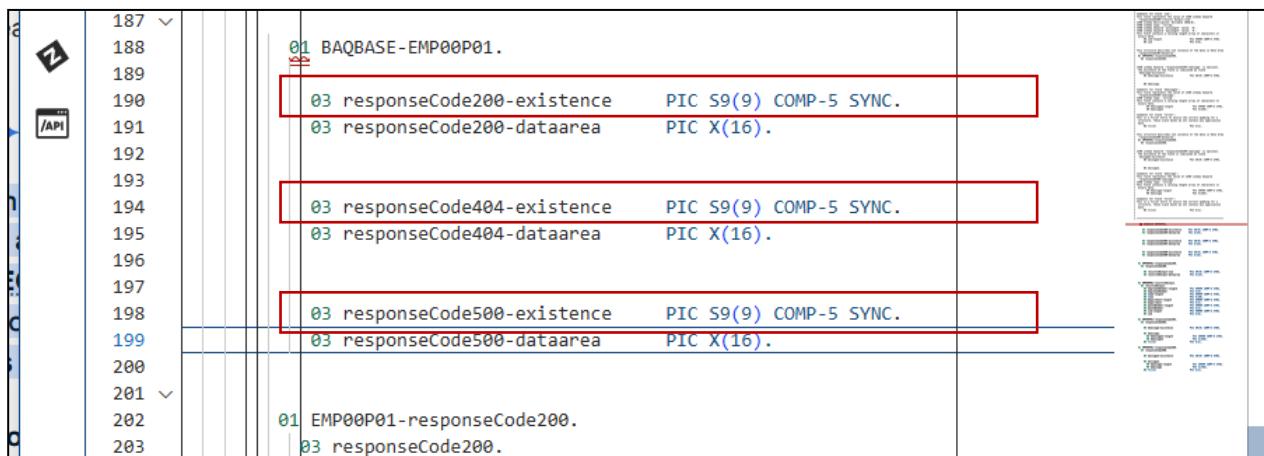
216	*-----
217	* Set up the data for the API Requester call
218	*-----
219	SET BAQ-REQ-BASE-ADDRESS to address of BAQBASE-EMP00Q01.
220	MOVE LENGTH OF BAQBASE-EMP00Q01 TO BAQ-REQ-BASE-LENGTH.
221	SET WS-APIinfoPointer to address of BAQ-API-INFO-EMP00I01
222	
223	CALL BAQ-EXEC-NAME USING
224	BY REFERENCE BAQ-ZCONNECT-AREA
225	BY VALUE WS-APIinfoPointer
226	BY REFERENCE BAQ-REQUEST-AREA
227	BY REFERENCE BAQ-RESPONSE-AREA
228	RETURNING WS-BAQreturnCode.
229	
230	SET address of BAQBASE-EMP00P01 to BAQ-RESP-BASE-ADDRESS.
231	

The diagram below shows the hierarchy or relationship of the structures in the Linkage section. The BAQEXEC requests populates the BAQBASE structures and the 3 response code structures. The application uses the BAQGETN request along with the -num and data area variables in the responseCode200 structure to retrieve the individual resultsXOutput structures.



Determining the HTTP status code using the BAQBASE structure

The names and the number of variables in the BAQBASE-EMP00P01 structure is based on the number and occurrences of the HTTP response codes defined in the specification document. In our case there were three possible HTTP responses, e.g., 202, 404 and 500 so there were three sets of two variables for each of the possible response codes for our initial use.



The screenshot shows the Z/OS API Explorer interface. On the left, there's a navigation tree with nodes like 'API Explorer', 'APIs', 'Data', 'Jobs', 'Logs', 'Metrics', 'Tables', and 'Workspaces'. The main area displays a COBOL source code editor. The code is as follows:

187	188	189	01 BAQBASE-EMP00P01.	
190	03 responseCode200-existence	PIC S9(9) COMP-5 SYNC.		
191	03 responseCode200-dataarea	PIC X(16).		
192				
193				
194	03 responseCode404-existence	PIC S9(9) COMP-5 SYNC.		
195	03 responseCode404-dataarea	PIC X(16).		
196				
197				
198	03 responseCode500-existence	PIC S9(9) COMP-5 SYNC.		
199	03 responseCode500-dataarea	PIC X(16).		
200				
201	01 EMP00P01-responseCode200.			
202	03 responseCode200.			
203				

The code defines a structure named BAQBASE-EMP00P01 with three sets of variables for response codes 200, 404, and 500. Each set contains an 'existence' variable (S9(9) PIC) and a 'dataarea' variable (X(16) PIC). A global variable EMP00P01-responseCode200 is also defined, which points to the 'existence' variable for response code 200.

The variables with the *-existence* suffix(i.e., *responseCode200-existence*) are used by the application to determine which HTTP status code was returned when the API was executed. The client application checks to see if an existence variable associated with a specific HTTP status code has value of zero or 1. If the existence variable of a HTTP status code has a value of 1(or true), then that was the status code that was returned by BAQEXEC when the API was invoked. Otherwise, the value of the existence variable will be zero (or false). The variable with the *-dataarea* suffix(i.e., *responseCode200-dataarea*) provides a key or index that can be used to return or retrieve additional information with a BAQGETN request and we will see how it is used shortly.

Knowing which HTTP status code was returned is not sufficient. The response message for each HTTP code will have a different layouts or mappings. Each of the HTTP status code response message mappings are also present in the response copybook. For this API the different response message mapping structures are shown below, these response structures have the string *responseCode* embedded in their structure name, e.g., *EMP00P01-responseCode-404*. Again, these structures are acting as templates representing the different possible response message layouts.

The screenshot shows the Zowe CLI interface with the following details:

- Title Bar:** Shows the current session is "cbl EMP00P01 1".
- Left Sidebar:** Contains icons for file operations (New, Open, Save, Print, Copy, Paste, Find, Replace, Refresh, Help).
- Central Area:** Displays COBOL source code for the EMP00P01 API. The code defines four response structures based on HTTP status codes:
 - 01 EMP00P01-responseCode200.** This structure is highlighted with a red box. It contains:
 - 03 responseCode200.
 - 06 resultsXOutput-num PIC S9(9) COMP-5 SYNC.
 - 06 resultsXOutput-dataarea PIC X(16).
 - 01 EMP00P01-resultsXOutput.** This structure is defined but not highlighted.
 - 01 EMP00P01-responseCode404.** This structure is highlighted with a red box. It contains:
 - 03 responseCode404.
 - 06 Xmessage-existence PIC S9(9) COMP-5 SYNC.
 - 06 Xmessage.
 - 09 Xmessage2-length PIC S9999 COMP-5 SYNC.
 - 09 Xmessage2 PIC X(256).
 - 06 filler PIC X(2).
 - 01 EMP00P01-responseCode500.** This structure is highlighted with a red box. It contains:
 - 03 responseCode500.
 - 06 message2-existence PIC S9(9) COMP-5 SYNC.
 - 06 message2.
 - 09 Xmessage-length PIC S9999 COMP-5 SYNC.
 - 09 Xmessage PIC X(256).
 - 06 filler PIC X(2).
- Bottom Status Bar:** Shows "No connection", line number "Ln 199, Col 35", spaces "Spaces: 2", encoding "UTF-8", CRLF, COBOL mode, and a note "No Zowe CLI profile found".

The application can determine which HTTP status was returned using the *-existence* variables. And now using the data area variable and information about the length of the response message, the application can call the BAQGETN request to retrieve the message specific to the HTTP status code and establish addressability to the response message storage area as shown below in an example below for an HTTP 404 response. Line 260 checks to see if the HTTP response was a 404 or not. Line 261 through 264 uses the index data area for retrieving the 404-response message and the length of the response message. Line 265 calls the BAQGETN API. Lines 266 through 267 establishes addressability to the 404-response message and sets up the COBOL mapping for this response.

```

257
258      * Check for an HTTP 404 response
259      *
260      IF responseCode404-existence > 0 THEN
261          MOVE responseCode404-dataarea of BAQBASE-EMP00P01 TO
262              | WS-dataAreaName
263          MOVE LENGTH OF EMP00P01-responseCode404 TO
264              | WS-elementLength
265          PERFORM BAQGETN
266          SET address of EMP00P01-responseCode404
267              | to WS-elementPointer
268          STRING
269              | Xmessage2 of EMP00P01-responseCode404
270              | DELIMITED BY SIZE
271              | INTO WS-displayMessage
272          PERFORM WRITE-RESPONSE-MSG.
273          EXIT.

```



Handling Multiple occurrences (arrays)

Using these *existence* and *dataarea* variables, the coding is simple and straight-forward for the 404 and 500 HTTP response. Handling a HTTP 200 is little more complex. Notice that the *EMP00P01-responseCode200* structure has embedded data area (*resultsXOutput-dataarea*) and a variable with a *-num* suffix. This means that when HTTP 200 is returned for the HTTP status code, the response will contain an array of multiple elements.

```

201
202      01 EMP00P01-responseCode200.
203          03 responseCode200.
204
205          06 resultsXOutput-num      PIC S9(9) COMP-5 SYNC.
206          06 resultsXOutput-dataarea  PIC X(16).
207
208
209      01 EMP00P01-resultsXOutput.
210          03 resultsXOutput.
211              06 employeeNumber-length  PIC S9999 COMP-5 SYNC.
212              06 employeeNumber        PIC X(6).
213              06 name-length           PIC S9999 COMP-5 SYNC.
214              06 name                 PIC X(30).
215              06 department-length     PIC S9999 COMP-5 SYNC.
216              06 department            PIC X(3).
217              06 phoneNumber-length    PIC S9999 COMP-5 SYNC.
218              06 phoneNumber           PIC X(4).
219              06 job-length            PIC S9999 COMP-5 SYNC.
220              06 job                  PIC X(8).
221

```



And each element in the array must be mapped individually to *EMP00P01-resultsXOutput* using a BAQGETN API call. So, when an HTTP 200 is returned with an array of elements, there will be one BAQGETN invoked to obtain addressability to the *EMP00P01-responseCode200* storage area and *resultsXOutput-num* requests for BAQGETN to obtain addressability to each element in the array.

In the code below,

- Line 25 checks to see if an HTTP 200 status code was returned.
- Lines 26 through 29 sets up for the first BAGETN request to establish addressability for structure *EMP00P01-responseCode200*. This structure has the number of entries returned and the initial data area index. Lines 31 through 36 set up the parameters for the BAGETN request to retrieve the first element in the array of returned items.
- Lines 37 through 40 is loop for processing each element in the array until all have been processed.

```

1      LINKAGE SECTION.
2
3      01 BAQBASE-EMP00P01.
4          03 responseCode200-existence      PIC S9(9) COMP-5 SYNC.
5          03 responseCode200-dataarea      PIC X(16).
6
7      01 EMP00P01-responseCode200.
8          03 responseCode200.
9              06 resultsXOutput-num      PIC S9(9) COMP-5 SYNC.
10             06 resultsXOutput-dataarea  PIC X(16).
11
12     01 EMP00P01-resultsXOutput.
13         03 resultsXOutput.
14             06 employeeNumber-length    PIC S9999 COMP-5 SYNC.
15             06 employeeNumber        PIC X(6).
16             06 name-length          PIC S9999 COMP-5 SYNC.
17             06 name                PIC X(30).
18             06 department-length     PIC S9999 COMP-5 SYNC.
19             06 department           PIC X(3).
20             06 phoneNumber-length   PIC S9999 COMP-5 SYNC.
21             06 phoneNumber         PIC X(4).
22             06 job-length          PIC S9999 COMP-5 SYNC.
23             06 job                 PIC X(8).
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

```

- Lines 45 through 47 establishes addressability to an entry in the array.
- Lines 48 through 56 use the template variables to access specific data in the entry.
- Line 59 through 71 used the BAQGETN request to process the next element in the array, establish addressability to the storage where details of the elements and uses the mapped variables to access the element's information.

```

43      GET-EACH-EMPLOYEE SECTION.
44          |    PERFORM BAQGETN
45          |    SET address of EMP00P01-resultsXOutput
46          |    |    to WS-elementPointer
47          |    ADD 1 to resultsIndex
48          |    MOVE employeeNumber of
49          |    |    resultsXOutput(1:employeeNumber-length)
50          |    |    TO employeeNumber OF RESULTS(resultsIndex)
51          |    MOVE name of
52          |    |    resultsXOutput(1:name-length)
53          |    |    TO employeeName OF RESULTS(resultsIndex)
54          |    MOVE phoneNumber of
55          |    |    resultsXOutput(1:phoneNumber-length)
56          |    |    TO employeePhone OF RESULTS(resultsIndex)
57          |    EXIT.
58
59      BAQGETN SECTION.
60          |    CALL BAQ-GETN-NAME USING
61          |    |    BY REFERENCE BAQ-ZCONNECT-AREA
62          |    |    WS-dataAreaName
63          |    |    BY REFERENCE WS-elementPointer
64          |    |    BY REFERENCE WS-elementLength
65          |    |    RETURNING WS-BAQreturnCode.
66          |    MOVE BAQ-ZCON-COMPLETION-CODE TO WS-completionCode.
67          |    MOVE BAQ-ZCON-REASON-CODE TO WS-reasonCode.
68          |    IF NOT BAQ-SUCCESS THEN
69          |    |    MOVE FAILED TO WS-returnCode
70          |    END-IF.
71          |    EXIT.

```

The purpose of this section was to show how the response copy book needs to be reviewed and understood before developing the COBOL request client application.

Deploying the API Requester WAR file

Next the API requester web application archive (WAR) was deployed and made available to the z/OS Connect server.

Creating the API requester WAR and generating the associated COBOL copy books is done by running the Gradle build process. This build process is documented in section *Generating the artifacts for an API requester* at URL <https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=30-generating-artifacts-api-requester>. These instructions are fine but one of our goals was to do the build on z/OS. The details of our process are described in the *Gradle on z/OS* section of this document on page 40.

Server XML configuration Updates

We updated the z/OS Connect server where the API requester will be installed with these configuration elements. The `<webApplication/>` configuration element defined the API requester application and provided the directory where the WAR file is located. An override is provided for the `connectionRef` attribute is provided just to show how it can be done.

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

    <webApplication id="Employee Roster API application"
        location="${server.config.dir}/apps/roster.war"
        name="roster">
        <appProperties>
            <property name="connectionRef" value="DB2SSID"/>
        </appProperties>
    </webApplication>

    <!-- The location of the API being called -->
    <zosconnect_endpointConnection id="DB2SSID"
        host="http://wg31.washington.ibm.com"
        domainBasePath="/roster"
        port="9082"
        authenticationConfigRef="zosconnectBasicAuthConfig"/>

    <zosconnect_authData id="zosconnectBasicAuthConfig"
        password="USER1"
        user="user1"/>

</server>
```

Make the WAR file available to the server

The Gradle generated WAR file needs to be placed in the directory identified by the *location* attribute. We used two techniques to do this. The first was simply to copy the WAR file into a directory known by the z/O Connect API requester server. The second method was to use a Liberty managed bean (MBean) and the *curl* command to install the WAR file into the directory. The advantage of using the MBean with curl is that this technique is that it can be used regardless of the platform on which the WAR file was created, e.g., Windows, z/OS, Linux, etc.

Using the OMVS copy (cp) command

A simple method was to copy the API requester WAR file to the API requester server's application directory using OMVS copy(cp) command.

```
*****  
/* SET SYMBOLS  
*****  
//EXPORT EXPORT SYMLIST=(*)  
// SET WARDIR='/u/johnson/gradle/roster/build/libs'  
// SET WARFILE='roster.war'  
// SET WLPUSER='/var/ats/zosconnect'  
// SET SERVER='OAS3ApiRequester'  
//COPY EXEC PGM=IKJEFT01,REGION=0M  
//SYSTSPRT DD SYSOUT=*  
//SYSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
//SYSTSIN DD *,SYMBOLS=EXECSYS  
BPXBATCH SH +  
export WARDIR=&WARDIR; +  
export WARFILE=&WARFILE; +  
export WLPUSER=&WLPUSER; +  
export SERVER=&SERVER; +  
cp $WARDIR/$WARFILE $WLPUSER/servers/$SERVER/apps
```

Using the curl command

Another option we used was the cURL command to ‘upload’ the WAR file to the application directory using a Liberty provided Manage Bean (MBean).

```
//*****
//** SET SYMBOLS
//*****
//EXPORT EXPORT SYMLIST=(*)
// SET WARDIR='/u/johnson/gradle/roster/build/libs'
// SET WARFILE='roster.war'
// SET WLPUSER='/var/ats/zosconnect'
// SET SERVER='OAS3ApiRequester'
//COPY EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//SYSERR DD SYSOUT=*
//STDOUT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH +
export WARDIR=&WARDIR; +
export WARFILE=&WARFILE; +
export WLPUSER=&WLPUSER; +
export SERVER=&SERVER; +
curl -X POST --user user1:user1 -insecure +
--header "Content-Type: application/zip" +
--data-binary @$WARDIR/$WARFILE -v -w " -HTTP CODE: ${http_code}" +
https://wg31.washington.ibm.com:9465/IBMJMXConnectorREST/file/+
$WLPUSER/servers/$SERVER/apps/$WARFILE
```

The *curl* command used in the JCL above was provided by the installation of the IBM Open Enterprise Foundation for z/OS product, for more information see URL <https://www.ibm.com/docs/en/oefzos>

Security Updates

The z/OS Connect server where the API requester will be installed had SAF security enabled which meant that an EJBRole SAF resource had to be defined and permission given to any identity who needed to access the API requester. This was done by using the RACF RDEFINE commands below:

- Define the *EJBRole* resource

```
rdefine ejbrole BBGZDFLT.roster.invoke uacc(read)
setropts raclist(ejbrole) refresh
```

CICS Updates

We added z/OS Connect API requester support to the CICS region by performing these tasks.

1. The DFHCSDUP utility was used to add the CICS resource definitions in member BAQHCSD in data set SBAQSAMP to the CICS region's CSD file. The group BAQHAPI was then added to a group list entry in the CICS region's GRPLIST SIT parameter and then the group was installed using the CEDA transaction.
2. The z/OS Connect SBAQLIB1 SMP/E target data set was added to the DFHRPL DD list in the region's startup JCL.
3. The API HOST API by default uses the BAQHZCON resource defined to CICS to locate the z/OS Connect server where the request will be sent. The BAQHZCON URIMAP entry installed in Step 1 was modified to our local values for host and port. The Path was changed to be just a single slash, see below.

The screenshot shows a CICS terminal window titled "WG31". The window has a menu bar with File, Edit, Settings, View, Communication, Actions, Window, Help. The main display area shows the following CEDA command and its output:

```
OVERTYPE TO MODIFY                               CICS RELEASE = 0740
CEDA  ALter UriMap( BAQHZCON )
      UriMap       : BAQHZCON
      Group        : BAQHAPI
      Description  ==> Default URIMAP for z/OS Connect server
      Status       ==> Enabled          Enabled | Disabled
      Usage        ==> Client           Server | Client | Pipeline | Atom
                           | Jvmserver
UNIVERSAL RESOURCE IDENTIFIER
      Scheme       ==> HTTP            HTTP | HTTPS | Iiop | Jms
      Port         ==> 09083          No | 1-65535
      Host         ==> wg31.washington.ibm.com
      Path         ==>
      (Mixed Case) ==>
      ==>
      ==>
      ==>
+ OUTBOUND CONNECTION POOLING
                                         SYSID=CICS APPLID=CICSS53Z
                                         DSN=CICSTS.CICSS61Z.DFHCSD
                                         6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
PF 1 HELP 2 COM 3 END
```

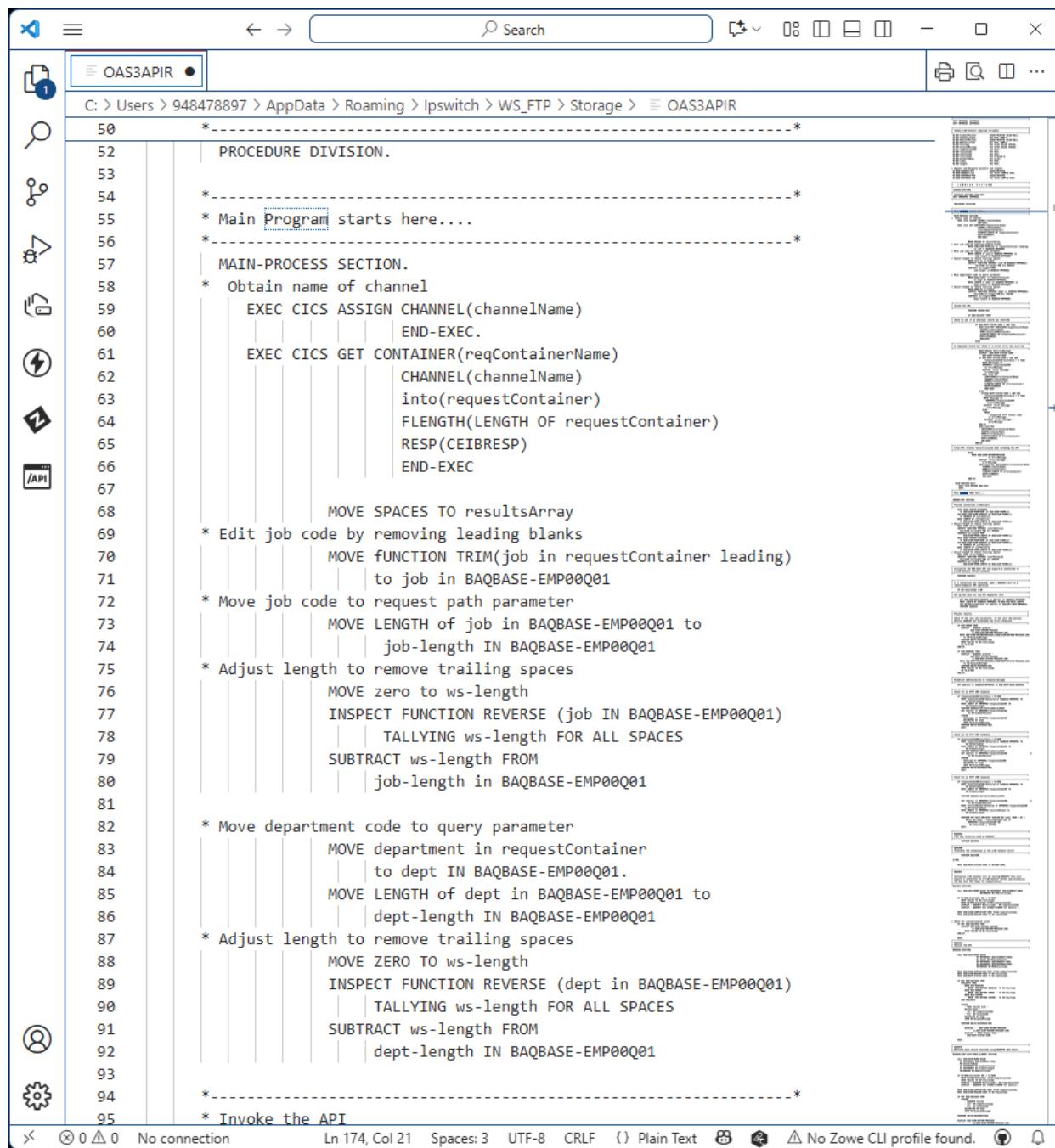
The status line at the bottom indicates: Connected to remote server/host wg31z using lu/pool TCP00143 and port 23.

The CICS API requester client application

Let us review the CICS application as it was coded to use the z/OS Connect Host APIs.

MAIN-PROCESS SECTION

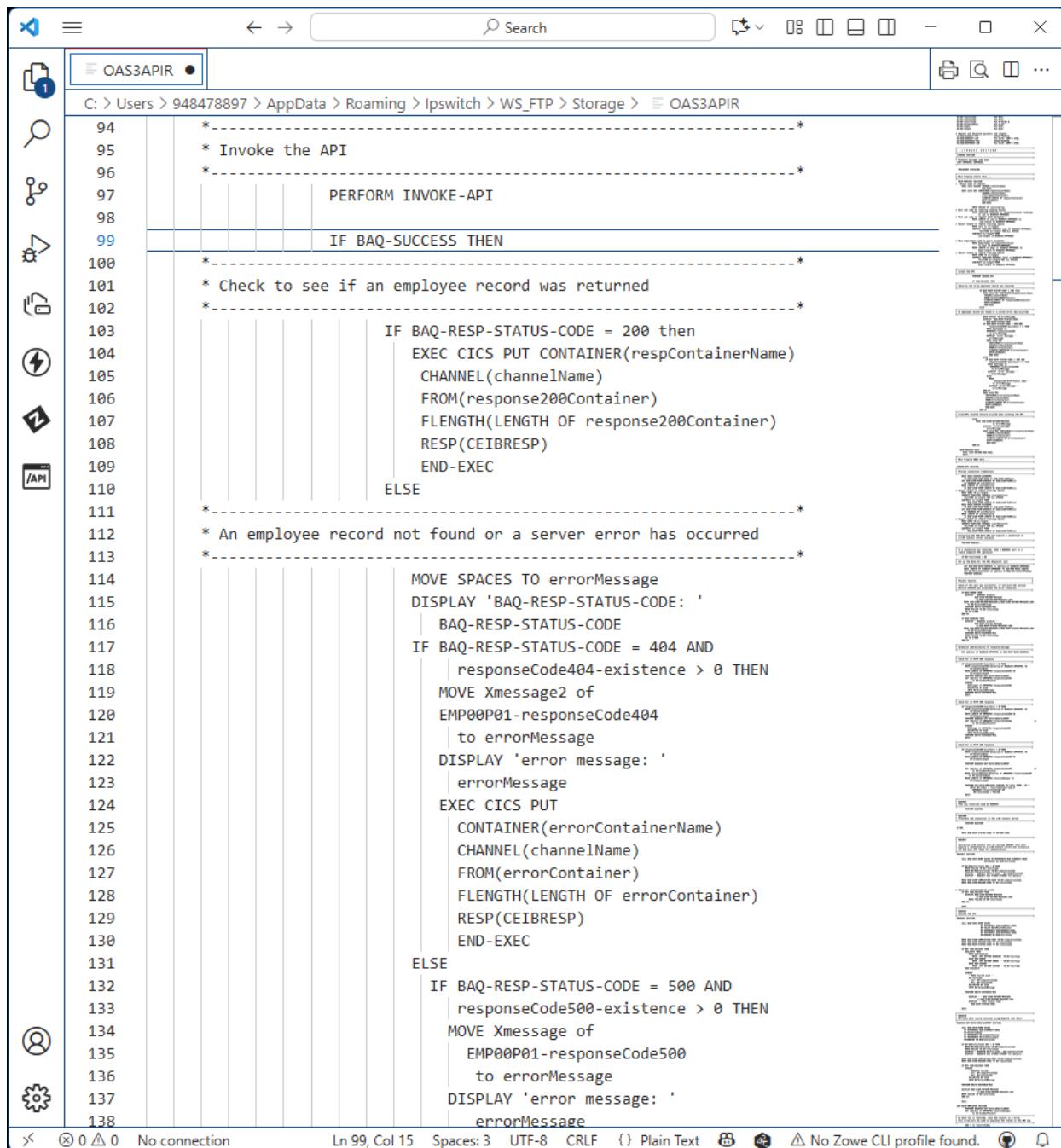
In the MAIN-PROCESS section of the application, the request container is retrieved from the channel (line 61). The request container contents are moved to the variables in API request message. Some minor editing of the contents of the fields is done and the length of each field is calculated and stored in the API request message (lines 69-92).



The screenshot shows a code editor window with the title bar "OAS3APIR". The file path is "C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR". The code is written in COBOL and is divided into several sections:

- Line 50:** PROCEDURE DIVISION.
- Line 52:** *-----
- Line 54:** * Main Program starts here....
- Line 55:** *-----
- Line 56:** MAIN-PROCESS SECTION.
- Line 57:** * Obtain name of channel
- Line 58:** EXEC CICS ASSIGN CHANNEL(channelName)
- Line 59:** | END-EXEC.
- Line 60:** EXEC CICS GET CONTAINER(reqContainerName)
- Line 61:** | CHANNEL(channelName)
- Line 62:** | into(requestContainer)
- Line 63:** | LENGTH(LENGTH OF requestContainer)
- Line 64:** | RESP(CEIBRESP)
- Line 65:** | END-EXEC
- Line 66:**
- Line 67:**
- Line 68:** MOVE SPACES TO resultsArray
- Line 69:** * Edit job code by removing leading blanks
- Line 70:** MOVE FUNCTION TRIM(job in requestContainer leading)
- Line 71:** | to job in BAQBASE-EMP00Q01
- Line 72:** * Move job code to request path parameter
- Line 73:** MOVE LENGTH of job in BAQBASE-EMP00Q01 to
- Line 74:** | job-length IN BAQBASE-EMP00Q01
- Line 75:** * Adjust length to remove trailing spaces
- Line 76:** MOVE zero to ws-length
- Line 77:** INSPECT FUNCTION REVERSE (job IN BAQBASE-EMP00Q01)
- Line 78:** | TALLYING ws-length FOR ALL SPACES
- Line 79:** SUBTRACT ws-length FROM
- Line 80:** | job-length in BAQBASE-EMP00Q01
- Line 81:**
- Line 82:** * Move department code to query parameter
- Line 83:** MOVE department in requestContainer
- Line 84:** | to dept IN BAQBASE-EMP00Q01.
- Line 85:** MOVE LENGTH of dept in BAQBASE-EMP00Q01 to
- Line 86:** | dept-length IN BAQBASE-EMP00Q01
- Line 87:** * Adjust length to remove trailing spaces
- Line 88:** MOVE ZERO TO ws-length
- Line 89:** INSPECT FUNCTION REVERSE (dept in BAQBASE-EMP00Q01)
- Line 90:** | TALLYING ws-length FOR ALL SPACES
- Line 91:** SUBTRACT ws-length FROM
- Line 92:** | dept-length IN BAQBASE-EMP00Q01
- Line 93:**
- Line 94:** *-----
- Line 95:** * Invoke the API

The execution of the Host APIs is performed in procedure INVOKE-API. Upon return, the results are checked for a HTTP 200 status code (line 103). If true, the results are placed in an “200” response container (line 104) and the container is returned in the channel. Otherwise, a check is made for a HTTP 404 (line 117) or a HTTP 500 (line 132) status codes. For either, the corresponding message from the status code’s response area is moved to a message variable in an error container and the container is returned in the channel (lines 124 and 146).



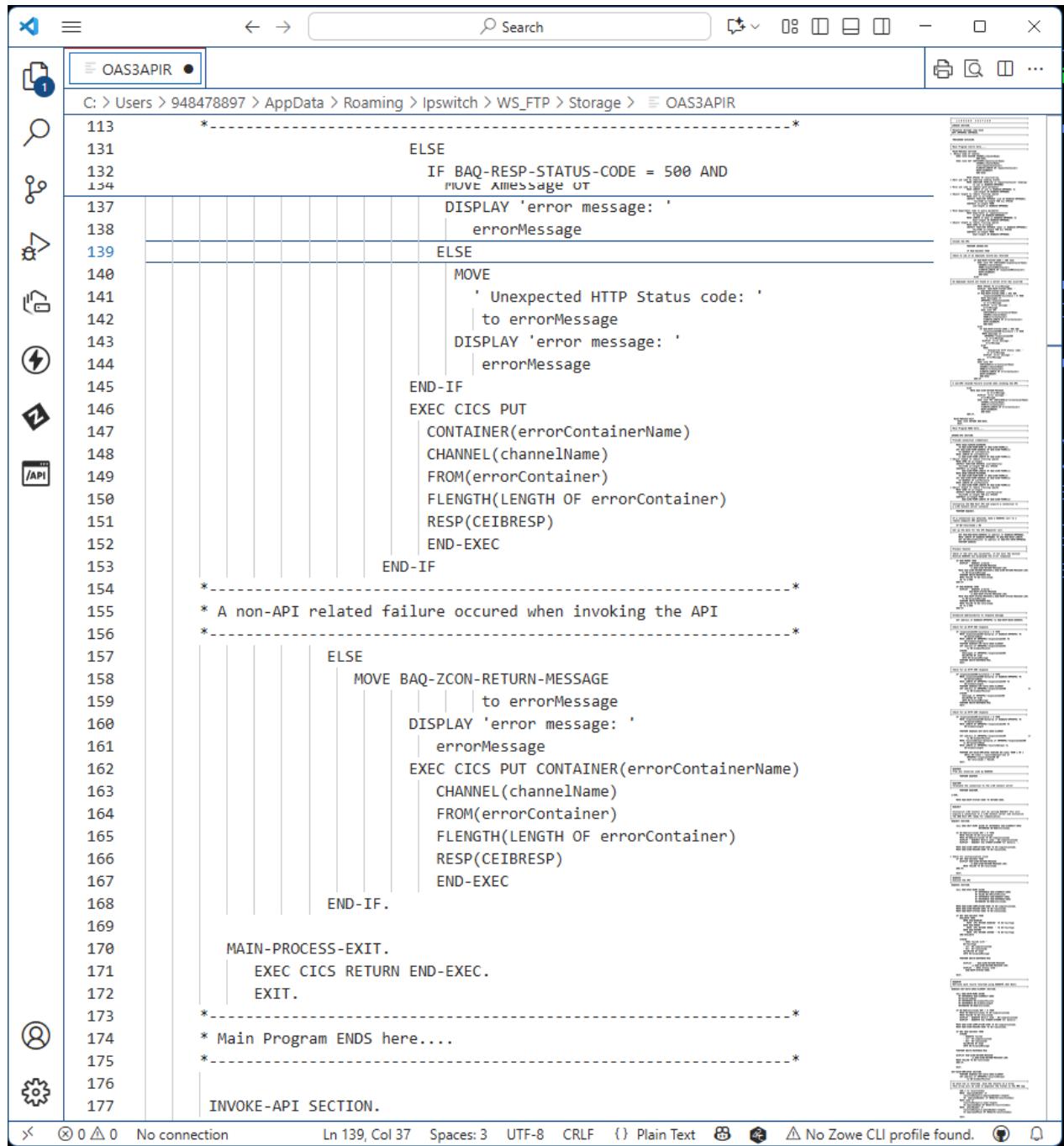
```

OAS3APIR
C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR

94      *-----*
95      * Invoke the API
96      *-----*
97          PERFORM INVOKE-API
98
99      IF BAQ-SUCCESS THEN
100     *-----*
101     * Check to see if an employee record was returned
102     *-----*
103     IF BAQ-RESP-STATUS-CODE = 200 then
104         EXEC CICS PUT CONTAINER(respContainerName)
105             CHANNEL(channelName)
106                 FROM(response200Container)
107                     FLENGTH(LENGTH OF response200Container)
108                         RESP(CEIBRESP)
109                             END-EXEC
110
111     ELSE
112     * An employee record not found or a server error has occurred
113     *-----*
114         MOVE SPACES TO errorMessage
115         DISPLAY 'BAQ-RESP-STATUS-CODE: '
116             BAQ-RESP-STATUS-CODE
117             IF BAQ-RESP-STATUS-CODE = 404 AND
118                 | responseCode404-existence > 0 THEN
119                     MOVE Xmessage2 of
120                         EMP00P01-responseCode404
121                             to errorMessage
122                             DISPLAY 'error message: '
123                             errorMessage
124                             EXEC CICS PUT
125                                 CONTAINER(errorContainerName)
126                                     CHANNEL(channelName)
127                                         FROM(errorContainer)
128                                             FLENGTH(LENGTH OF errorContainer)
129                                                 RESP(CEIBRESP)
130                                                     END-EXEC
131
132             ELSE
133                 IF BAQ-RESP-STATUS-CODE = 500 AND
134                     | responseCode500-existence > 0 THEN
135                         MOVE Xmessage of
136                             EMP00P01-responseCode500
137                                 to errorMessage
138                                 DISPLAY 'error message: '
139                                 errorMessage

```

Finally, checks for errors that are not caused by invoking the z/OS Connect API requester and are returned to the caller with as much information as possible (line 162).



The screenshot shows a terminal window titled "OAS3APIR" displaying a CICS API requester program. The code handles errors from the API requester and returns them to the caller. It includes logic for handling 500 errors, unexpected status codes, and non-API related failures. The code uses MOVE, DISPLAY, EXEC CICS PUT, and END-IF statements. The terminal window has a sidebar with various icons and a status bar at the bottom.

```
113      *-----*
131          ELSE
132              IF BAQ-RESP-STATUS-CODE = 500 AND
134                  MOVE AMESSAGE TO
137                  DISPLAY 'error message: '
138                      errorMessage
139          ELSE
140              MOVE
141                  ' Unexpected HTTP Status code: '
142                  to errorMessage
143                  DISPLAY 'error message: '
144                      errorMessage
145          END-IF
146          EXEC CICS PUT
147              CONTAINER(errorContainerName)
148              CHANNEL(channelName)
149              FROM(errorContainer)
150              FLENGTH(LENGTH OF errorContainer)
151              RESP(CEIBRESP)
152          END-EXEC
153      END-IF
154      *-----*
155      * A non-API related failure occurred when invoking the API
156      *-----*
157          ELSE
158              MOVE BAQ-ZCON-RETURN-MESSAGE
159                  to errorMessage
160                  DISPLAY 'error message: '
161                      errorMessage
162              EXEC CICS PUT CONTAINER(errorContainerName)
163                  CHANNEL(channelName)
164                  FROM(errorContainer)
165                  FLENGTH(LENGTH OF errorContainer)
166                  RESP(CEIBRESP)
167              END-EXEC
168          END-IF.
169
170      MAIN-PROCESS-EXIT.
171          EXEC CICS RETURN END-EXEC.
172          EXIT.
173      *-----*
174      * Main Program ENDS here....
175      *-----*
176
177      INVOKE-API SECTION.
```

Invoke-API Section

For clarity, each of the HOST APIs were coded in separate procedure sections and each section is driven by PERFORM statements in the Invoke-API section. The Invoke-API section starts by providing the connection properties to be used to establish the connection to the z/OS Connect server. In this application the identity(BAQZ-SERVER-USERNAME) (lines 181-192) and password(BAQZ-SERVER-PASSWORD) (lines 193-204) are provided. Other connection properties that can be provided are BAQZ-SERVER-URIMAP for CICS clients and BAQZ-SERVER-HOST and BAQZ-SERVER-PORT for other non-CICS clients. The full set of available properties are provided in the product's SBAQCOB data sets.

The screenshot shows a code editor window with the title bar "OAS3APIR". The left sidebar contains various icons for file operations like Open, Save, Find, and Print. The main pane displays a COBOL program listing:

```
175      *-----*
176
177     (INVOKE-API SECTION.
178      *-----*
179      * Provide connection credentials
180      *-----*
181      | MOVE BAQZ-SERVER-USERNAME
182      | | TO BAQ-ZCON-PARM-NAME of BAQ-ZCON-PARMS(1)
183      | SET BAQ-ZCON-PARM-ADDRESS OF BAQ-ZCON-PARMS(1)
184      | | TO ADDRESS OF userIdentity
185      | MOVE LENGTH OF userIdentity
186      | | to BAQ-ZCON-PARM-LENGTH OF BAQ-ZCON-PARMS(1)
187      * Adjust length to remove trailing spaces
188      | MOVE ZERO TO ws-length
189      | INSPECT FUNCTION REVERSE (userIdentity)
190      | | TALLYING ws-length FOR ALL SPACES
191      | SUBTRACT ws-length FROM
192      | | BAQ-ZCON-PARM-LENGTH OF BAQ-ZCON-PARMS(1)
193      MOVE BAQZ-SERVER-PASSWORD
194      | TO BAQ-ZCON-PARM-NAME of BAQ-ZCON-PARMS(2)
195      SET BAQ-ZCON-PARM-ADDRESS OF BAQ-ZCON-PARMS(2)
196      | TO ADDRESS OF userPassword
197      MOVE LENGTH OF userPassword
198      | to BAQ-ZCON-PARM-LENGTH OF BAQ-ZCON-PARMS(2)
199      * Adjust length to remove trailing spaces
200      | MOVE ZERO TO ws-length
201      | INSPECT FUNCTION REVERSE (userPassword)
202      | | TALLYING ws-length FOR ALL SPACES
203      | SUBTRACT ws-length FROM
204      | | BAQ-ZCON-PARM-LENGTH OF BAQ-ZCON-PARMS(2)
205      *-----*
206      * Initialize the BAQ Host API and acquire a connection to
207      * a z/OS Connect server instance
208      *-----*
209      | | PERFORM BAQINIT.
210
211      *-----*
212      * If a connection was obtained, make a BAQEXEC call to a
213      * remote endpoint API operation
214      *-----*
215      | | IF WS-returnCode = OK
216      *-----*
217      * Set up the data for the API Requester call
218      *-----*
219      | | SET BAQ-REQ-BASE-ADDRESS to address of BAQBASIC-EMP00001.
```

At the bottom of the editor, there are status indicators: "No connection", "Ln 139, Col 37", "Spaces: 3", "UTF-8", "CRLF", "Plain Text", and "No Zowe CLI profile found".

Once the connection properties are set, perform the BAQINIT section (line 209) and upon return, check the results (line 215). If no issues were encountered connecting to the server, set up the parameter for invoking the BAQEXEC API (lined 219-221) and then perform the BAQEXEC section and start checking its results (lines 230-248).

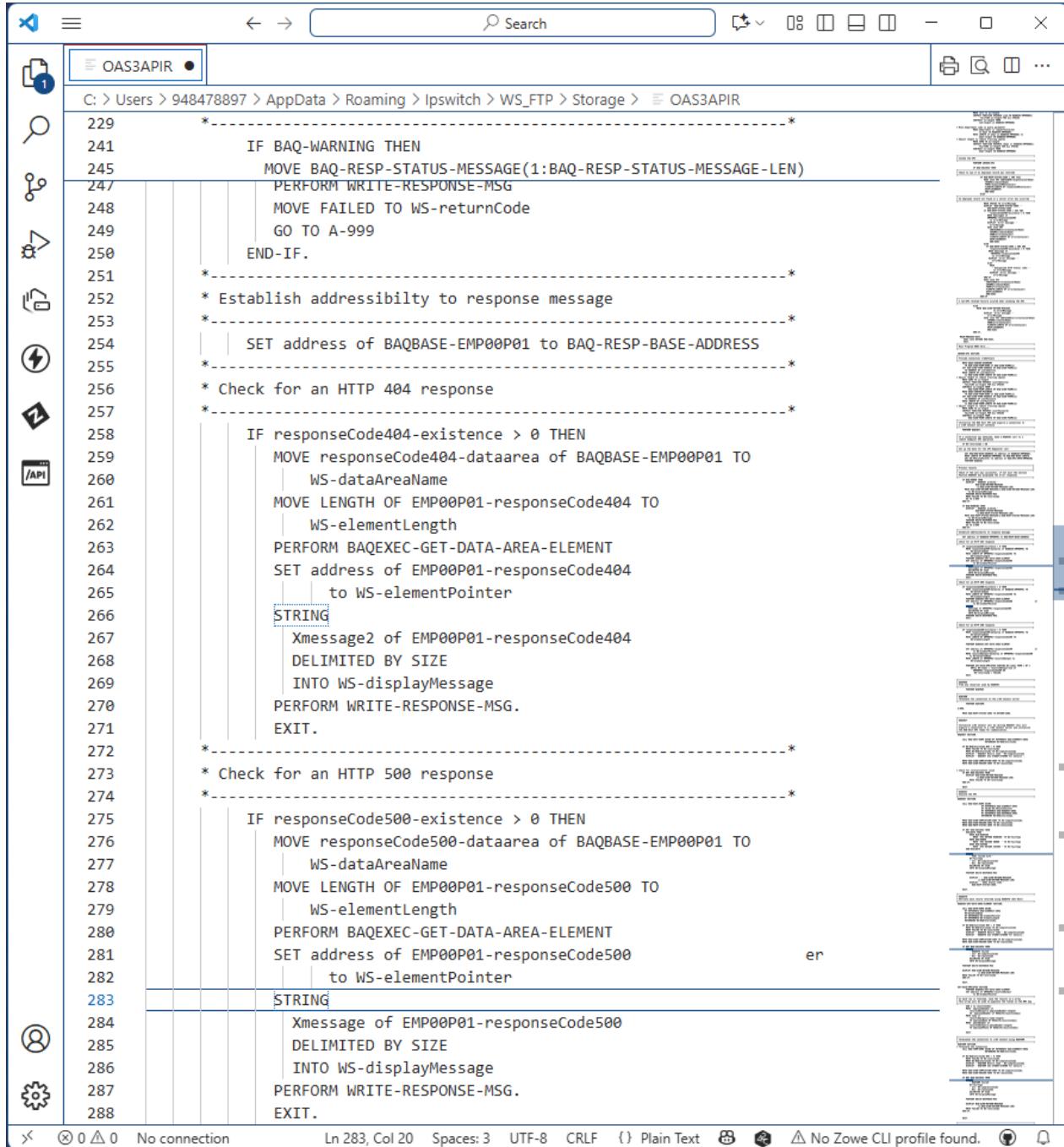
```

C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR
199     Adjust length to remove trailing spaces

205     *-----*
206     * Initialize the BAQ Host API and acquire a connection to
207     * a z/OS Connect server instance
208     *-----*
209     | PERFORM BAQINIT.
210
211     *-----*
212     * If a connection was obtained, make a BAQEXEC call to a
213     * remote endpoint API operation
214     *-----*
215     | IF WS-returnCode = OK
216     *-----*
217     * Set up the data for the API Requester call
218     *-----*
219     | SET BAQ-REQ-BASE-ADDRESS to address of BAQBASE-EMP00Q01.
220     MOVE LENGTH OF BAQBASE-EMP00Q01 TO BAQ-REQ-BASE-LENGTH.
221     SET WS-APIInfoPointer to address of BAQ-API-INFO-EMP00I01
222     PERFORM BAQEXEC
223
224     *-----*
225     * Process results
226     *-----*
227     * Check if the call was successful, if not exit the section
228     * Routine BAQEXEC has displayed the error responses
229     *-----*
230     | IF BAQ-ERROR THEN
231         DISPLAY ' BAQEXEC problem '
232         | BAQ-ZCON-RETURN-MESSAGE
233         | (1:BAQ-ZCON-RETURN-MESSAGE-LEN)
234         MOVE BAQ-ZCON-RETURN-MESSAGE(1:BAQ-ZCON-RETURN-MESSAGE-LEN)
235         | To WS-displayMessage
236         PERFORM WRITE-RESPONSE-MSG
237         MOVE FAILED TO WS-returnCode
238         GO TO A-999
239     END-IF.
240
241     IF BAQ-WARNING THEN
242         DISPLAY ' BAQEXEC problem '
243         | BAQ-RESP-STATUS-MESSAGE
244         | (1:BAQ-RESP-STATUS-MESSAGE-LEN)
245         MOVE BAQ-RESP-STATUS-MESSAGE(1:BAQ-RESP-STATUS-MESSAGE-LEN)
246         | To WS-displayMessage
247         PERFORM WRITE-RESPONSE-MSG
248         MOVE FAILED TO WS-returnCode

```

If the API was successfully executed, establish addressability from the linkage section structure in the COBOL application to the address where the initial results are located (line 254). Access the linkage area storage and see if an HTTP status code 404 (line 258) or an HTTP status code 500 (line 275) was returned. If either occurred, access the details of the results (line 259 or 276) and return to the main section.



```

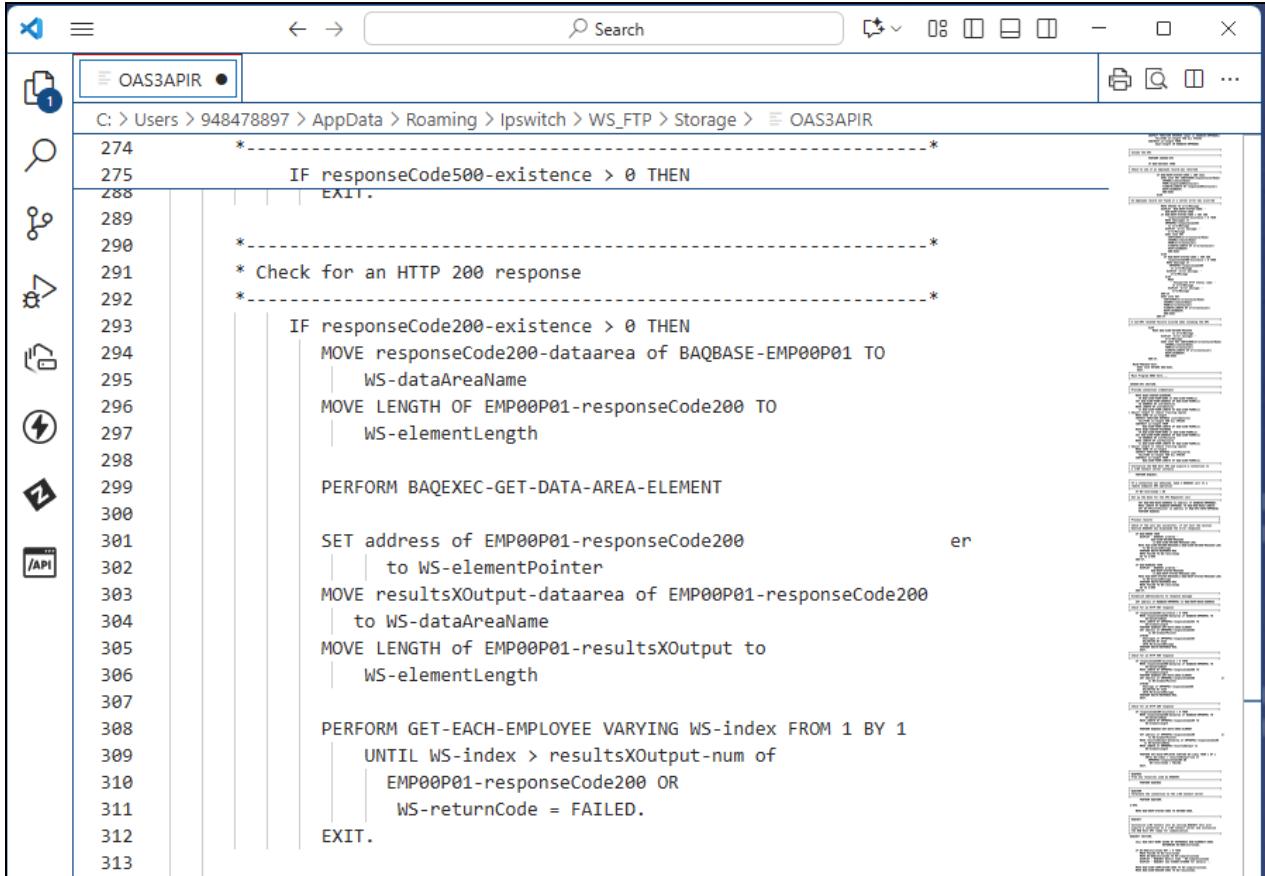
C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR
229      *
241      IF BAQ-WARNING THEN
245      MOVE BAQ-RESP-STATUS-MESSAGE(1:BAQ-RESP-STATUS-MESSAGE-LEN)
247      PERFORM WRITE-RESPONSE-MSG
248      MOVE FAILED TO WS-returnCode
249      GO TO A-999
250      END-IF.
251      *
252      * Establish addressability to response message
253      *
254      | | SET address of BAQBASE-EMP00P01 to BAQ-RESP-BASE-ADDRESS
255      *
256      * Check for an HTTP 404 response
257      *
258      | | IF responseCode404-existence > 0 THEN
259      | | MOVE responseCode404-dataarea of BAQBASE-EMP00P01 TO
260      | |   WS-dataAreaName
261      | | MOVE LENGTH OF EMP00P01-responseCode404 TO
262      | |   WS-elementLength
263      | | PERFORM BAQEXEC-GET-DATA-AREA-ELEMENT
264      | | SET address of EMP00P01-responseCode404
265      | |   to WS-elementPointer
266      | | STRING
267      | |   Xmessage2 of EMP00P01-responseCode404
268      | |   DELIMITED BY SIZE
269      | |   INTO WS-displayMessage
270      | | PERFORM WRITE-RESPONSE-MSG.
271      | | EXIT.
272      *
273      * Check for an HTTP 500 response
274      *
275      | | IF responseCode500-existence > 0 THEN
276      | | MOVE responseCode500-dataarea of BAQBASE-EMP00P01 TO
277      | |   WS-dataAreaName
278      | | MOVE LENGTH OF EMP00P01-responseCode500 TO
279      | |   WS-elementLength
280      | | PERFORM BAQEXEC-GET-DATA-AREA-ELEMENT
281      | | SET address of EMP00P01-responseCode500
282      | |   to WS-elementPointer
283      | | STRING
284      | |   Xmessage of EMP00P01-responseCode500
285      | |   DELIMITED BY SIZE
286      | |   INTO WS-displayMessage
287      | | PERFORM WRITE-RESPONSE-MSG.
288      | | EXIT.

```

The terminal window shows the COBOL source code for the OAS3APIR application. The code handles responses from a CICS API request, checking for 404 and 500 errors and displaying messages. The code uses various COBOL constructs like IF, MOVE, PERFORM, and STRING. The terminal interface includes a sidebar with icons for file operations, a search bar at the top, and status information at the bottom.

The INVOKIE-API continues by checking for a HTTP 200 response (line 293). If the response were a HTTP 200, an array with multiple entries could have been returned. First establish addressability to the HTTP 200 response area (line 294-297). Invoke a BAQGETN request in procedure BAQEXEC-GETUse the 200-response area to obtain addressability to the array (line 301-306).

Iterate for each item in the array performing section GET-EACH-EMPLOYEE starting with item 1 until the number of items have been processed(lines 308-311).

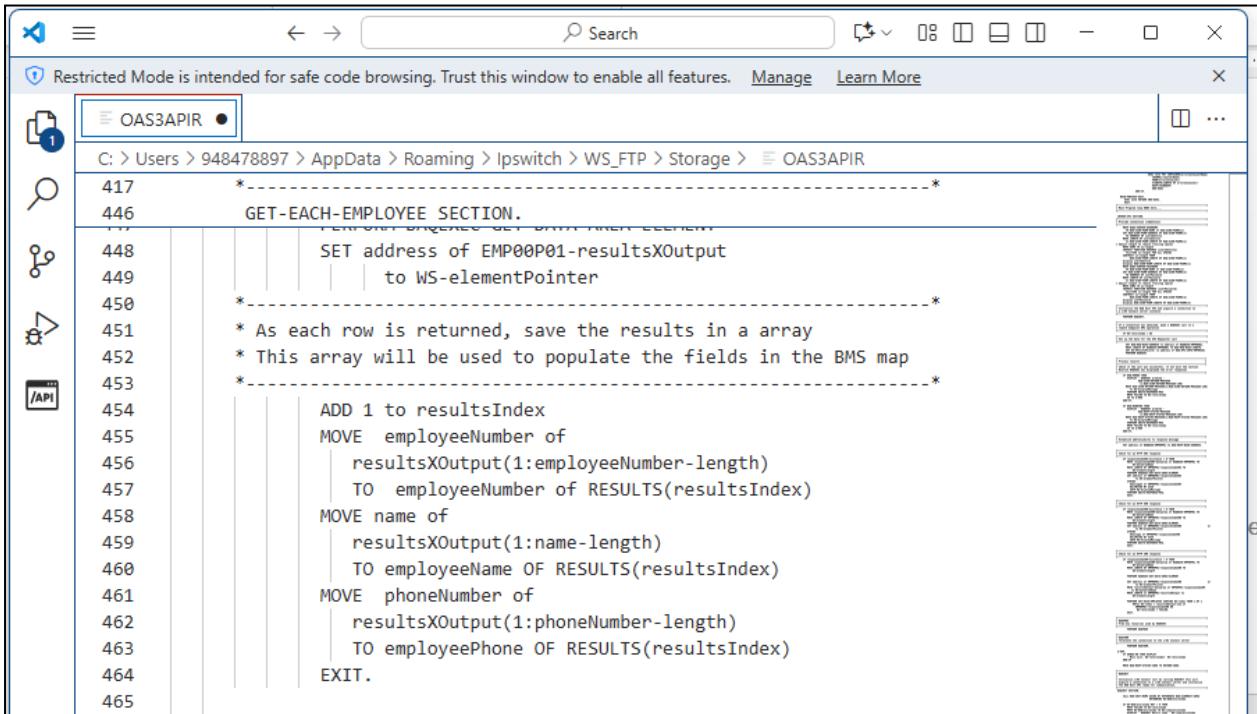


The screenshot shows a code editor window with the title bar "OAS3APIR". The left sidebar contains icons for file operations like Open, Save, Find, and Print, along with a "API" icon. The main pane displays REXX source code:

```
C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR
274      *-
275      IF responseCode500-existence > 0 THEN
288          EXIT.
289
290      *-
291      * Check for an HTTP 200 response
292      *-
293          IF responseCode200-existence > 0 THEN
294              MOVE responseCode200-dataarea of BAQBASE-EMP00P01 TO
295                  | WS-dataAreaName
296              MOVE LENGTH OF EMP00P01-responseCode200 TO
297                  | WS-elementLength
298
299              PERFORM BAQEXEC-GET-DATA-AREA-ELEMENT
300
301              SET address of EMP00P01-responseCode200
302                  | to WS-elementPointer
303              MOVE resultsXOutput-dataarea of EMP00P01-responseCode200
304                  | to WS-dataAreaName
305              MOVE LENGTH of EMP00P01-resultsXOutput to
306                  | WS-elementLength
307
308              PERFORM GET-EACH-EMPLOYEE VARYING WS-index FROM 1 BY 1
309                  UNTIL WS-index > resultsXOutput-num of
310                      | EMP00P01-responseCode200 OR
311                          | WS-returnCode = FAILED.
312
313      EXIT.
```

GET-EACH-EMPLOYEE SECTION

This section processes each entry in the array returned by the BAQGETN Host API. Note that it sets the address of the EMP00P01-resultsXOutput to the pointer returned by BAGETN(line 448). It then adds one to the counter index (line 454) and then uses the length of each returned file to move the data from the linkage area storage to a local array (line 455 to 463)

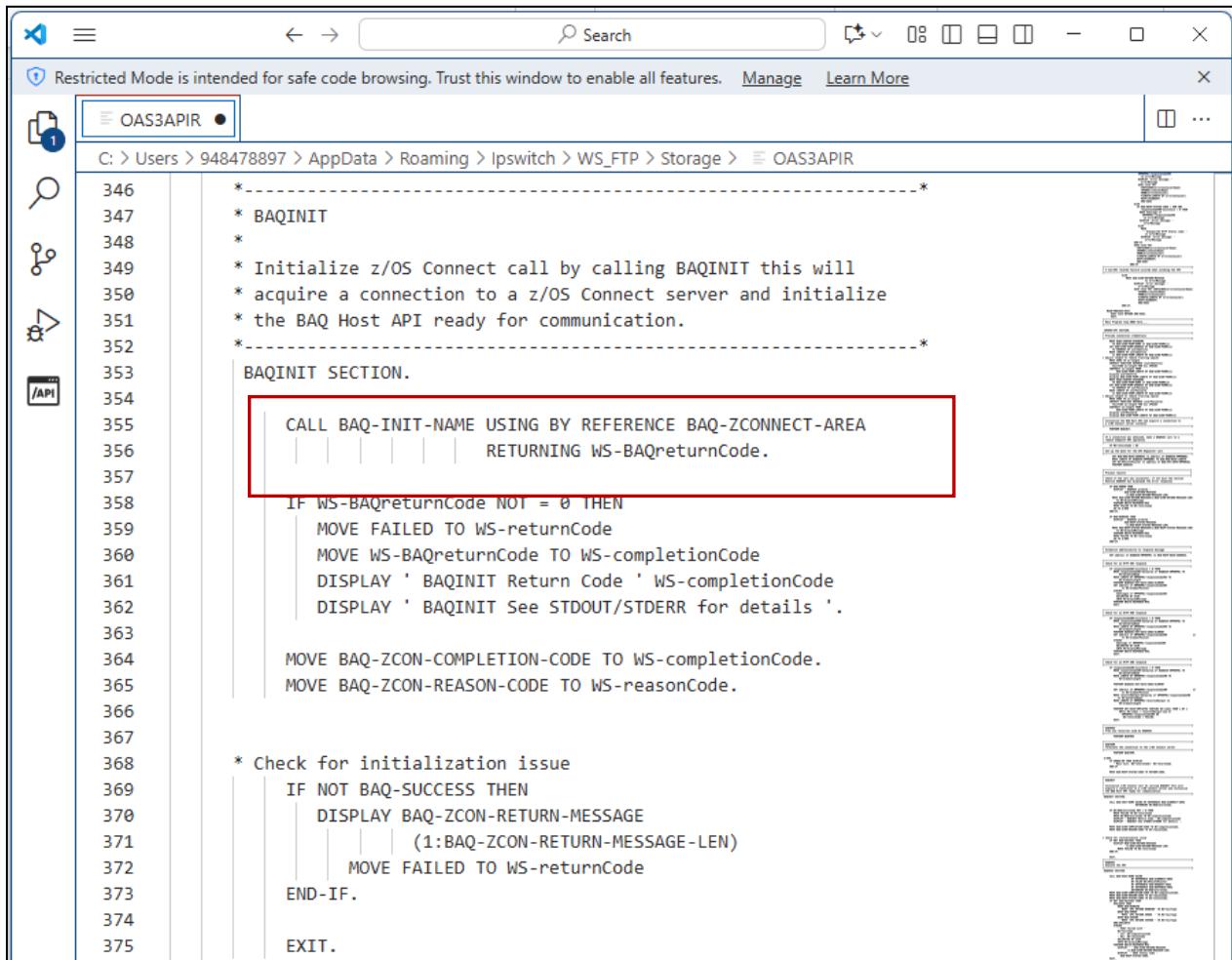


The screenshot shows a window titled 'OAS3APIR' with the following code:

```
417      *--*
446      GET-EACH-EMPLOYEE SECTION.
448      |   SET address of EMP00P01-resultsXOutput
449      |       to WS-elementPointer
450      *--*
451      * As each row is returned, save the results in a array
452      * This array will be used to populate the fields in the BMS map
453      *--*
454      |   ADD 1 to resultsIndex
455      MOVE employeeNumber of
456      |       resultsXOutput(1:employeeNumber-length)
457      |       TO employeeNumber OF RESULTS(resultsIndex)
458      MOVE name of
459      |       resultsXOutput(1:name-length)
460      |       TO employeeName OF RESULTS(resultsIndex)
461      MOVE phoneNumber of
462      |       resultsXOutput(1:phoneNumber-length)
463      |       TO employeePhone OF RESULTS(resultsIndex)
464
465      EXIT.
```

BAQINIT SECTION

Invoke the Host API that connects to the z/OS Connect server.

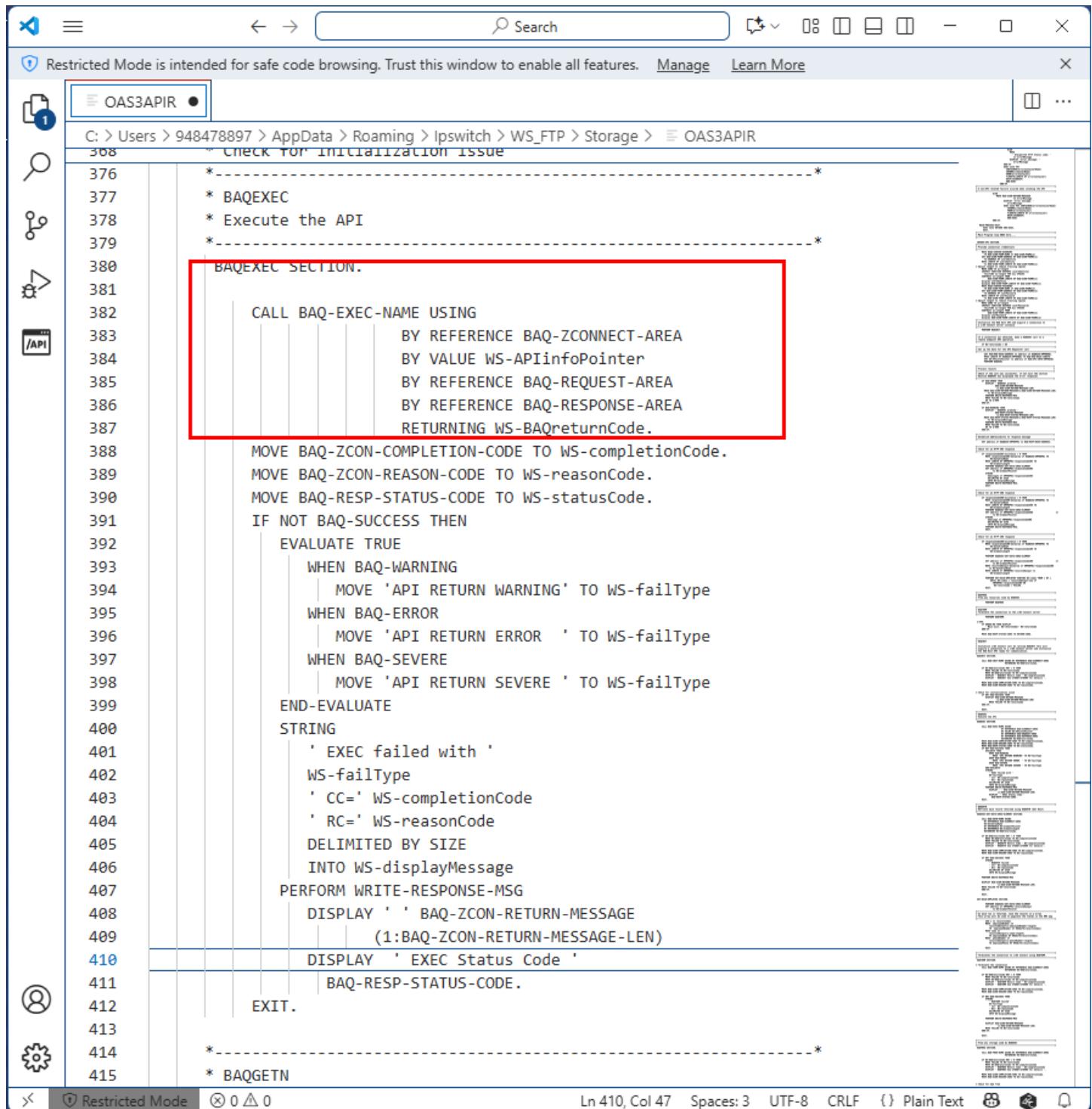


The screenshot shows a code editor window titled "OAS3APIR". The code is written in COBOL and defines the BAQINIT SECTION. The section starts with a call to BAQ-INIT-NAME and includes logic to handle return codes and completion codes. A red box highlights the call to BAQ-INIT-NAME.

```
346      *-----*
347      * BAQINIT
348      *
349      * Initialize z/OS Connect call by calling BAQINIT this will
350      * acquire a connection to a z/OS Connect server and initialize
351      * the BAQ Host API ready for communication.
352      *-----
353      BAQINIT SECTION.
354
355      CALL BAQ-INIT-NAME USING BY REFERENCE BAQ-ZCONNECT-AREA
356          |           |           |           |
357          |           |           |           | RETURNING WS-BAQreturnCode.
358
359      IF WS-BAQreturnCode NOT = 0 THEN
360          MOVE FAILED TO WS-returnCode
361          MOVE WS-BAQreturnCode TO WS-completionCode
362          DISPLAY ' BAQINIT Return Code ' WS-completionCode
363          DISPLAY ' BAQINIT See STDOUT/STDERR for details '.
364
365          MOVE BAQ-ZCON-COMPLETION-CODE TO WS-completionCode.
366          MOVE BAQ-ZCON-REASON-CODE TO WS-reasonCode.
367
368      * Check for initialization issue
369      | IF NOT BAQ-SUCCESS THEN
370          |     DISPLAY BAQ-ZCON-RETURN-MESSAGE
371          |         | (1:BAQ-ZCON-RETURN-MESSAGE-LEN)
372          |         MOVE FAILED TO WS-returnCode
373      END-IF.
374
375      EXIT.
```

BAQEXEC SECTION

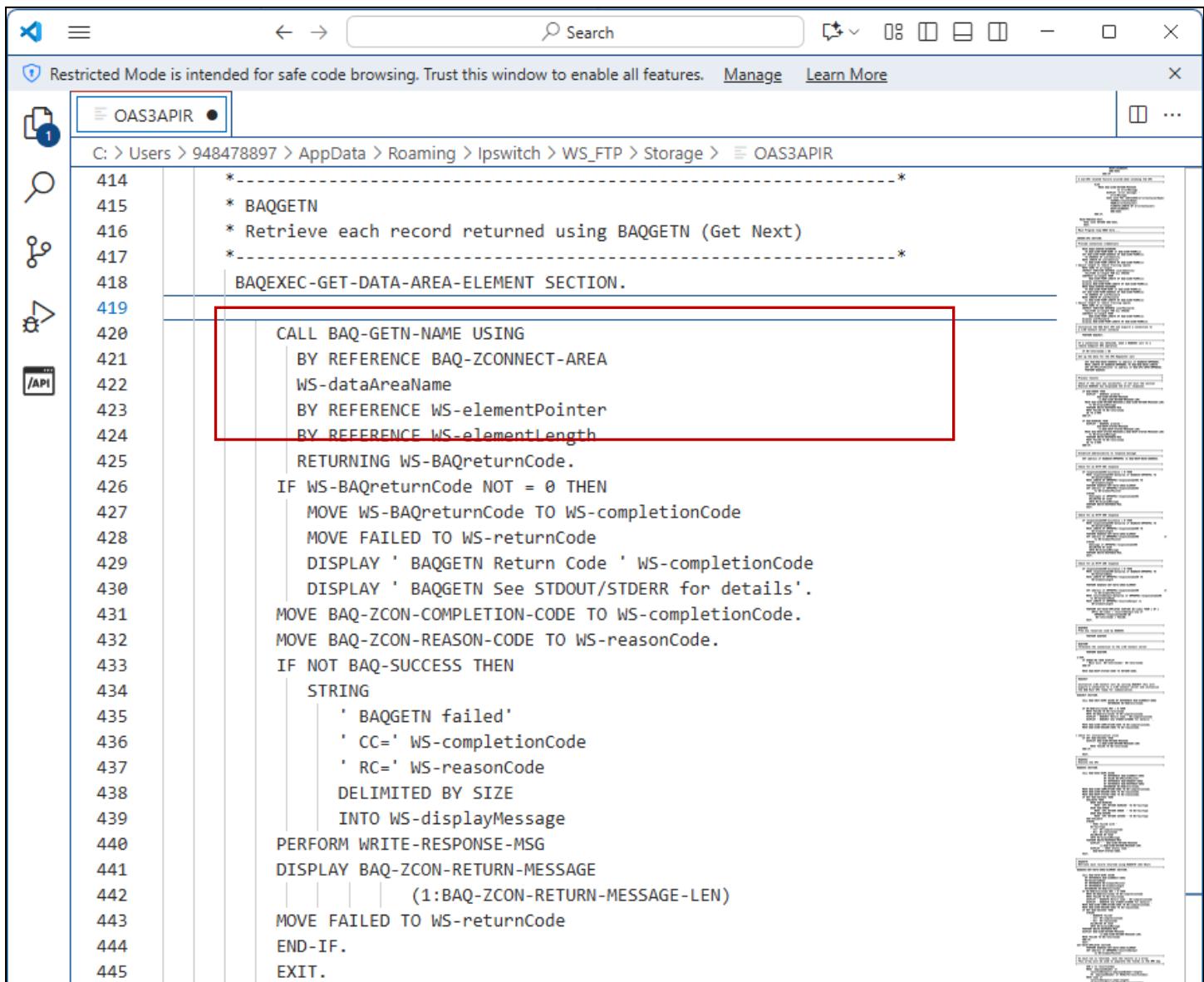
Scroll down further and you will see the call to the z/OS Connect API requester stub



```
C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR
368      *-----*
376      * BAQEXEC
377      * Execute the API
378      *
379      *-----*
380      BAQEXEC SECTION.
381
382      CALL BAQ-EXEC-NAME USING
383          | BY REFERENCE BAQ-ZCONNECT-AREA
384          | BY VALUE WS-APIinfoPointer
385          | BY REFERENCE BAQ-REQUEST-AREA
386          | BY REFERENCE BAQ-RESPONSE-AREA
387          | RETURNING WS-BAQreturnCode.
388
389      MOVE BAQ-ZCON-COMPLETION-CODE TO WS-completionCode.
390      MOVE BAQ-ZCON-REASON-CODE TO WS-reasonCode.
391      MOVE BAQ-RESP-STATUS-CODE TO WS-statusCode.
392      IF NOT BAQ-SUCCESS THEN
393          EVALUATE TRUE
394              WHEN BAQ-WARNING
395                  MOVE 'API RETURN WARNING' TO WS-failType
396              WHEN BAQ-ERROR
397                  MOVE 'API RETURN ERROR ' TO WS-failType
398              WHEN BAQ-SEVERE
399                  MOVE 'API RETURN SEVERE ' TO WS-failType
400          END-EVALUATE
401          STRING
402              ' EXEC failed with '
403              WS-failType
404              ' CC=' WS-completionCode
405              ' RC=' WS-reasonCode
406          DELIMITED BY SIZE
407          INTO WS-displayMessage
408          PERFORM WRITE-RESPONSE-MSG
409              DISPLAY ' ' BAQ-ZCON-RETURN-MESSAGE
410              (1:BAQ-ZCON-RETURN-MESSAGE-LEN)
411              DISPLAY ' EXEC Status Code '
412              BAQ-RESP-STATUS-CODE.
413
414      *-----*
415      * BAQGETN
```

BAQEXEC-GET-DATA-NAME SECTION (BAQGETN)

Section BAQEXEC-GET-DATA-NAME is invoked from various sections within the application.

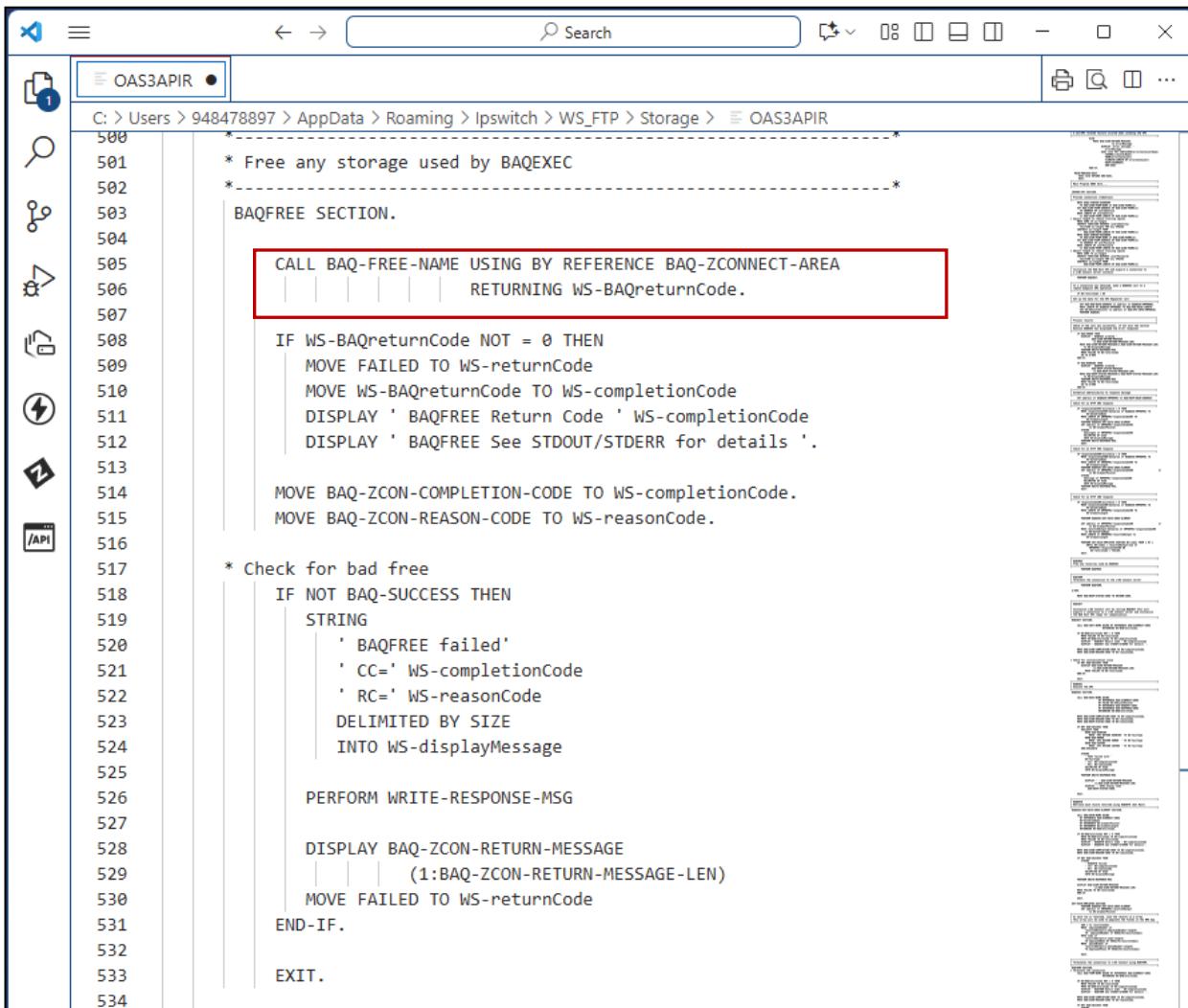


The screenshot shows a code editor window with the title "OAS3APIR". The code is written in COBOL and is part of the OAS3APIR application. The code is as follows:

```
414      *-----*
415      * BAQGETN
416      * Retrieve each record returned using BAQGETN (Get Next)
417      *-----*
418      BAQEXEC-GET-DATA-AREA-ELEMENT SECTION.
419
420      CALL BAQ-GETN-NAME USING
421          BY REFERENCE BAQ-ZCONNECT-AREA
422          WS-dataAreaName
423          BY REFERENCE WS-elementPointer
424          BY REFERENCE WS-elementLength
425          RETURNING WS-BAQreturnCode.
426
427      IF WS-BAQreturnCode NOT = 0 THEN
428          MOVE WS-BAQreturnCode TO WS-completionCode
429          MOVE FAILED TO WS-returnCode
430          DISPLAY ' BAQGETN Return Code ' WS-completionCode
431          DISPLAY ' BAQGETN See STDOUT/STDERR for details'.
432
433      MOVE BAQ-ZCON-COMPLETION-CODE TO WS-completionCode.
434      MOVE BAQ-ZCON-REASON-CODE TO WS-reasonCode.
435
436      IF NOT BAQ-SUCCESS THEN
437          STRING
438              ' BAQGETN failed'
439              ' CC=' WS-completionCode
440              ' RC=' WS-reasonCode
441              DELIMITED BY SIZE
442              INTO WS-displayMessage
443
444      PERFORM WRITE-RESPONSE-MSG
445      DISPLAY BAQ-ZCON-RETURN-MESSAGE
446          (1:BAQ-ZCON-RETURN-MESSAGE-LEN)
447      MOVE FAILED TO WS-returnCode
448      END-IF.
449      EXIT.
```

BAQFREE SECTION

The BAQFREE section frees the storage acquired by the Host APIs when processing.

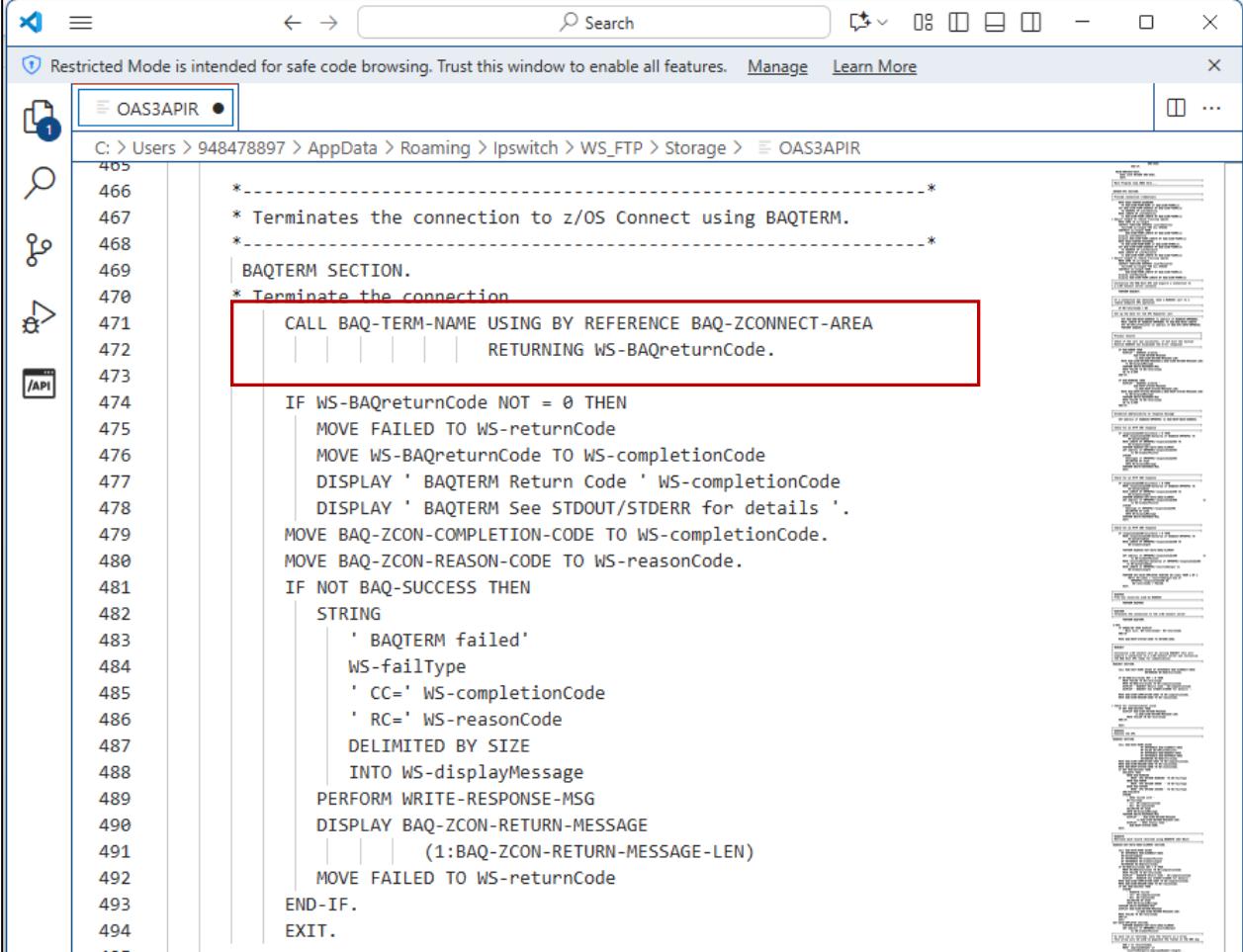


```
C: > Users > 948478897 > AppData > Roaming > Ipswitch > WS_FTP > Storage > OAS3APIR

500      *-----*
501      * Free any storage used by BAQEXEC
502      *-----*
503      BAQFREE SECTION.
504
505      CALL BAQ-FREE-NAME USING BY REFERENCE BAQ-ZCONNECT-AREA
506          | | | | | RETURNING WS-BAQreturnCode.
507
508      IF WS-BAQreturnCode NOT = 0 THEN
509          MOVE FAILED TO WS-returnCode
510          MOVE WS-BAQreturnCode TO WS-completionCode
511          DISPLAY ' BAQFREE Return Code ' WS-completionCode
512          DISPLAY ' BAQFREE See STDOUT/STDERR for details '.
513
514          MOVE BAQ-ZCON-COMPLETION-CODE TO WS-completionCode.
515          MOVE BAQ-ZCON-REASON-CODE TO WS-reasonCode.
516
517      * Check for bad free
518      IF NOT BAQ-SUCCESS THEN
519          STRING
520              ' BAQFREE failed'
521              ' CC=' WS-completionCode
522              ' RC=' WS-reasonCode
523          DELIMITED BY SIZE
524          INTO WS-displayMessage
525
526          PERFORM WRITE-RESPONSE-MSG
527
528          DISPLAY BAQ-ZCON-RETURN-MESSAGE
529              | | | (1:BAQ-ZCON-RETURN-MESSAGE-LEN)
530          MOVE FAILED TO WS-returnCode
531          END-IF.
532
533          EXIT.
534
```

BAQTERM SECTION

The BAQTERM section terminates the connection to the z/OS Connect server using the BAQTERM Host API.



The screenshot shows a code editor window with the title "OAS3APIR". The code is assembly language, specifically for the CICS API requester client application. A red box highlights the following section of code:

```
466      *-----*
467      * Terminates the connection to z/OS Connect using BAQTERM.
468      *-----*
469      | BAQTERM SECTION.
470      | * Terminate the connection
471      |     CALL BAQ-TERM-NAME USING BY REFERENCE BAQ-ZCONNECT-AREA
472      |             RETURNING WS-BAQreturnCode.
473
474      IF WS-BAQreturnCode NOT = 0 THEN
475          MOVE FAILED TO WS-returnCode
476          MOVE WS-BAQreturnCode TO WS-completionCode
477          DISPLAY ' BAQTERM Return Code ' WS-completionCode
478          DISPLAY ' BAQTERM See STDOUT/STDERR for details ' .
479          MOVE BAQ-ZCON-COMPLETION-CODE TO WS-completionCode.
480          MOVE BAQ-ZCON-REASON-CODE TO WS-reasonCode.
481          IF NOT BAQ-SUCCESS THEN
482              STRING
483                  ' BAQTERM failed'
484                  WS-failType
485                  ' CC=' WS-completionCode
486                  ' RC=' WS-reasonCode
487                  DELIMITED BY SIZE
488                  INTO WS-displayMessage
489                  PERFORM WRITE-RESPONSE-MSG
490                  DISPLAY BAQ-ZCON-RETURN-MESSAGE
491                      (1:BAQ-ZCON-RETURN-MESSAGE-LEN)
492                  MOVE FAILED TO WS-returnCode
493          END-IF.
494          EXIT.
495
```

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).

Compiling and link-editing the application program

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

Note that the CALL statements in program OAS3APIR uses a variable for the names z/OS Connect HOST API modules. This means that these program will be dynamically loaded or linked at execution. The load module library containing the CICS version of the HOST API modules, SBAQLIB1 were made available in the CICS region's DFHRPL concatenation sequence. We used the JCL below to compile and link-edit the programs.

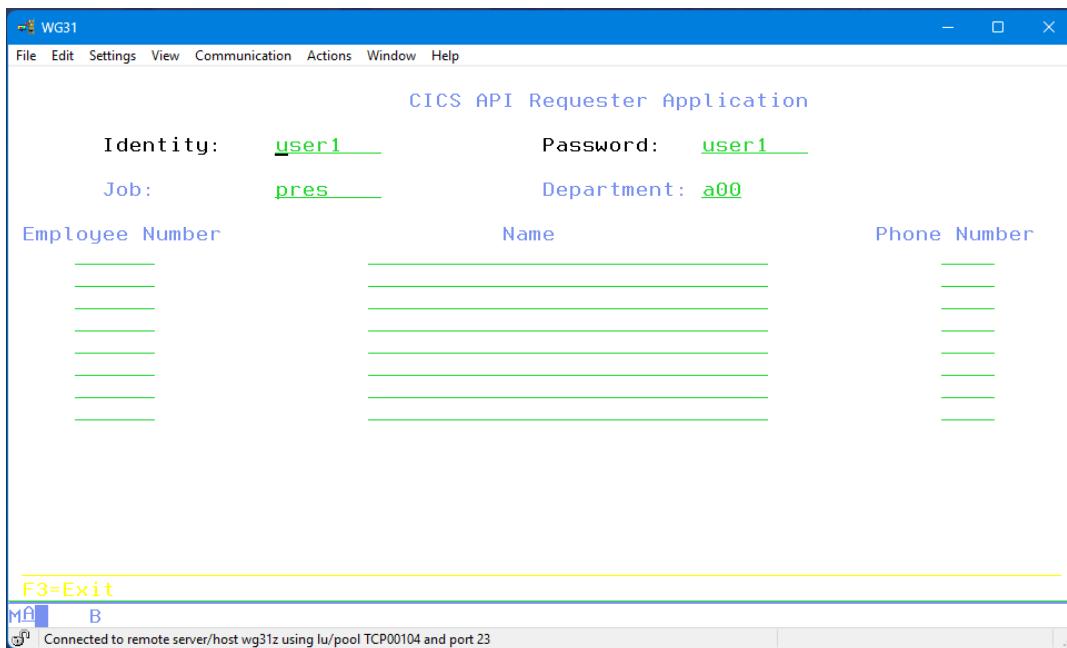
```
//COMPILE EXEC IGYWCL,LNGPRFX=SYS1.ECOBOL,  
// PARM.COBOL='LIB,NODYNAM,CICS,SIZE(4000K)'  
//COBOL.STEPLIB DD  
//          DD DISP=SHR,DSN=CICSTS61.CICS.SDFHLOAD  
//COBOL.SYSIN  DD DISP=SHR,DSN=JOHNSON.OAS3.SOURCE(OAS3APIR)  
//COBOL.SYSLIB DD DISP=SHR,DSN=JOHNSON.OAS3.MAPLIB  
//          DD DISP=SHR,DSN=JOHNSON.OAS3.COPYLIB  
//          DD DISP=SHR,DSN=ZCEE30.SBAQCOB  
//          DD DISP=SHR,DSN=CICSTS61.CICS.SDFHCOB  
//LKED.SYSLMOD DD DISP=SHR,DSN=JOHNSON.ZCEE.SDFHLOAD(OAS3APIR)  
//LKED.CICSLIB  DD DISP=SHR,DSN=CICSTS61.CICS.SDFHLOAD  
//LKED.SYSIN   DD *  
INCLUDE CICSLIB(DFHELII)
```

The jobs finished with zero return codes for all steps.

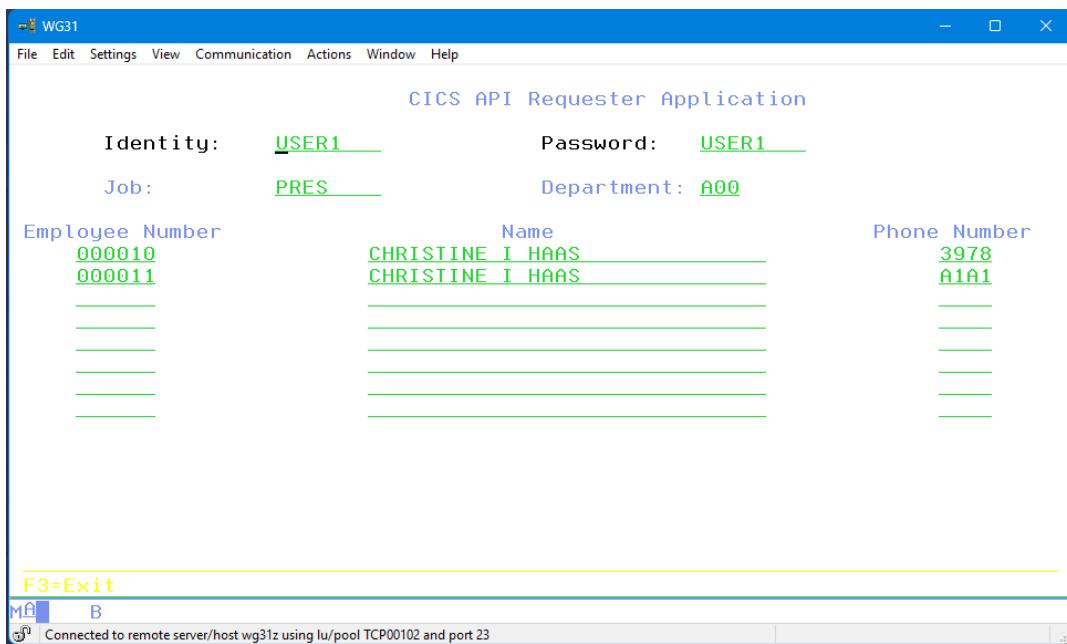
Testing the CICS API requester application program

The CICS API requester application was then tested. This section shows our results.

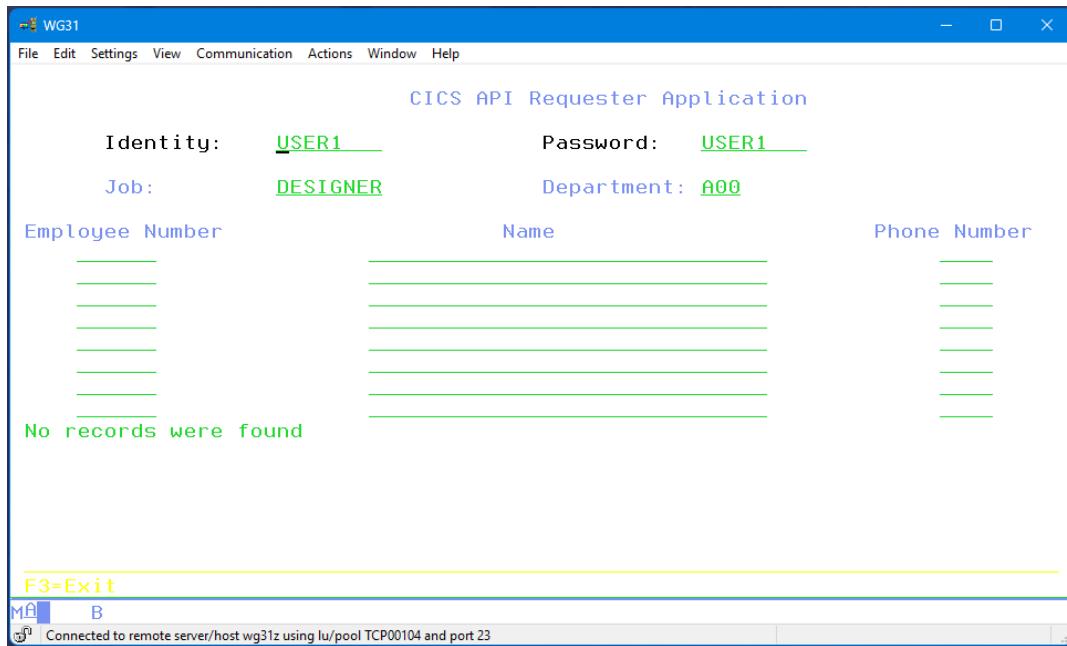
1. The transaction for the BMS application was entered and the screen below was displayed. The credentials for accessing the z/OS Connect server were entered (e.g., identity: *user1*, password: *user1*) and a value of *pres* was entered for the value of job and *a00* was entered for the value of department.



1. Pressing the **Enter** key displayed the requests below. The API returned an HTTP response of 200 along with the rows that met the selection criteria.



2. The job value was changed to DESIGNER and when enter was pressed, the API returned an HTTP response of 404 along with the message that no records were found.



3. Change the department to D11 and pressing entered returned the results below. The API returned an HTTP response of 200 along with the rows that met the selection criteria.

The screenshot shows a window titled 'WG31' running the 'CICS API Requester Application'. The application interface includes a menu bar with File, Edit, Settings, View, Communication, Actions, Window, and Help. The main area displays the following input fields:

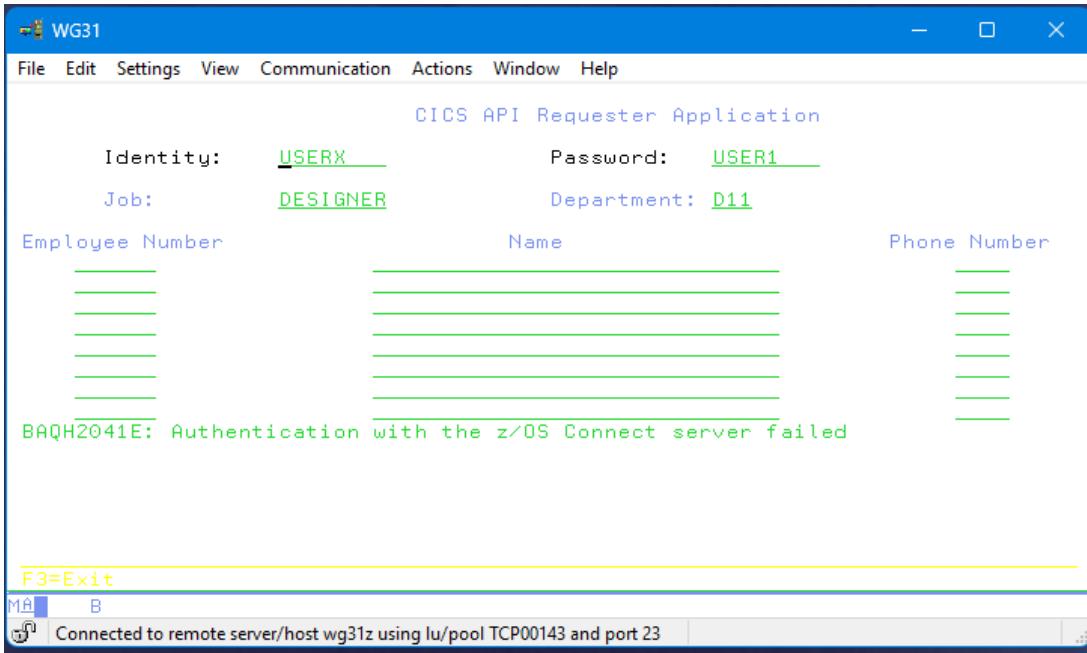
Identity:	USER1	Password:	USER1
Job:	DESIGNER	Department:	D11

Below these fields, there are three columns labeled 'Employee Number', 'Name', and 'Phone Number', each containing ten populated rows. The data is as follows:

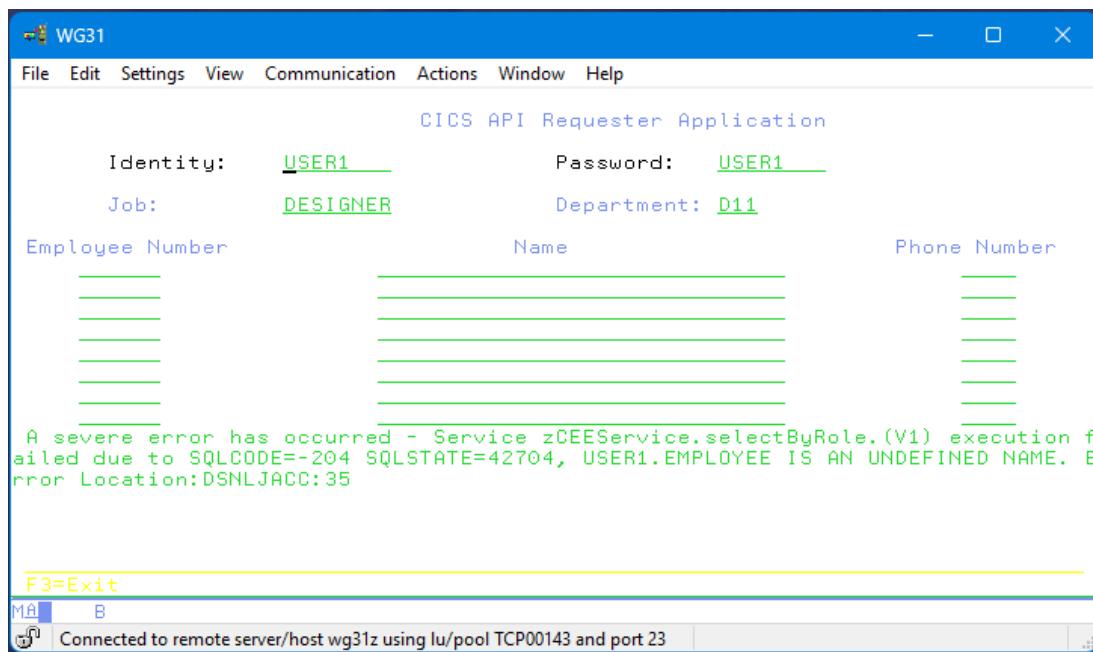
Employee Number	Name	Phone Number
000150	BRUCE ADAMSON	4510
000160	ELIZABETH R PIANKA	3782
000170	MASATOSHI J YOSHIMURA	2890
000180	MARILYN S SCOUTTEN	1582
000190	JAMES H WALKER	2986
000200	DAVID BROWN	4501
000210	WILLIAM T JONES	0942
000220	JENNIFER K LUTZ	0672

At the bottom of the window, there is a status bar with the text 'F3=Exit' and 'Connected to remote server/host wg31z using lu/pool TCP00143 and port 23'.

The value for identity was changed to USERX and when enter was pressed, the API request returned with the message authentication with the z/OS Connect server failed. Note that the message that was returned was from z/OS Connect (BAQG2041E). This failure occurred before the BAQEXEC host API could be invoked.



5. The value for identity was changed back to USER1. But before enter was pressed, the table being accessed was dropped. When enter was pressed, the API returned an HTTP response of 500 along with the message that a SQL code of -204 had been encountered.



Conclusion

In this document we tried to describe the process we used to develop a API requester application starting with the target API's specification document. The specification document was reviewed to understand the impact of how the details in the specification document have on the COBOL copy book details and how the z/OS Connect Host API is used.

Gradle on z/OS

Installing and enabling the Gradle Build Tool on z/OS

Building the API requester artifacts on any platform requires the use of the Gradle open-source automation tool. A Google search of “installing Grade on z/OS” provided the initial instructions on the steps required. We also followed the instructions in section *Using the API requester Gradle plug-in of the z/OS Connect documentation* ([URL https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=requester-using-api-gradle-plug-in](https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=requester-using-api-gradle-plug-in)) to create the required directories and files in directory my API requester project directory `/u/johnson/gradle/roster`.

Below are the steps we followed to install and enable the Gradle Build Tool on z/OS.

1. We began by updating the PATH environment variable the `.profile` in the home directory (`/u/johnson`) with the contents below:

```
export PATH=$PATH:$JAVA_HOME/bin:  
export PATH=$PATH:$JAVA_HOME/bin/classic:
```

These commands added access to the Java `jar` command to the PATH environment variable so it could be located for execution (equivalent to adding load modules to the STEPLIB DD statement). The Java `jar` will be useful for expanding or uncompacting compressed or zipped files.

N.B.: Updating the hidden file (`.profile`) made these changes effective for new shells in both batch and online processes. This environment variable can also be updated globally by updating the `profile` file in the `/etc` directory

2. We accessed OMVS and created a Gradle projects directory in the user’s home directory, e.g., `/u/johnson` using the `mkdir` command

```
mkdir /u/johnson/gradle
```

This directory is where we will be installing the Gradle code and where directories related to Gradle and the API requester projects will reside.

3. To ensure there was sufficient space for Gradle and my API requester projects we created a new ZFS file system and mounted it over this new directory. A MOUNT command was also added to the appropriate BPXPRM## PARMLIB to ensure the filesystem was mounted at the next IPL.

4. z/OS Connect ships the Grade code in its service stream (`gradle.zip` in `/usr/lpp/IBM/zosconnect/v3r0`). So, the Gradle code was extracted the Gradle code by positioning in directory `/u/johnson/gradle` in an OMVS shell and use the Java `jar` command to extract the Gradle code as shown in the command below.

```
jar -xf /usr/lpp/IBM/zosconnect/v3r0/gradle.zip
```

This created a new directory whose name reflected the level of Gradle code in the `gradle.zip` file. This new directory now contains the Gradle Build Tool. For our service level of z/OS Connect the Gradle level was 8.11.1 and the new directory name reflected this level, e.g., `/u/johnson/gradle/gradle-8.11.1`.

___ 5. The *.profile* file in the home directory (*/u/johnson*) with the contents below:

```
export PATH=$PATH:/u/johnson/gradle/gradle-8.11.1/bin  
export GRADLE_OPTS="-Dfile.encoding=UTF-8"  
export _BPXK_AUTOCVT=ON
```

This added the Gradle executables to my PATH environment variable so the executables could be located for execution and they set the required Gradle and OMVS environment variables.

___ 6. Next a Gradle project directory */u/johnson/gradle/roster* was created using the *mkdir* command.

```
mkdir /u/johnson/gradle/roster
```

The API requester project directory must be primed with two ASCII files, build.gradle and settings.gradle. The next steps show how the contents of these two files are determined.

N.B.: ASCII encoded files can be created using OMVS commands. First, use the *touch* command to create the file, **touch build.gradle**, then use the *change tag* command to change a file encoding to ASCII, **chtag -t -c ISO885901 build.gradle**

___ 7. The name and version of the z/OS Connect API requester plugin needs to be identified to Gradle using the contents of the *build.gradle* file and should have contents like this:

```
plugins {  
    id 'com.ibm.zosconnect.requester' version '1.2.1'  
}
```

N.B.: We were able to use the ISPF editor with ASCII files by using ISPF option 3.4. We simply enter */u/johnson/gradle* beside the Dsname Level area on this screen and pressed Enter. Then by using the line command **L**, we were able to traverse into the */u/johnson/gradle/roster* sub directories until the contents of directory */u/johnson/gradle/employee/src/main/api* were displayed. Then by entering line command **EA** (edit ASCII or **VA** (view ASCII) we opened an ISPF edit or an ISPF browse session for the file.

Alternatively, if you are in an OMVS shell, the *OEDIT* command can be used to open an ISPF edit session or *OBROWSE* can be used to open an ISPF browse session.

The version number (1.2.1) in the file was based on the service level of z/OS Connect installed. The relationship between the service level and the plug-in version can be determined from information available in *Table 2 at IBM z/OS Connect capabilities compatibility*, see URL <https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=reference-zos-connect-capabilities-compatibility>. For example, we had service level 3.0.88 installed and using information from the table below, we set the API requester plug-in version to 1.2.1.

Table 2. z/OS Connect zosConnect-3.0 artifact compatibility				
z/OS Connect release version	API provider Gradle plug-in version	Gradle versions that are supported by the API provider Gradle plug-in	API requester Gradle plug-in version	Gradle versions that are supported by the API requester Gradle plug-in
3.0.96	1.4.11	7.6.1 to 9.0.0	1.2.6	7.6.1 to 9.0.0
3.0.95	1.4.10	7.6.1 to 8.12.1	1.2.5	7.6.1 to 8.12.1
3.0.94	1.4.10	7.6.1 to 8.12.1	1.2.5	7.6.1 to 8.12.1
3.0.93	1.4.10	7.6.1 to 8.12.1	1.2.4	7.6.1 to 8.12.1
3.0.92	1.4.9	7.6.1 to 8.12.1	1.2.3	7.6.1 to 8.12.1
3.0.91	1.4.9	7.6.1 to 8.12.1	1.2.3	7.6.1 to 8.12.1
3.0.90	1.4.8	7.6.1 to 8.12.1	1.2.2	7.6.1 to 8.12.1
3.0.89	1.4.8	7.6.1 to 8.11.1	1.2.2	7.6.1 to 8.11.1
3.0.88	1.4.7	7.6.1 to 8.11.1	1.2.1	7.6.1 to 8.11.1
3.0.87	1.4.6	7.6.1 to 8.10	1.2.0	7.6.1 to 8.10
3.0.86	1.4.5	7.6.1 to 8.10	1.1.7	7.6.1 to 8.10
3.0.85	1.4.5	7.6.1 to 8.9	1.1.7	7.6.1 to 8.9
3.0.84	1.4.4	7.6.1 to 8.9	1.1.6	7.6.1 to 8.9
3.0.83	1.4.4	7.6.1 to 8.9	1.1.5	7.6.1 to 8.9

8. According to information in section *Using the API requester Gradle plug-in* (see URL <https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=requester-using-api-gradle-plugin>), Gradle will by default, try to retrieve plug-in from the repository hosted on the *Gradle Plug-in Portal*. Our z/OS image did not have access to the internet so we had to provide a local repository for Gradle to retrieve the necessary plug-ins. z/OS Connect provides a solution when the image is isolated from the internet (commonly called an *air gapped image*). Adding a *settings.gradle* file in the Gradle project directory provided a location from where additional code can be obtained locally. So, we added an ASCII file named *settings.gradle* in the API requester project directory with these contents.

```
pluginManagement {
    repositories {
        maven {
            url '/u/johnson/gradle/gradleLibs'
        }
    }
}
```

9. The contents of */u/johnson/gradle/gradleLibs* directory were obtained by extracting the contents of the *dependencies.zip* file shipped with z/OS Connect while in the Gradle projects directory using the Java *jar* command again, as in.

```
jar -xf /usr/lpp/IBM/zosconnect/v3r0/dependencies.zip
```

This created the directory *gradleLibs* in */u/johnson/gradle*.

10. We next needed to create the required API requester directory structure along with a default z/OS Connection API requester generation options file. WE used the *cd* command to change to *roster* project directory. We then entered the command *gradle apiRequesterLayout* with the results shown below.

```
JOHNSON:/u/johnson/gradle/roster:> gradle apiRequesterLayout
> Task :apiRequesterLayout
BAQG1028I: Created the layout for a z/OS Connect API requester project.

Next:
1. Place the OpenAPI 3 document(s) in the src/main/api folder.
2. Configure the generation of the API requester in
file:///u/johnson/gradle/roster/src/main/config/options.yaml.
3. Run the build task of the project to create the API requester WAR and
copybooks.

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed
```

Later we will need a directory that will contain COBOL books that need to be included in the COBOL API requester programs. This was a good time to create this directory so we used a *mkdir* command to create a *SYSLIB* directory.

11. We placed the API specification document in directory */u/johnson/gradle/employee/src/main/api* and as a file named *openapi.yaml*. The encoding for this file must be ASCII for the Gradle process to work correctly, so we ensured that if this file was uploaded to z/OS it is uploaded in binary mode so it is not converted to EBCDIC formatting.

```
WG31
File Edit Settings View Communication Actions Window Help
EDIT </u/johnson/gradle/employee/src/main/api/openapi> Columns 00001 00072
Command ==>
***** Top of Data ***** Scroll ==> PAGE
000001 openapi: 3.0.0
000002 info:
000003   description: "Maintain employee roster"
000004   version: 1.0.0
000005   title: employees
000006   servers:
000007     url: /roster
000008     x-ibm-zcon-roles-allowed:
000009       - Manager
000010   paths:
000011     "/roles/{job}":
000012       get:
000013         tags:
000014           - employee roster
000015           - select by job and department
000016         summary: Retrieve a list of employees based on job and department c
000017         description: Retrieve a list of employees based on job and department c
000018         operationId: getSelectByRole
000019         x-ibm-zcon-roles-allowed:
000020           - Manager
000021         parameters:
000022           - name: Authorization
000023             in: header
000024             required: false
000025             schema:
000026               type: apiKey
000027             - name: job
000028             in: path
000029             required: true
000030             schema:
000031               $ref: '#/components/schemas/Job'
000032             - name: dept
000033             in: query
000034             required: true
000035             schema:
000036               $ref: '#/components/schemas/Departments'
000037             responses:
000038
MA B
Connected to remote server/host wg31z using lu/pool TCP00112 and port 23
```

12. Next we used the process described in the above note to edit ASCII file *options.yaml* in directory */u/johnson/gradle/roster/src/main/config*. In this file we entered the contents below:

```
apiName: roster      1
characterVarying: YES 2
connectionRef: DB2SSID 3
defaultCharacterMaxLength: 255 4
inlineMaxOccursLimit: 1 5
language: COBOL 6
requesterExtension: cpy 7
requesterPrefix: EMP 8
```

1. Identifies the name of the API
2. Describes how strings (PIC X) files are to be generated.
3. Identifies the corresponding *zosconnect_endpointConnection* element that is defined in the z/OS Connect server's *server.xml* file.
4. Provides the default maximum string length for unbounded or unconstrained strings.
5. Controls when array is generated as an OCCURS Depending structure or as a dynamic area in the LINKAGE section.
6. Specifies that COBOL copy books should be generated.
7. Provides a 3-character file extension to be applied to all generated copybooks.
8. Provides a 3-character prefix to be used for all generated copybooks

N.B.: The copy books and WAR files for specific methods or operations can be generated by using the *operations* Gradle plug-in generation property, as providing *operation=getSelectByRole* in the *options.yaml* file.

For the complete set of plug-in properties that can be used, see section *The API requester Gradle plug-in properties and options* section of the documentation at URL

<https://www.ibm.com/docs/en/zos-connect/3.0.0?topic=requester-api-gradle-plug-in-properties-options>

When finished, the contents of the API requester project directory contain these directories and files.

```
JOHNSON:/u/johnson/gradle/roster:> ls -al
total 96
drwxr-xr-x  4 JOHNSON  SYS1          8192 Sep 11 18:18 .
drwxr-xr-x  7 JOHNSON  SYS1          8192 Sep 11 13:01 ..
drwxr-xr-x  2 JOHNSON  SYS1          8192 Sep 11 09:08 SYSLIB
-rw-r--r--  1 JOHNSON  SYS1           66 Sep 10 14:59 build.gradle
-rw-r--r--  1 JOHNSON  SYS1          119 Sep 10 14:43 settings.gradle
drwxr-xr-x  3 JOHNSON  SYS1          8192 Sep 10 16:42 src
```

Building the API requester artifacts from an OpenAPI3 specification document

Gradle builds can be performed on a variety platforms but because of the work done in the previous section, we now can run the JCL below to invoke the Gradle build process on z/OS.

The API's specification document is used to generate the COBOL copybooks required for the COBOL API client application and the API requester web application archive (WAR) file. The API requester WAR is accessed by the z/OS Connect API requester runtime to have the transformation of COBOL working storage to and from JSON request and response message as well as other API related actions.

```
*****  
/* SET SYMBOLS  
*****  
//EXPORT EXPORT SYMLIST=(*)  
// SET JAVAHOME='/usr/lpp/java/J8.0_64'  
// SET GRDLHOME='/u/johnson/gradle/roster'  
*****  
/* Step GRADLE - Use the gradle build command  
*****  
//GRADLE EXEC PGM=IKJEFT01,REGION=0M  
//SYSTSPRT DD SYSOUT=*  
//SYSERR DD SYSOUT=*  
//STDOUT DD SYSOUT=*  
//SYSTSIN DD *,SYMBOLS=EXECSYS  
BPXBATCH SH +  
export JAVA_HOME=&JAVAHOME; +  
export GRADLE_HOME=&GRDLHOME; +  
cd $GRADLE_HOME; +  
gradle build --info
```

N.B.: Note that each HTTP method defined in the specification 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., *EMP*) 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.

13. The STDOUT output of the job will display numerous messages about the processing of the specification document. Some of the more important messages are shown below. These messages shown here identify the copy books that were generated for each method (or *operation*) selected or found in the API specification. It is important to note which copy books were generated for each method and use the corresponding copy book for each method.

```
> Task :generateApiRequester
Caching disabled for task ':generateApiRequester' because:
Total 6 operation(s) (success: 6, ignored: 0) defined in api : roster
----- Successfully processed operation(s) -----
operationId: deleteDeleteEmployee, path: /employees/{employee}, method: DELETE
- request data structure : EMP05Q01.cpy
- response data structure : EMP05P01.cpy

operationId: getSelectByRole, path: /roles/{job}, method: GET
- request data structure : EMP00Q01.cpy
- response data structure : EMP00P01.cpy

operationId: getDisplayEmployee, path: /employees/details/{employee}, method: GET
- request data structure : EMP02Q01.cpy
- response data structure : EMP02P01.cpy

operationId: postInsertEmployee, path: /employees, method: POST
- request data structure : EMP01Q01.cpy
- response data structure : EMP01P01.cpy

operationId: putInsertEmployee, path: /employees/{employee}, method: PUT
- request data structure : EMP04Q01.cpy
- response data structure : EMP04P01.cpy

operationId: getSelectEmployee, path: /employees/{employee}, method: GET
- request data structure : EMP03Q01.cpy
- response data structure : EMP03P01.cpy

BAQP0010I: Successfully processed the OpenAPI document.
BAQG1023W: Processing of the OpenAPI definition completed with warnings. See the log for more details:
file:///u/johnson/gradle/roster/build/logs/zosConnectRequester/
BAQG1012I: The following operations were parsed successfully:
deleteDeleteEmployee
getSelectByRole
getDisplayEmployee
postInsertEmployee
putInsertEmployee
getSelectEmployee
BAQG1022I: Parsing complete for OAS definition file: openapi.yaml.
```

14. Below shows the directory structure created by the Gradle build process and the copy books associated with each operation, (method).

```
JOHNSON:/u/johnson/gradle/roster/build/generated/zosConnectRequester/structures/COBOL:> ls -Ral
.:
total 128
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  3 JOHNSON  SYS1      8192 Sep 11 09:07 ..
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 deleteDeleteEmployee
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 getDisplayEmployee
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 getSelectByRole
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 getSelectEmployee
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 postInsertEmployee
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 putInsertEmployee
./deleteDeleteEmployee:
total 80
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 ..
-rw-r--r--  1 JOHNSON  SYS1     1361 Sep 11 09:07 API05I01.cpy
-rw-r--r--  1 JOHNSON  SYS1     6676 Sep 11 09:07 API05P01.cpy
-rw-r--r--  1 JOHNSON  SYS1    1181 Sep 11 09:07 API05Q01.cpy
./getDisplayEmployee:
total 112
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 ..
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 ..
-rw-r--r--  1 JOHNSON  SYS1     1366 Sep 11 09:07 API02I01.cpy
-rw-r--r--  1 JOHNSON  SYS1    19404 Sep 11 09:07 API02P01.cpy
-rw-r--r--  1 JOHNSON  SYS1    1179 Sep 11 09:07 API02Q01.cpy
./getSelectByRole:
total 96
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 ..
-rw-r--r--  1 JOHNSON  SYS1     1344 Sep 11 09:07 API00I01.cpy
-rw-r--r--  1 JOHNSON  SYS1     9880 Sep 11 09:07 API00P01.cpy
-rw-r--r--  1 JOHNSON  SYS1    1912 Sep 11 09:07 API00Q01.cpy
./getSelectEmployee:
total 96
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 ..
-rw-r--r--  1 JOHNSON  SYS1     1355 Sep 11 09:07 API03I01.cpy
-rw-r--r--  1 JOHNSON  SYS1    12298 Sep 11 09:07 API03P01.cpy
-rw-r--r--  1 JOHNSON  SYS1    1178 Sep 11 09:07 API03Q01.cpy
./postInsertEmployee:
total 96
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 ..
-rw-r--r--  1 JOHNSON  SYS1     1340 Sep 11 09:07 API01I01.cpy
-rw-r--r--  1 JOHNSON  SYS1     6674 Sep 11 09:07 API01P01.cpy
-rw-r--r--  1 JOHNSON  SYS1    9703 Sep 11 09:07 API01Q01.cpy
./putInsertEmployee:
total 80
drwxr-xr-x  2 JOHNSON  SYS1      8192 Sep 11 09:07 .
drwxr-xr-x  8 JOHNSON  SYS1      8192 Sep 11 09:07 ..
-rw-r--r--  1 JOHNSON  SYS1     1355 Sep 11 09:07 API04I01.cpy
-rw-r--r--  1 JOHNSON  SYS1     6673 Sep 11 09:07 API04P01.cpy
-rw-r--r--  1 JOHNSON  SYS1    3022 Sep 11 09:07 API04Q01.cpy
```

15. The copy books were generated into OMVS directories and encoded in ASCII. To use them in a COBOL application they need to be converted to EBCDIC and then copied them into a partitioned datasets using the job below. Below is a snippet of the JCL used to perform these functions. Step ICONV invokes the OMVS *iconv* command. This command converts an ASCII file to EBCDIC and stores the results in the SYSLIB directory earlier. STEP COPY copies the contents of the SYSLIB directory into a PDS. The full job can be viewed in the Appendix of this document.

```

//*****SET SYMBOLS
//*****EXPORT EXPORT SYMLIST=(*)
// SET GRDLPROJ='/u/johnson/gradle/roster/'
// SET COBOL='build/generated/zosConnectRequester/structures/COBOL/'
// SET DSNAME='JOHNSON.OAS3.COPYLIB'
//ICONV EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH +
export GRADLE_PROJECT=&GRDLPROJ; +
export COBOL=&COBOL; +
cd $GRADLE_PROJECT; +
cd $COBOL; +
cd deleteDeleteEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP05I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP05I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP05P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP05P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP05Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP05Q01 ; +
cd ../getDisplayEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP02I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP02I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP02P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP02P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP02Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP02Q01 ; +
. . .
cd ../putInsertEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP04I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP04I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP04P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP04P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP04Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP04Q01
//COPY EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH +
export GRADLE_PROJECT=&GRDLPROJ; +
export DSNAME=&DSNAME; +
cp $GRADLE_PROJECT/SYSLIB/* " // '$DSNAME' "

```

Convert ASCII copy books to EBCDIC copy books

JCL to convert the copy books from ASCII to EBCDIC encoding and then copy them to a partitioned data set.

```
//EXPORT EXPORT SYMLIST=(*)
// SET GRDLPROJ='/u/johnson/gradle/roster/'
// SET COBOL='build/generated/zosConnectRequester/structures/COBOL/'
// SET DSNAME='JOHNSON.OAS3.COPYLIB'
//ICONV EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH +
export GRADLE_PROJECT=&GRDLPROJ; +
export COBOL=&COBOL; +
cd $GRADLE_PROJECT; +
cd $COBOL; +
cd deleteDeleteEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP05I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP05I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP05P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP05P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP05Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP05Q01 ; +
cd ../getDisplayEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP02I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP02I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP02P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP02P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP02Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP02Q01 ; +
cd ../getSelectByRole ; +
iconv -f ISO8859-1 -t IBM-1047 EMP00I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP00I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP00P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP00P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP00Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP00Q01 ; +
cd ../getSelectEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP03I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP03I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP03P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP03P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP03Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP03Q01 ; +
cd ../postInsertEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP01I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP01I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP01P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP01P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP01Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP01Q01 ; +
cd ../putInsertEmployee ; +
iconv -f ISO8859-1 -t IBM-1047 EMP04I01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP04I01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP04P01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP04P01 ; +
iconv -f ISO8859-1 -t IBM-1047 EMP04Q01.cpy > +
    $GRADLE_PROJECT/SYSLIB/EMP04Q01
//OGET EXEC PGM=IKJEFT01,REGION=0M
//SYSTSPRT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//SYSTSIN DD *,SYMBOLS=EXECSYS
BPXBATCH SH +
export GRADLE_PROJECT=&GRDLPROJ; +
export DSNAME=&DSNAME; +
cp $GRADLE_PROJECT/SYSLIB/* //'$DSNAME'
```