IBM – CICS Integration Workshop

L90 - CICS JSON services using the CICS JSON Assistant

Lab Version V6.1.04.zVA

January 18, 2024

Please send any comments on this lab exercise to:
Leigh Compton, Steve Fowlkes, or Eric Higgins
lcompton@us.ibm.com
fowlkes@us.ibm.com
erichiggins@us.ibm.com

Overview

CICS provides a facility called the JSON Assistant to generate JSON services definitions from supplied program language structures. This technique is called bottom-up because we start with a copybook definition and generate the parts necessary to receive a JSON request, parse the JSON into a copybook, then invoke a CICS business logic program. Then, when the business logic program returns, CICS will put the response into JSON.

The purpose of this exercise is to illustrate the use of the CICS JSON Assistant in a bottom-up fashion. You can also do JSON service top-down, and can also translate a copybook to/from within a program using either top-down or bottom-up.

Each section of this lab exercise goes into detail on the various functions of the CICS JSON Assistant for a bottom up scenario.

Scenario

In our scenario, you are a programmer at a mutual fund company. You have an existing COBOL program called FUNDPROG that can Create, Read, Update, and Delete funds. You want to expose this program as a JSON service. To quickly expose the program as a JSON service, you will run the DFHLS2JS program (part of the CICS JSON Assistant) and generate a schema file and a WSBind file.

After installing the new CICS resources, you will test your new JSON service.

Lab Requirements

Please note that there are often several ways to perform functions in and for CICS. This lab exercise will present one of the ways. If you are familiar with CICS, you will notice that some of the statements are general, and not necessarily true for every situation.

This lab uses the IBM Explorer, ISPF under TSO, and CICS. If you are not familiar with these, please contact one of the lab instructors for assistance.

The following are other assumptions made in this lab exercise.

- **CICS TS V6.1:** This lab exercise uses facilities and naming conventions that were introduced in CICS TS V5.2. Each team performing the lab exercise has their own CICS region, to include their own CSD and other supporting CICS files. It currently uses CICS TS V6.1.

- **Login:** TSO userids are available with appropriate passwords

- **IBM Explorer:** This lab exercise using the IBM Explorer.

Lab Step Overview

Part 1: Configuring the IBM Explorer connection to CICS

Although you can complete the lab without using the IBM Explorer, in this part of the lab, you will configure the IBM Explorer.

Part 2: Understand the Existing Program Interface

In this part of the lab, you will explore the interface of the CICS COBOL program we are working with.

Part 3: Run the DFHLS2JS Program

This part of the lab exercise has you run the DFHLS2JS program. The inputs to the DFHLS2JS program are a Language Structure data definition and input parameters. The outputs are schema definition and a WSBind file.

Part 4: Define CICS resource definitions

In this step you will define a JVM server, and program, and a VSAM file to CICS.

Part 5: Install the CICS JSON Services Resources

In a previous lab exercise you defined a PIPELINE definition. In this lab exercise we will place a generated WSBind file in the pickup directory of your PIPLINE definition and tell CICS to SCAN the pipeline.

Part 6: Test your new JSON Service

We will use the RESTClient plug-in that is part of Firefox to test your JSON service.

Part 7: Summary

This is a recap of the steps performed in this lab exercise.

Important: There is a folder on the Windows desktop named *CopyPaste Files.* This folder contains a file with the commands and other text used in this workshop. Locate and open the copy/paste file specific to this exercise. Use the copy-and-paste function (**Ctrl-C** and **Ctrl-V**) to enter commands or text. It will save time and help avoid typo errors.

As a reminder text that appears in this file will be highlighted in yellow.

Part 1: Configure the IBM Explorer Connection to CICS

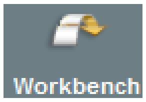
In this part of the lab exercise you will configure the connection between the IBM Explorer running on your workstation to CICS running on z/OS.

Start the IBM Explorer

___1. From the **desktop**, **double-click** the **IBM Explorer** icon to start it is not already running.

___2. **If** you are prompted for a workspace, click the **OK** button to select the default.

___3. **If** the IBM Explorer shows you a **Welcome page** click the **Workbench icon** in the upper-right corner to go to the Explorer workbench. Then **maximize** the window.



Verify that you have RSE and CMCI connections to z/OS in your IBM Explorer

___4. If you have not already created connections to the z/OS host system, follow the instructions in the **Connection Document** and then return here. Both the **Remote System Explorer** and **CMCI** connections should be started and active.

Part 2: Understand Existing Program Interface

CICS TS provides the JSON Assistant to generate JSON services definitions from supplied program language structures, and for generating language structures from supplied schema documents. The CICS JSON Assistant is actually two utility programs that run in z/OS batch. Both programs use the IBM Java SDK.

In this part of the lab exercise we will use the DFHLS2JS utility (Language Structure to JSON service) to generate a new schema from an existing data definition. We will be using a COBOL program, but the CICS JSON Assistant supports COBOL, PL/I, C, and C++. As input, the DFHLS2JS utility needs to import the language copybooks that matches the program's COMMAREA for the request and response. The DFHLS2JS utility can invoke a CICS program using a channel interface, but for this lab exercise, we will use a COMMAREA interface (to simulate a COMMAREA like you might encounter with an older CICS program.

Below is the COBOL definition for the COMMAREA in our FUNDPROG program. This is the layout of the input and output messages for our new JSON service.

```
*    FUNDPROG COMMAREA LAYOUT
     03  REQUEST-TYPE             PIC X.
     03  RET-CODE                 PIC XX.
     03  FUND-ID                  PIC X(8).
     03  FUND-NAME                PIC X(50).
     03  FUND-RATING              PIC X.
     03  FUND-PRICE               PIC X(15).
     03  RETURN-COMMENT           PIC X(50).
```

___1. If you'd like to look at the FUNDPROG program, you can view it through the IBM Explorer. From the IBM Explorer, the **Remote Systems** view, double-click **USER1.CICSLAB.UTIL(FUNDPROG)** to open it in an editor. When you are finished reviewing the program, just close the editor. (There should have been no changes to the program.)

Part 3: Run the program DFHLS2JS

For an existing CICS application, the COMMAREA will already be mapped by a language structure. The language structure and a set of input parameters are used as input to a CICS supplied utility which runs as a batch procedure. The procedure is called DFHLS2JS. This converts the supplied language structure into a JSON schema document and also generates a WSBind file and a log file.

The **DFHLS2JS** input parameters are:

REQMEM (the definition of the request),
RESPMEM (the definition of the response),
PDSLIB (PDS containing REQMEM and RESPMEM),
LANG (language),
PGMNAME (name of the target program),
PGMINT (program interface (COMMAREA or CHANNEL)),
URI – the name of the JSON service end-point that will be placed in the schemas and also used by CICS to dynamically create a URIMAP.
MAPPING-LEVEL – The CICS processing level
WSBind, **JSON Schemas**, **LOGFILE** – to specify the files to generate for the WSBind file, the JSON Schemas, and the log file.

For a full description of the input parameters, search for “DFHLS2JS” in the Information Center.

This JCL has been customized for the workshop environment as follows (note that in addition to names and locations for our environment we specified mapping level 4.0 since it is recommended to use the most current mapping level):

```
->input to the DFHLS2JS job<-
JSON-SCHEMA-REQUEST=/u/wspot01/cicslab/json/FundProgReqJSON.json
JSON-SCHEMA-RESPONSE=/u/wspot01/cicslab/json/FundProgRespJSON.json
LANG=COBOL
LOGFILE=/u/wspot01/cicslab/logs/FundProgReqJSONUtilLog.log
PDSLIB=//WSPOT01.CICSLAB.UTIL
PGMINT=COMMAREA
PGMNAME=FUNDPROG
MAPPING-LEVEL=4.0
REQMEM=FUNDCOMM
RESPMEM=FUNDCOMM
URI=http://ESYSMVS1.ZTEC.DMZ:9004/json/fundprog
WSBIND=/u/wspot01/cicslab/json/wspickup/provider/FundProgReqJSON.wsbind
*/
*/
```

Log file (generated)

The log file contains detailed information on the steps the job has taken to create the output files. The contents of the log file are not normally required by end users. However, it does contain a copy of the generated schema which can be used if the generated schema file is misplaced. The log file could also be used in problem determination.

Schema file (generated)
The schema file defines all the information required for a client to access this JSON service.

WSBind file (generated)
The WSBind file contains the meta-data that CICS uses at runtime to marshal and de-marshal the request and response messages in JSON into the format expected by the application program. The WSBind file is read by CICS when a WEBSERVICE resource is installed onto a PIPELINE (it is not read on every request).

It is important to ensure that the version of the WSBind file matches the schema file used by clients. If changes are made to the language structure, the schema and WSBind file need to be re-generated, and the new schema sent to the client.

If the PIPELINE scanning mechanism is used to install a WEBSERVICE, the resource name will be based on the name of the WSBind file. It is suggested to keep the names of the WSBind file and the schema files the same to ensure installed JSON services and clients match.

Compile FUNDPROG

> Note: If you have performed another lab exercise which uses the FUNDPROG program, it has already been compiled. You can skip to step 4.

___**1.** From the **Remote System Explorer** perspective, the **Remote Systems** view, **right-click USER1.CICSLAB.JCL(FUNDPROG)**, and from the context menu click **Submit**.

___**2.** From the **Job Submission Confirmation** dialog, click the **OK** button.

___**3.** From the **Remote System Explorer** perspective, the **Remote Systems** view, verify that your job ran **successfully**. **Note** that you may have to refresh the view. **Note** that the return codes are towards the top of the JCL listing.

Isolate the COMMAREA in a separate file for DFHLS2JS

The CICS JSON Assistant accepts a language structure such as a COBOL copybook as input. If your application program has the Commarea defined in-line rather than in a copybook, you will need to create a separate file containing just the Commarea structure. For this workshop, that step has been done for you.

___**4.** **USER1.CICSLAB.UTIL(FUNDCOMM)** was created by copying the fields from the COMMAREA into a PDS member, as shown below. Note that the 03's should be in area B (column12).

```
03  REQUEST-TYPE             PIC X.
03  RET-CODE                 PIC XX.
03  FUND-ID                  PIC X(8).
03  FUND-NAME                PIC X(50).
03  FUND-RATING              PIC X.
03  FUND-PRICE               PIC X(15).
03  RETURN-COMMENT           PIC X(50).
```

Look at the job that invokes the CICS JSON Assistant

___**5.** From the IBM Explorer, the **Remote Systems** view, double-click **USER1.CICSLAB.JCL(LS2JSJC5)** to open it in an editor. (this is the JCL used to generate the schema and a WSBind file)

The input statements indicate that:

The PDS containing the members describing the input and output messages are in USER1.CICSLAB.UTIL.

___**6.** The PDS member describing the request (REQMEM) is named FUNDCOMM.

The PDS member describing the response (RESPMEM) is name FUNDCOMM.

The program interface (PGMINT) is a COMMAREA.

The application program (PGMNAME) that will be invoked after the COMMAREA is prepared (conversion from JSON to COMMAREA is complete) is named FUNDPROG.

___7. **Close** the file.

Run the CICS JSON Assistant

___8. From the **Remote System Explorer** perspective, the **Remote Systems** view, expand the **USER1.CICSLAB.JCL** file, **right-click** the **LS2JSJC5** member, and from the context menu, select **Submit**. Then from the **Job Submission Confirmation** dialog, click the **OK** button.

___9. From the **Remote System Explorer** perspective, the **Remote Systems** view, the **JES** node, click the **refresh** button, then **double-click your Job**.

___10. **Ensure** that your LS2JSJC5 **Job** ran **successful**ly. Note step return codes are towards the top of the listing. They should all be 0. Close the Job Listing when you are done looking at it.

___11. From the **Remote System Explorer** perspective, the **Remote Systems** view, navigate to the **z/OS UNIX Files/My Home/cicslab/json/wspickup/provider** directory.

The LS2JSJC5 job you just ran created the WSBind file and schema file for your JSON service. Based on the input parameters, the WSBind file was placed in /My Home/cicslab/json/wspickup/provider and is called FundProgReqJSON.wsbind.

The schema files were placed in /My Home/cicslab/json, and are called FundProgReqJSON.json and FundProgRespJSON.json.

Note: You may need to click the refresh button on the z/OS UNIX Files view to see your new WSBind and schema files. If you can not see the files contact a lab instructor.

Part 4: Define CICS resource definitions

The FUNDPROG program needs a VSAM file named USER1.CICSLAB.FUNDFILE. To invoke FUNDPROG as a JSON service, your CICS region needs a PIPELINE resource. In this step we will add resource definitions in CICS for these resources.

Submit a job to build the VSAM file used by FUNDPROG

> Note: If you have performed a lab exercise which uses the FUNDPROG program and FUND VSAM file, this resource definition already exists in your CICS region. In this case, you can skip to step 6.

___1. From the **Remote System Explorer** perspective, the **Remote Systems** view, expand the **USER1.CICSLAB.JCL** file, **right-click** the **FUNDVSAM** member, and from the context menu, select **Submit**. Then from the **Job Submission Confirmation** dialog, click the **OK** button.

___2. From the **Remote System Explorer** perspective, the **Remote Systems** view, the **JES** node, click the **refresh** button, then **double-click** your **Job**.

___3. **Ensure** that your **FUNDVSAM** Job ran **successful**ly. Note step return codes are towards the top of the listing. They should all be 0. **Close** the Job Listing when you are done looking at it.

Define and install a FILE definition

___4. From the IBM Explorer, the **CICS SM** view, the **menu bar**, select **Definitions** > **File Definitions**. From the **File Definitions** view, **right-click** in an open area and select **New**. Then **define and save** a file definition for the **VSAM file** named **FUND**.

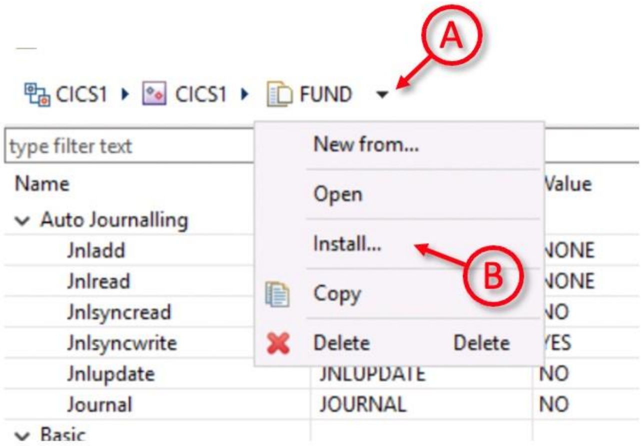
Attribute	**Value**
Group	WORKSHOP
Name	FUND
Description	The FUNDFILE file
DSNAme	USER1.CICSLAB.FUNDFILE
Open editor	Checked

From the **File Definition (FUND)** editor, **click** the **Attributes** tab (at the bottom).

Attribute	**Value**
Record Format	Fixed (use the pull-down menu to select “Fixed”)
Add	Yes (use the pull-down menu to select “YES”)
Browse	Yes (use the pull-down menu to select “YES”)
Delete	Yes (use the pull-down menu to select “YES”)

Read	Yes
Update	Yes (use the pull-down menu to select "YES")

___**5.** To **Install** the File resource, first **save** the resource (press Ctrl-S) and then click FUND in the A section (see diagram below), then click **Install**. From the **Perform INSTALL Operation** dialog, click **OK**.



Check for the message at the bottom of the Explorer; CNX0609I: Install of File Definition "FUND" into CICS1 succeeded.

Close the **File Definition (FUND)** editor.

Verify the TCPIPService definition

___**6.** A **TCPIPService** resource definition is needed to tell CICS to listen for incoming HTTP requests. **We have added a TCPIPService definition for you**, but **if** one did not exist, you would need to create one with the characteristics similar to those in the following table. Note that the TCPIPService definition we defined for you does not require security over the connection (SSL or HTTP Basic Authentication).

Field	Value
TCPIPService	HTTPPORT (some name)
Group	WORKSHOP
POrtnumber	1423
PROtocol	Http
Transaction	CWXN

Note that the TCPIPService definition we added causes the CICS region to listen on port 1423. For this series of lab exercises, we are using the same port for all HTTP-based requests. In your own CICS regions you will most likely use different ports for JSON services and browser requests depending on your application design, firewall, network configuration, etc.

Define and install a PROGRAM definition

Note: If you have performed a lab exercise which uses the FUNDPROG program and FUND VSAM file, this resource definition already exists in your CICS region. In this case, you can skip to step 9.

___**7.** From the IBM Explorer, the **CICS SM** view, the **menu bar**, select **Definitions > Program Definitions**. From the **Program Definitions** view, **right-click** in an open area and select **New**. Then **define** the following program definition. (this may already have been created in another lab)

Attribute	Value
Resource/CSD Group	WORKSHOP
Program	FUNDPROG
Program Type	Assembler, C/C++, COBOL, or PL/I

To **Install** the Program resource, first **save** the resource (press Ctrl-S) and then click the FUNDPROG pulldown, then click **Install** in the same manner as installing the FILE resource. From the **Perform INSTALL Operation** dialog, click **OK**.

Close the **Program Definition (FUNDPROG)** editor.

Define and install a PIPELINE definition

___**8.** From the **IBM Explorer**, the **Remote System Explorer** perspective, the **Remote Systems** view, **right-click** the **JSONServiceConfig.xml** file located in **/u/user1/cicslab/pipelines** and select **Open**. This configuration xml is referenced by the Pipeline definition.

___**9.** Change **JSONServiceConfig.xml** from:

From

```
<?xml version="1.0"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
  <service>
    <terminal_handler>
      <cics_json_handler_java>
        <jvmserver>DDWAXIS2</jvmserver>
      </cics_json_handler_java>
    </terminal_handler>
  </service>
  <apphandler_class>com.ibm.cicsts.axis2.CICSAxis2ApplicationHandler</apphandler_class>
</provider_pipeline>
```

To

```
<?xml version="1.0"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline">
 <service>
  <terminal_handler>
     <handler>
      <program>DFHPIJT</program><handler_parameter_list/>
    </handler>
  </terminal_handler>
 </service>
</provider_pipeline>
```

We will be updating the virtual lab environment in to avoid having to make this change.

___**10.** From the **CICS SM** perspective, from the **menu bar**, select **Definitions > Pipeline Definitions**. From the **Pipeline Definitions** view, **right-click** in an **open area** and from the context menu select **New**.

___**11.** From the **Create Pipeline Definition** pop-up dialog, **define** a new **PIPELINE** definition with the following attributes. **After** you fill in the details from the first table, click the **Finish** button to open the PIPELINE resource in an editor. **Modify** the resource definition as indicated in the second table. Note: this definition is case sensitive.

Attribute	Value
Resource/CSD Group	WORKSHOP
Name	MYJPIPEI
Description	JSON provider pipeline
Configfile	/u/user1/cicslab/pipelines/JSONServiceConfig.xml
Open editor	Checked

In the **Pipeline Definition (MYJPIPEI)** editor, from the **Overview tab** (across the bottom), **enter the following**. Be sure that your changes took effect, and **save** MYJPIPEI.

Attribute	Value
Shelf	/u/user1/cicslab/shelf/
WSBind Pickup Directory	/u/user1/cicslab/json/wspickup/provider/

___**12.** To **Install** the PIPELINE resource, first **save** the resource (press Ctrl-S) and then click the MYJPIPEI pulldown and click **Install** in the same way you installed the FILE and PROGRAM resources. From the **Perform INSTALL Operation** dialog, click **OK**.

Close the **Pipeline Definition (MYJPIPEI)** editor.

You have just created a VSAM file, and defined the program, file, and pipeline to CICS and installed the definitions.

Part 5: Install the CICS JSON Service Resources

When you ran the DFHLS2JS utility, you asked for the FundProgReqJSON.wsbind file to be placed in /u/user1/cicslab/json/wspickup/provider. Also, you created a PIPELINE definition and specified a pickup directory of /u/user1/cicslab/json/wspickup/provider. Now, we will tell CICS to SCAN (and install) the JSON services it finds in the pickup directory. After you perform the SCAN, CICS should have dynamically installed URIMAP and WEBSERVICE resources.

___1. From the IBM Explorer, the **CICS SM** perspective, from the **menu bar** select **Operations > Pipelines**, then **right-click** on the **MYJPIPEI** pipeline definition and from the context menu, select **Scan**. From the **Perform SCAN Operation** dialog, click **OK**.

Note that the SCAN just asked CICS to start a scan. The response you saw when the window went away was successful. A successful response just indicated that the start of the scan was normal. It didn't indicate that all the work you wanted the scan to complete was successful. This means that although you received a successful response, your FundProgReqJSON service may not be usable.

___2. From the IBM Explorer, the **CICS SM** perspective, from the **menu bar**, select **Operations > URI Maps**. From the **URI Maps view**, verify that your new JSON service has been installed properly. Since the entries were dynamically generated, the Name will start with a '$'.

___3. You can use the IBM Explorer, the **CICS SM** perspective, to view your CICS **Web Service** definition to verify your JSON service was properly installed.

You have just exposed your CICS COBOL Program as a JSON service.

Part 6: Test your Work

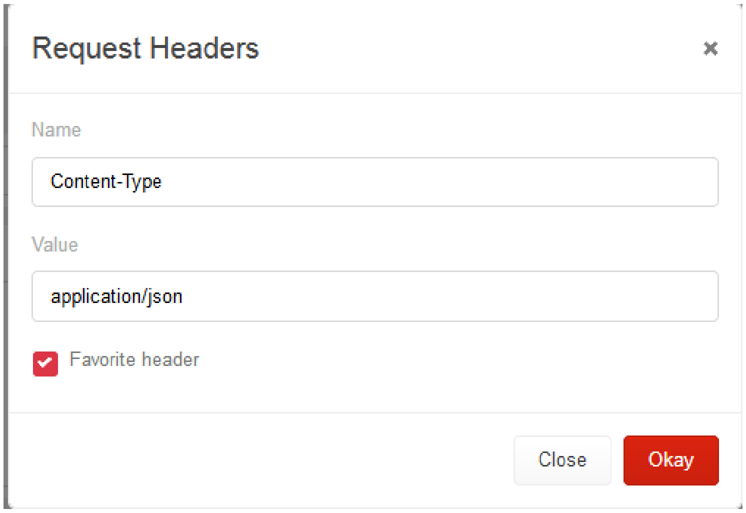
We will be testing with the RESTClient Firefox plug-in as a generic REST client.

___**1.** From the **Firefox browser**, press the button in the upper-right corner. If you hover over the button, it says "RESTClient". If you get a pop-up message regarding migrating from RESTClient 2, just close the window.

___**2.** From the RESTClient, on the **URL** line type:

http://wg31.washington.ibm.com:1423/json/fundprog

___**3.** Add a **Content Type** of **application/json.** Select **Headers > Custom Header**.



Request Headers ×

Name

Content-Type

Value

application/json

☑ Favorite header

___**4.** For the **body** of the HTTP request enter:

{"FUNDPROGOperation":{"request_type":"C","ret_code":"","fund_id":"00000008","fund_name":"Some Name","fund_rating":"A","fund_price":"1.23","return_comment":""}}

___**5.** From **RESTClient**, select the **POST** method and select **SEND**. This created the fund with an id of 00000008 and you should receive a "Response" dialog saying Status Code: 200 OK.

___**6.** Select the **Preview** tab to see the results of your POST. The server (CICS and the FUNDPROG program) has returned a response containing JSON.

___**7.** ➔➔➔ **NOTE:** you have just exposed a program with a Request-Response JSON interface. **HOWEVER**, this isn't a typical RESTful interface.

With bottom-up enablement of LINKable programs as JSON services, the only 'Action' that is supported is a **POST** request. **Otherwise**, if you try any of the other 'Actions' (methods) you will get a **405 Method Not Supported**.

Typically, with a RESTful interface, a GET request reads a record, a POST adds a record, a PUT updates a record, and a DELETE will delete a record. You will have to write a **wrapper program** to have your program follow the 'normal' RESTful interactions.

___**8.** Using this 'quick' way of exposing your program as a JSON service you can specify the following to read a record (press **SEND** again) (note that any fields that weren't entered will be empty):

{"FUNDPROGOperation":{"request_type":"**R**","fund_id":"00000008"} }

Again, note that this is illogical and does not follow RESTful conventions because the method we are using submits a **POST** request, where a RESTful API would use the **GET** method to retrieve a record.

___**9.** Using this 'quick' way of exposing your program as a REST service you can specify the following to update a record (note that the fund price is different) (press **SEND** again):

{"FUNDPROGOperation":{"request_type":"**U**","ret_code":"","fund_id":"00000008","fund_name":"Some Name","fund_rating":"A","fund_price":"4.56","return_comment": ""} }

Again, note that this simple JSON service does not follow RESTful conventions because you have used a POST request instead of the **PUT** method to update a record. Additionally, when using PUT method, you would only indicate the fields that have changed. With a Request-Response JSON service, if you omit any fields on the POST request, they will be changed to 'empty'.

___**10.** Using this 'quick' way of exposing your program as a REST service you can specify the following to delete the record (press **SEND** again):

{"FUNDPROGOperation":{"request_type":"**D**","fund_id":"00000008"} }

Again, note that this does not follow RESTful conventions because you have used a POST request instead of the DELETE method to delete a record. Note also that the program, when specifying a request type of D, only cares what the key of the record is to delete, therefore all other field values are ignored on input. Note that the FUNDPROG program always returns a COMMAREA when deleting a record.

To convert this simple Request-Response JSON service to a RESTful API, you would write a thin wrapper program to front-end the FUNDPROG program in CICS. With this wrapper program you can detect the HTTP method and follow normal REST conventions.

___**11.** Using this ‘quick’ way of exposing your program as a REST service you can specify an invalid request type (press **SEND** again):

{"FUNDPROGOperation":{"request_type":"**X**","ret_code":"","fund_id":"00000008","fund_name":"Some Name","fund_rating":"A","fund_price":"4.56","return_comment": ""} }

Again, note on the pop-up window that the **HTTP Status** is **200 OK**. This status that is always returned if the request is properly formatted.

___**12.** Remove one of the quotes and press the POST button.

You will receive a 500 Internal Server Error and a message:

```
{
   "Fault": {
      "faultstring": "Failure interacting with CICS”,
      "detail": {
         "CICSFault": “DFHPI1007 09\/13\/2023 21:20:36 CICS1 CPIH 00995 JSON to data
transformation failed because of incorrect input (UNDEFINED_ELEMENT fund_id:) for
WEBSERVICE FundProgReqJSON.”
      }
   }
}
```

___**13. Hmm** – what if your tests weren’t successful?

Your approach to debugging a problem when using CICS-based JSON service will be similar to the debugging techniques you already use for other CICS programs. Some of the areas you might check, depending on where you think the problem is, may be:

- Did the request get to CICS? If you have no network connectivity, or CICS isn’t listening, you may get a connection refused.
- Is the JSON service defined to CICS? If you didn’t specify the right path (json/fundprog in this case), CICS won’t know what to do with the request and will return an HTTP 500 response code. Likewise, if the json/fundprog JSON service hasn’t been properly defined to CICS (like maybe your SCAN didn’t do what it was supposed to, or maybe your PIPELINE definition didn’t get

properly installed), then again, CICS won’t know what to do with the request and will return an HTTP 500 response code.

- Was your target business logic program invoked? You could check the program count on your program to check if it was invoked. If your program is being invoked, then it is business as usual and you can use your favorite debugger to step through your program.
- You can also use CEDX to go through your program. By default the transaction we are running our JSON service under is CPIH, so you can ‘CEDX CPIH’ to walk your program through EDF.
- If you received a ‘return_comment: Problem adding fund, see CICS message log”, you may have defined the file incorrectly. It should be fixed length, readable, addable, updateable, and browseable.

Contact a lab instructor if you have any questions.

Part 7: Summary

Congratulations, you have run the DFHLS2JS utility and exposed a CICS COBOL program as a Web service.

In this lab you:

- Ran the DFHLS2JS utility using the COBOL program’s COMMAREA as input
- Compiled the CICS COBOL program, defined a VSAM file.
- Defined and installed the program and file in CICS
- Defined and installed a JSON Pipeline resource
- Told CICS to pickup your new Web service
- Tested your new Web service

Note that this way of exposing an existing CICS program only uses the HTTP POST method. This is not typical of a RESTful API program. Rather, the quick bottom-up process used in this lab exercise creates a simple request-response service which drives the existing application directly from the JSON Pipeline. The target application program is typical of written in the 1990s and has no knowledge of HTTP methods with only a COMMAREA as input. In another lab exercise you will implement a thin wrapper program in CICS that can respond to all of the usual HTTP methods and provide a typical RESTful interface.

If you have any questions about anything you did in this lab exercise, please contact one of the lab instructors.