



L93 - CICS-Java REST services using JAX-RS

Lab Version V61.04.zVA

June 28, 2024

Please send any comments on this lab exercise to:
Steve Fowlkes, Eric Higgins, or Leigh Compton

fowlkes@us.ibm.com

erichiggins@us.ibm.com

lcompton@us.ibm.com

Overview

CICS supports Java in the Liberty profile, including JAX-RS (Java Extension for RESTful Services). The Liberty profile provides a non-OSGi-based Java environment. You can provide REST from a CICS application under the Liberty Profile without using JAX-RS. This lab exercise covers JAX-RS. JAX-RS will return XML, JSON, or anything as a REST response.

This lab exercise will have you compose a non-OSGI Java front-end program that provides RESTful responses carrying a message in JSON format (using JSON4J). The Java front-end program will invoke our business logic, also written in Java. This program will invoke a backend business logic program written in COBOL (we are invoking a CICS COBOL program because executing Java business logic is as straightforward as invoking a method on a class).

The purpose of this exercise is to illustrate the use and preparation of a program that uses JAX-RS and the JSON4J API in a CICS-based Liberty profile as well as invoking a CICS business logic program written in some language other than Java.

Scenario

In our scenario, you are a programmer at a mutual fund company. You want to write a quick REST service in Java (to run in CICS), you will use the Java RESTful API (JAX-RS) along with JSON4J to build a ‘front-end’ to the business logic written in COBOL.

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, 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 should work fine in any supported release of CICS TS, but in this lab environment, we use CICS TS V5.6. The directions have you build the application in a single region environment. In our lab environment, each team performing the lab exercise has their own CICS region, to include their own CSD and other supporting CICS files.
- **Login:** A TSO userid is available with appropriate passwords are provided, however the directions for this lab have you using the TSO userid with the IBM Explorer.
- **IBM Explorer:** In the lab environment we have installed the latest IBM Explorer for z/OS with the CICS Explorer plug-in.

Lab Step Overview

Part 1: Configuring the IBM Explorer and the Remote System Explorer

Although you can complete the lab without using the IBM Explorer, in this part of the lab, you will configure the IBM Explorer. The same is true for the Remote System Explorer.

Part 2: Prepare the back-end FUNDPROG program and define the VSAM file

The business logic program is written in COBOL. We will invoke this COBOL program using a Java program that interprets REST requests.

Part 3: Prepare the IBM Record Generator for Java classes

IBM Record Generator for Java is a separate, no charge feature of Java that can take a COBOL COMMAREA and generate a Java class with getters and setters for each field. This is a quick, simple way to bridge the series-of-bytes world of CICS and the getter/setter world of Java.

Part 4: Create a project and create a JAX-RS Servlet

In this part of the lab, you will create a Dynamic Web Project and put a JAX-RS Servlet into it.

Part 5: Test the application on your desktop Liberty profile

This part of the lab exercise you will verify that you have coded the RESTful Servlet correctly by executing the RESTful Servlet on the Liberty profile we have installed on your desktop.

Part 6: Define a CICS JVM server with a Liberty profile

In this step you will define a JVM server that will contain your Liberty profile.

Part 7: Define and export your Java project to z/OS

In this part of the lab exercise you will define a CICS bundle on your workstation, insert the Java project into the bundle, and place it on z/OS UNIX System Services.

Part 8: Define and install a CICS Bundle definition

We define a bundle definition to CICS and point it at the bundle we placed on z/OS UNIX System Services in the previous step.

Part 9: Test the CICS-based REST Servlet

In the part we will test the REST Servlet from a browser, and from the RESTClient Firefox add on.

Part 10: Summary

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

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, if it is not already running.



- ___2. **IF** you are prompted for a workspace, from the **Select a workspace** dialog click the **OK** button to select the default.
- ___3. If the Explorer shows you a **Welcome page** click the **Workbench icon** in the upper-right corner to go to the workbench. Then **maximize** the IBM Explorer window.



Verify that you have a z/OS Remote System connection 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: Prepare the Backend FUNDPROG Program

In this part of the lab exercise you will copy the FUNDPROG program and associated copybook (FUNDFILE) from your workstation to z/OS. You will also compile the FUNDPROG program. **Note** that these files may already exist in that dataset from a previous lab exercise.

This is the ‘backend’ business logic program written in COBOL. We have decided to write a JSON service interface to this COBOL program in Java.

Note: CICS’s Java JSON service support, or running Java in CICS is not dependent on other languages such as Java or C. In this lab exercise we have arbitrarily decided to interface with an existing, non-Java business-logic program (in COBOL). We took this approach because while converting your application to Java, due to the size of your applications, you may not be able to convert all of your application at one time. The use of an existing COBOL application in this lab exercise just shows it can be done. The LINK to a CICS COBOL program could be replaced with accessing DB2 in Java, for example.

Note: If you have used the FUNDPROG program and its file in another exercise, you do not need to create these artifacts again. You can skip to Part 3.

Compile the FUNDPROG JCL

- ___1. From the **Remote Systems** view, **right-click** on **USER1.CICSLAB.JCL(FUNDPROG)** and from the context menu select **Submit**.
- ___2. Use the **Remote Systems** view, the **JES** node to verify that the Job ran successfully.

Create the FUNDVSAM file

- ___3. From the **Remote Systems** view, **right-click** on **USER1.CICSLAB.JCL(FUNDVSAM)** and from the context menu select **Submit**.
- ___4. Use the **Remote Systems** view, the **JES** node to verify that the Job ran successfully.

Part 3: Prepare the IBM Record Generator for Java classes

Our ‘backend’ business-logic program is written in COBOL. We will be writing a Java-based web service that interfaces with that program. To LINK to the COBOL program, the Java web service must create a FUNDPROG COMMAREA before it LINKs to the FUNDPROG program.

In this part of the lab exercise we will generate a Java class that represents the FUNDPROG COMMAREA. This class will contain getter and setter methods for each field plus a `getByteBuffer()` method to access the series of bytes that represents the field-oriented COMMAREA we will pass on the LINK command.

We will use the IBM Record Generator for Java classes to generate a Java class that represents this COMMAREA. The Java program will instantiate the generated class (mentioned in the previous paragraph), use its setters to place values in the generated COMMAREA structure, then pass the COMMAREA object when the Java program LINKs to the backend COBOL program named FUNDPROG.

- ___1. From the **Remote Systems** perspective, **Remote Systems** view, **right-click** on **USER1.CICSLAB.JCL(UPDJAVA3)** and from the context menu select **Submit**.
- ___2. Use the **Remote Systems** view, the **JES** node to verify that the Job ran successfully (00 return codes)
- ___3. Double click on **USER1.CICSLAB.JCL(UPDJAVA3)** member to open in an editor and examine the content. Note on the EXEC COBGENIT statement, that PGMNAME=FUNDPROG and that PGMLOC=USER1.CICSLAB.UTIL. This is our backend FUNDPROG program that contains the COMMAREA layout, and that our Java wrapper program will invoke. The COMMAREA was the data structure used to generate the Java class.

Note that in the DATAIN DD statement (towards the bottom of the member), that the IBM Record Generator for Java will be invoked. The 01 level we want to turn into a Java class is the ‘symbol’ DFHCOMMAREA. The generated class will have a name of FUNDPROG_CommArea which will be placed in a package named `com.library.cobol.records`, which will be placed in your z/OS UNIX directory named `~/cicslab/cobgen` where the `~` indicates your home directory.

- ___4. **Close the UPDJAVA3.jcl editor.**
- ___5. Note that this Job generated Java source code so in the next few steps we will compile the Java class into a .class file.

If you would like to look at the generated Java code, it is
`/u/user1/cicslab/cobgen/com/library/cobol/records/FUNDPROG_CommArea.java`

Tech-Tip: For those not familiar with Java...

The name of the fully qualified Java source code that was generated is `com.library.cobol.records.FUNDPROG_CommArea.java`. The name of the Java source code is `FUNDPROG_CommArea.java`. Java source files are organized into 'packages'. The first part of the fully qualified name is the package name (`com.library.cobol.records`). Each level of the package equates to a directory on the file system.

This means that if someone told you to look at the `com.library.cobol.records.FUNDPROG_CommArea.java` program in your `~/cicslab/cobgen` directory, you would look for
`<Home_Directory>/cicslab/cobgen/com/library/cobol/records/FUNDPROG_CommArea.java`

Note that there is a 'package' statement in `FUNDPROG_CommArea.java` that specifies that it resides in the `com.library.cobol.records` package. Therefore the Java file -must- be in the specified directory structure.

Part 4: Create a Project and Create a JAX-RS Servlet with JSON4J

In this part of the lab exercise we will use the IBM Explorer to create a Dynamic Web Project in an Enterprise Application Project. These projects will indicate JAX-RS, JSON4J, and IBM Record Generator for Java capabilities. We will then add a pre-written Java program (similar to one that you might write) to the project.

We will initially test your Java program on your desktop (we have used the IsCICS class (part of the JCICS API) so that the application knows when it is running in a CICS environment or somewhere else). When we run the program on Windows the IsCICS class will cause a dummy record to be returned (and not LINK to the CICS program). In a later step, when we run the program on CICS, the program will know it is in a CICS environment and will LINK to the COBOL program to return a record from the VSAM file.

Running the program in Windows will provide immediate feedback on our Java program (easier, quicker development), however we will have to run the program under the Liberty profile running in CICS to verify the LINK to the COBOL program is returning everything properly. On our workstation we can add 'println' statements to our code for debugging and could use the Java debugger if we wanted. When we move to CICS most of the program will already be debugged and we only need to verify the LINK command returns properly. From CICS we can debug our program by using CEDX, writing messages to the log, or using the source-line debugger. We can use the Java source-line debugger in CICS, but only one person at a time may use the Java Debugger in a CICS region, so depending on how many developers you have using the CICS region, it may be most efficient to debug most of your application from your desktop.

Create a Java Dynamic Web Project

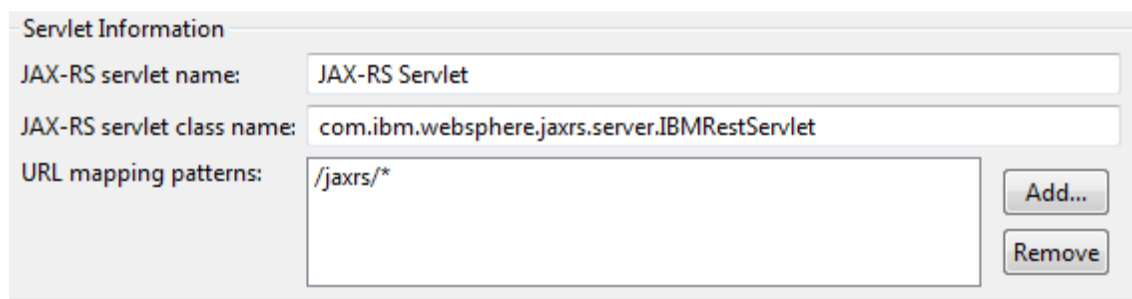
- ___1. In the **IBM Explorer**, open (or switch to) a **Java EE perspective**.
- ___2. In the **IBM Explorer**, the **menu bar**, select **File > New > Dynamic Web Project**.
- ___3. From the **Dynamic Web Project** dialog, fill in the following and press the **Modify** button to the right of Configuration.

Field	Value
Project name	com.ddw.sample.jaxrs.json4j.service
Target runtime	Liberty Runtime
Dynamic web module version	4.0
Configuration	<custom>
Add project to an EAR	NOT checked

- ___4. From the **Project Facets** dialog, **check** the box next to **JAX-RS (REST Web Services)** and **select 2.1** from the drop-down, then click the **OK** button.

- ___5. **Back** on the **Dynamic Web Project** dialog, click the **Next** button.
- ___6. From the **New Dynamic Web Project** dialog, the **Java** page, click the **Next** button.
- ___7. From the **New Dynamic Web Project** dialog, the **Web Module** page, **check** the box next to **Generate web.xml deployment descriptor**, and press the **Next** button.
- ___8. From the **New Dynamic Web Project** dialog, the **JAX-RS Capabilities** page, **check** the box next to **Update Deployment Descriptor**.
- ___9. Ensure that JAX-RS servlet class name is **com.ibm.websphere.jaxrs.server.IBMRestServlet**.

Note the **URL mapping patterns** value, then, click the **Finish** button.



Servlet Information

JAX-RS servlet name: JAX-RS Servlet

JAX-RS servlet class name: com.ibm.websphere.jaxrs.server.IBMRestServlet

URL mapping patterns: /jaxrs/*

Add... Remove

Add some tags to the web.xml

- ___10. From the **Project Explorer** view, **double-click** on the **com.ddw.sample.jaxrs.json4j.service > WebContent > WEB-INF > web.xml** file to open the file in an editor.
- ___11. Click the **Source** tab (bottom of the editor), then **Add** the following lines right **after** the **</servlet-class>** tag


```
<init-param>
  <param-name>javax.ws.rs.Application</param-name>
  <param-value>com.ddw.sample.jaxrs.json4j.service.FundProgRestApplication</param-value>
</init-param>
```
- ___12. Delete the lines containing **<welcome-file-list>** thru **</welcome-file-list>**.
- ___13. Change the **<url-pattern>** value to **/data/***

This changes the second part of the HTTP 'path' (i.e. to **/.../data/***).

Note that we could have changed this in the wizard we used to create the project, but I wanted you to edit the web.xml file.

___ **14. Save and close** the web.xml file.

Note that you may have some **errors**. This ok for now, we need to add more files.

In the web.xml file, in the <servlet-mapping> we told the environment that the URL pattern would invoke the “JAX-RS Servlet”. In a <servlet> tag, we indicated that the “JAX-RS Servlet” had an initial class named com.ibm.websphere.jaxrs.server.IBMRestServlet. This is an IBM-supplied program. In the <init-param> tag we passed a value of com.ddw.sample.jaxrs.json4j.service.FundProgRestApplication.class (this Class must extend javax.ws.rs.core.Application).

Add two classes to your project that use JAX-RS

- ___ **15.** From the **Project Explorer** view, **expand** your **com.ddw.sample.jaxrs.json4j.service** project, then expand **Java Resources > src**.
- ___ **16.** From the **Project Explorer** view, **right-click** on **src**, and from the context menu, select **New > Package**.
- ___ **17.** From the **New Java Package** dialog ensure the **Name** is **com.ddw.sample.jaxrs.json4j.service**, then click the **Finish** button.
- ___ **18.** From the **Project Explorer** view, **right-click** on the **com.ddw.sample.jaxrs.json4j.service** package and from the context menu select **New > Class**.
- ___ **19.** From the **New Java Class** wizard, enter a **name** of **FundProgRest** and click the **Finish** button.
- ___ **20.** From the **Remote System Explorer** perspective, the **Remote Systems** view, **double-click** on **C:\CICSLAB\CICS-Java-RESTinterfaceToCOBOL\FundProgRest.java** to open it in the editor. Then press **Ctrl-A** to mark all the lines and **Ctrl-C** to copy the lines to the clipboard. **Close** the editor.
- ___ **21.** **Back** in the **Java EE** perspective, **replace** the contents of the **FundProgRest.java** file by pressing **Ctrl-A** then **Ctrl-V** (to paste).
- ___ **22.** **Save and close** the FundProgRest.java editor. Note the file will contain ‘errors’.
- ___ **23.** From the **Project Explorer** view, **right-click** on the **com.ddw.sample.jaxrs.json4j.service** package and from the context menu select **New > Class**.
- ___ **24.** From the **New Java Class** wizard, enter a **name** of **FundProgRestApplication** and click the **Finish** button.
- ___ **25.** From the **Remote System Explorer** perspective, the **Remote Systems** view, **double-click** on **C:\CICSLAB\CICS-Java-RESTinterfaceToCOBOL\FundProgRestApplication.java** to

open it in the editor. Then press **Ctrl-A** to mark all the lines, then press **Ctrl-C** to copy the lines to the clipboard. **Close** the editor.

- ___26. **Back** in the **Java EE** perspective, **replace** the contents of the **FundProgRestApplication.java** file by pressing **Ctrl-A** and **Ctrl-V** (to paste).
- ___27. **Save** and **close** the FundProgRestApplication.java editor.

The FundProgRestApplication, which extends javax.ws.rs.core.Application, is basically a router program. The com.ibm.websphere.jaxrs.server.IBMRestServlet program looks into the FundProgRestApplication to see which class or classes to invoke. In FundProgRestApplication we added the name of the 'real' program to get control into a HashSet using an add() method (this program name is com.ddw.sample.jaxrs.json4j.service.FundProgRest.class). So, the FundProgRestApplication is basically a router program.

The FundProgRest program will respond to GET and POST requests when it receives a path with 'rest' in it (i.e. /com.ddw.sample.jaxrs.json4j.service/data/rest)

Add the IBM Record Generator for Java class you developed in Part 3

You generated the IBM Record Generator for Java class from the COBOL program's COMMAREA in an earlier step. We will copy the generated Java code into your project so that the Java program can easily interface with the COBOL program.

- ___28. From the **Remote System Explorer** perspective, the **Remote Systems** view, navigate to **z/OS UNIX Files > My Home > cicslab > cobgen > com > library > cobol > records** and **double-click** on **FUNDPROG_CommArea.java** to open it in an editor. Then in the editor press **Ctrl-A** and **Ctrl-C** to copy the contents to the clipboard. **Close** the editor.
- ___29. From the **Java EE** perspective, the **Project Explorer** view, expand your **com.ddw.sample.jaxrs.json4j.service** project, then **right-click** on the **Java Resources > src** directory, and from the context menu select **New > Package**. On the **New Java Package** dialog, enter a **name** of **com.library.cobol.records** and press the **Finish** button.
- ___30. From the **Java EE** perspective, the **Project Explorer** view, **right-click** on your **com.library.cobol.records** package, and from the context menu, select **New > Class**. In the

New Java Class dialog specify a **name** of **FUNDPROG_CommArea**, then press the **Finish** button. An editor opens.

- ___ **31.** In the **FUNDPROG_CommArea.java** editor, replace all of the current text by pressing **Ctrl-A**, then **Ctrl-V** (to paste).
- ___ **32.** **Save** and **close** the **FUNDPROG_CommArea.java** editor (this class shows errors).

Note you have just copied the contents of **FUNDPROG_CommArea.java** from z/OS to a file in your project.

The **FUNDPROG_CommArea** class is the **COMMAREA** representation that you developed in Part 3. This class was developed using the IBM Record Generator for Java utilities by running a batch Job on z/OS. This job analyzed a COBOL copybook and generated a String equivalent for each field (you can have other datatypes other than String, but our copybook only had data elements of type PIC X).

You can fill in the values on the generated class by using the **setByteBuffer()** class and passing it the contents of the **COMMAREA** or **VSAM** record or any series of bytes you receive from **CICS**. In the Java program you can access the individual fields with getters and setters.

IBM offers two techniques to work similarly, the Java Class Records in the IBM Record Generator for Java package, and the **CICS Java Data Binding** that is part of Rational Application Developer (RAD). You submit a batch job on z/OS to develop the IBM Record Generator for Java Java Class Record. You run a wizard under RAD to develop **CICS Java Data Bindings**. There is no charge for IBM Record Generator for Java. The generated classes from both products function very similarly.

Fix the error indicators on your project...

- ___ **33.** From the **Project Explorer** view, **right-click** on your **com.ddw.sample.jaxrs.json4j.service** project, and from the context menu, select **Build Path > Configure Build Path**.
- ___ **34.** From the **Properties for com.ddw.sample.jaxrs.json4j.service** dialog, **on the left** select **Java Build Path**, and **on the right** select the **Libraries** tab (across the top right).
- ___ **35.** Then on the **far-right** of the dialog, select **Add Library** and from the **Add Library** dialog, select **CICS with Enterprise Java and Liberty** and click the **Next** button. Choose **CICS TS 6.1** from the drop-down menu. Ensure that **Java EE and Jakarta EE 8** is selected and click **Finish**.
- ___ **36.** Again, **back** on the **far-right** of the **Properties for com.ddw.sample.jaxrs.json4j.service** dialog, press **Add Library** and from the **Add Library** dialog, select **IBM JZOS Toolkit Library** and click the **Next** button. Ensure that Version **8.0** is selected and click **Finish**.
- ___ **37.** **Back** on the **Properties for com.ddw.sample.jaxrs.json4j.service** dialog, press the **Apply and Close** button.

NOTE that this should remove all of the error indicators in your project. If you still have errors consult a lab assistant.

Note: we have added some files to the build path. This allows Eclipse to cleanly compile the Java classes. We have just pointed at the classes, and haven't included them into our project. This means that although Eclipse was able to compile our project, when we want to execute our project on our desktop or in CICS, we will have to have our runtime environment include the packages again.

The program logic

The FundProgRest program will respond to GET, POST, PUT, and DELETE requests when it receives a path with '/rest' in it (i.e. /jaxrsjson4j/data/rest).

In the FundProgRest.java program, look at the GET method (line 31). Note the annotations available in JAX-RS. The GET method is marked with @GET. The @Path requires a key (for example /jaxrsjson4j/data/rest/000000001). The @Produces tells the program that it should send "application/json" in the Content-Type header. The 'key' parameter is required as input to the getFund() method.

In line 36 we instantiate the COMMAREA object we developed using the IBM Record Generator for Java classes on z/OS.

In line 42 we invoke the CICS program. In the invokeCICSFUNDPROGProgram() (line 217), we use the IsCICS class (available with the JCICS classes). This class will indicate whether we are running in a CICS environment. If we are in a CICS environment we invoke the COBOL backend program. If we are not running in CICS, this method returns a dummy record (the 'else' part of the 'if' statement). This allows us to be very productive, developing and doing some testing of our code in the desktop Liberty profile environment. Then we only have to test our application on CICS when we are ready to interact with CICS.

Back towards the top of the program, in line 50 we test the return code in the COMMAREA returned by the COBOL program (or the dummy record). If we receive a '00' in the COMMAREA, the record was retrieved, so we use the JSON4J classes to encapsulate the output in a JSON format. Around line 267 we use the JSON4J classes to take the values from the COMMAREA object and change them into a string of JSON. The COMMAREA on our simple program doesn't contain many fields, your program will likely contain many more fields.

Part 5: Test the application on your desktop Liberty server

In this section, you will start the Liberty profile on your desktop. You will

- Configure the Liberty profile on your desktop to support JAX-RS
- Start the Liberty profile on your desktop

- Test your application (which will return a dummy record, since we aren't yet on CICS)

Note that we have 'dummied' out the calls to the backend COBOL program by using the IsCICS class. The IsCICS class will return true if we are running in CICS, and will return false if we are not in a CICS environment. We have done this so we can test your program on the Liberty profile on your desktop. Although you are being provided with code, if you were writing the code, it would be much easier to write and test most of your code on the more productive desktop test environment (where you can easily code, test, code, test, etc). Later, you can execute your program under CICS to test the code specific to CICS. Having a desktop version of the Liberty may be a very useful environment - unless you happen to be a perfect Java coder and do everything right the first time.

In this section we will configure the Liberty profile running on your desktop to support JAX-RS, JSON4J, and IBM Record Generator for Java, and, we will test your program (under the Liberty profile running on your desktop).

Configure the Liberty Profile in your desktop environment

1. From the **Java EE** perspective, the **Servers** view, **Stop** your Liberty Server, if it is **started**.
2. From the **Java EE** perspective, the **Servers** view, **right-click** on **Liberty Server at localhost** and from the context menu select **Add and Remove**.
3. From the **Add and Remove** dialog **ensure** your **com.ddw.sample.jaxrs.json4j.service** project is **on the right side** (under Configured).

Note: for configuration of the desktop Liberty profile, having your project on the right side indicates that you project will be started under the Liberty profile.

4. **Still** on the **Add and Remove** dialog, click the **Finish** button.
5. From the **Java EE** perspective, the **Servers view**, expand **Liberty Server at localhost**, then expand **Server Configuration**, and **double-click** on **Feature Manager**.
6. From the **server.xml** editor, the **Feature Manager** section, ensure that feature list includes **json-1.0**, and **jaxrs-2.1**.
7. **Still** in the **server.xml** editor, **add the following** (this is how to add Jar files to the CLASSPATH):

```
<library id="aClasspath1">
    <fileset dir="C:\CICSLAB\Java\JARs\" includes="jzos_recgen.jar" />
</library>
<library id="aClasspath2">
    <fileset dir="C:\CICSLAB\Java\CICSJars\" includes="com.ibm.cics.server.jar" />
</library>
```

- ___ 8. Still in the **server.xml** editor, add the following inside the **webApplication** tag. **NOTE: You are just adding the “classloader” parm to the existing entry** (to have your application recognize the CLASSPATH):

Note: for the above, remove the “/” in the ‘<webApplication’ line (2nd character from the end, then from the end of the line press Enter, then enter the last two lines below.

```
<webApplication
    id="com.ddw.sample.jaxrs.json4j.service"
    location="com.ddw.sample.jaxrs.json4j.service.war"
    name="com.ddw.sample.jaxrs.json4j.service">
    <classloader commonLibraryRef="aClasspath1,aClasspath2"></classloader>
</webApplication>
```


- ___ 9. Save and close the **server.xml** file.

Start the Liberty Profile

- ___ 10. Back on the **Servers** view, **right-click Liberty Server** and from the context menu, select **Start** (it may take a couple of seconds to start).

Try your application

- ___ 11. Start the **Firefox** browser.

- ___ 12. From the **Firefox browser**, press the  button in the upper-right corner. If you hover over the button, it says “RESTClient”.

- ___ 13. From the **RESTClient dialog**, enter a URL of <http://localhost:9080/com.ddw.sample.jaxrs.json4j.service/data/rest/00000001>

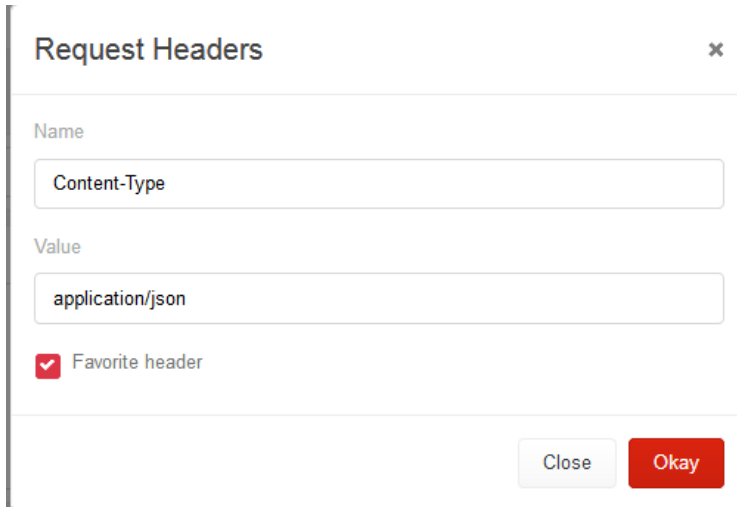
and click the **SEND** button. Ensure the Method is **GET**. (the last part was 7 zeros and a 1)

- ___ 14. You should see a “Response” with a status ‘200 OK’. Click the “Preview” tab to see the results. Similar to the following.

```
{ "id": "0000001", "price": "1.23", "name": "Some Fund", "rating": "A", "comment": "Dummy call to CICS" }
```

If you don't see the results of your query, consult a lab instructor.

- ___15. From the **RESTClient** dialog, add a **Content-Type** of **application/json**. Select **Headers > Custom Header**.



Then enter the following in the “Body” section.

```
{ "name":"Fred","price":"1.23","rating":"A" }
```

Then with a URL of

<http://localhost:9080/com.ddw.sample.jaxrs.json4j.service/data/rest/00000001>, select the **POST** method, then click the **SEND** button to add the data.

- ___16. In the “Headers” tab, you should see a status ‘201 Created’ and the words in the “Preview” tab.

```
{"id":"0000001","price":"1.23","name":"Some Fund","rating":"A","comment":"Dummy call to CICS"}
```

If you don't see the results of your query, consult a lab instructor.

Part 6: Define a CICS Java Server with the Liberty Profile

In this step you will perform the steps to start a Liberty profile running under CICS.

Note that you can define a Liberty profile and use it for Servlets, REST, and web services.

Note that you **may have already defined a Liberty profile** in CICS, and can use the existing Liberty profile for this lab exercise.

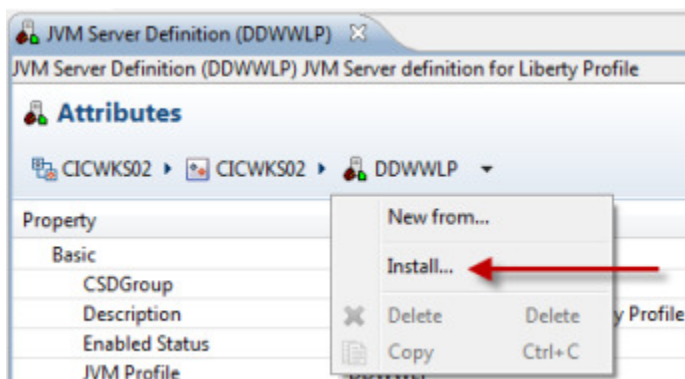
➔ **If you already have a Liberty Profile named DDWWLP using a JVM profile named DDWWLP – then skip to step 8 in this section.**

Define and Install a JVMServer into CICS

- ___1. From the **IBM Explorer**, the **CICS SM** perspective, from the **menu bar** click on **Definitions > JVM Server Definitions**.
- ___2. **Right-click** in the **open area** in the JVM Server Definitions panel and select **New**.
- ___3. In the **New JVM Server Definition** dialog, specify the following, then click the **Finish** button.

Field	Value
Resource Group	WORKSHOP
Name	DDWWLP
Description	JVM Server Definition for Liberty Profile
Enabled Status	ENABLED
JVM Profile	DDWWLP
Open editor	Checked

- ___4. From the **JVM Server Definitions (DDWWLP) editor**, under **Attributes**, click the down arrow after your new JVM Server (DDWWLP) and from the context menu select **Install**.



- ___5. From the **Perform INSTALL Operation** pop-up dialog, click the **OK** button.
- ___6. **Close** the **JVM Server Definition (DDWWLP) editor**.
- ___7. From the **CICS SM** perspective, the **menu bar**, select **Operations > Java > JVM Servers** to open a **JVM Servers** view, and **verify** that your DDWWLP JVMServer is **ENABLED**.

Note that you may need to press 'refresh' to see this new definition.

Note that we have added the `-Dcom.ibm.cics.jvmserver.wlp.autoconfigure=true` parameter (the default is false). This parameter causes CICS to look for the `server.xml` file and if it is not present, CICS will create a default `server.xml`.

Define and install a FILE and PROGRAM CICS resource definitions

- ___8. 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, save, and install** a file definition for the **VSAM file** named **FUND**. (this may already be created if you did a prior lab that used the file)

Attribute	Value
Group	WORKSHOP
Name	FUND
Description	The FUNDFILE file
DSName	<zOS_userid>.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")

- ___9. 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 and install** the following program definition. (This resource may already be created if you used it in a previous lab exercise.)

Attribute	Value
Group	WORKSHOP
Program	FUNDPROG
Language	Cobol

Modify server.xml to include the JAX-RS 2.1 feature

- ___10. From the **Remote System Explorer** perspective, the **Remote Systems** view, **double-click** on **z/OS UNIX Files > My Home > cicslab > logs > CICS1 > > DDWWLP > wlp > usr > servers > defaultServer > server.xml** file. In the `<featureManager>` tag, add the following:

```
<feature>jaxrs-2.1</feature>  
<feature>json-1.0</feature>
```

Note: These features were likely added by default earlier.

Save and close **server.xml**.

Your running Liberty profile should automatically pick up the change.

Part 7: Define and Export a CICS Bundle to z/OS

When deploying a Java REST program to CICS, you define a CICS BUNDLE and place your REST program into the BUNDLE. Then, you export the CICS Bundle from your workstation to z/OS where it can be accessed by your CICS region.

Create a CICS BUNDLE containing your Java REST program on your workstation

- ___1. From the **IBM Explorer**, switch to a **Java EE** perspective, and from the **menu bar**, click **File > New > Other**, and from the Select a wizard dialog select **CICS Resources > CICS Bundle Project**. Click the **Next** button.
- ___2. From the **CICS Bundle Project** dialog, type in a name of **com.ddw.sample.jaxrs.json4j.serviceBUNDLE**, then click the **Finish** button.
- ___3. Right-click your **com.ddw.sample.jaxrs.json4j.serviceBUNDLE** project, select **New > Dynamic Web Project Include**.

- ___ 4. In the **Dynamic Web Project Include** dialog, **highlight** your **com.ddw.sample.jaxrs.json4j.service** project in the top section. Then type in a **JVM Server** name of **DDWWLP**, then click the **Finish** button.
- ___ 5. Close the **com.ddw.sample.jaxrs.json4j.serviceBUNDLE** CICS BUNDLE manifest editor.

Export your CICS Bundle to UNIX System Services on z/OS

- ___ 6. In the **Project Explorer** view, **right-click** on your **com.ddw.sample.jaxrs.json4j.serviceBUNDLE** project and from the context menu, select **Export Bundle Project to z/OS UNIX File System**.
- ___ 7. In the **Export to z/OS UNIX File System** pop-up dialog, **click** the radio button that says **Export to a specific location in the file system**, and click **Next**.
- ___ 8. If you have a connection, but not signed on, select the **drop-down** to the right of **Connection** and choose your z/OS Connection to sign on.
- ___ 9. In the **Export Bundle** pop-up dialog, in **Bundle project** it should say **com.ddw.sample.jaxrs.json4j.serviceBUNDLE**
- ___ 10. Still on the **Export Bundle** page, in the **Parent Directory**, ensure it says **</u>user1/cicslab/bundles/**
- ___ 11. Still on the **Export Bundle** page, in **Bundle Directory**, ensure it says **/u/user1/cicslab/bundles/com.ddw.sample.jaxrs.json4j.serviceBUNDLE_1.0.0**
- ___ 12. Still on the **Export Bundle** page, click the **Finish** button

Part 8: Define and Install a CICS Bundle Definition

In this part of the exercise you will define and install a CICS Bundle Definition for your REST Java program into your CICS region.


- ___ 1. From the **IBM Explorer**, the **CICS SM** perspective, from the **menu bar**, select **Definitions > Bundle Definitions**, and from the Bundle Definitions view, **right-click** in an open area and select **New**.
- ___ 2. From the **Create Bundle Definition** dialog, **enter the following** and press **Finish**. (note that you can press the Browse button to the right of the Bundle Directory to locate the bundle directory)

Field	Value
Resource/CSD Group	WORKSHOP
Name	BUREST93
Description	REST Servlet with JAX-RS, JSON4J, and IBM-RecGen
Bundle Directory	/u/user1/cicslab/bundles/ com.ddw.sample.jaxrs.json4j.serviceBUNDLE_1.0.0 (➔ all on one line, use the Browse button)
Open editor	Checked

- ___3. From the **Bundle Definition(BUREST93)** editor, under **Attributes**, click the **down-arrow** to the right of **BUREST93** (on the top), and select **Install**.
- ___4. From the **Perform INSTALL Operation** pop-up dialog, click **OK**.
- ___5. Close your **Bundle Definition(BUREST93)** editor.
- ___6. From the menu bar, select **Operations > Bundles**, then verify that the BUREST93 BUNDLE is **ENABLED**. **Note** that if your Bundle view was already open, you will have to click the refresh button (🔄).

Part 9: Test the CICS-based REST Servlet

In this part you will use a web browser to access your CICS-based REST Servlet. Firefox web browser can display your REST Servlet.

- ___1. Start the **Firefox** browser.
- ___2. From the **Firefox browser**, press the  button in the upper-right corner.
- ___3. From the **RESTClient**, enter a URL of <http://wg31:1424/com.ddw.sample.jaxrs.json4j.service/data/rest/00000001>

Ensure that the **GET** method is selected, then click the **SEND** button). (the last part was 7 zeros and a 1)

If this is the first time you have done anything with the **FUND VSAM** file you may get a messages saying { “code”:”04”,”comment”:”Fund not found” } in the “Preview” tab.

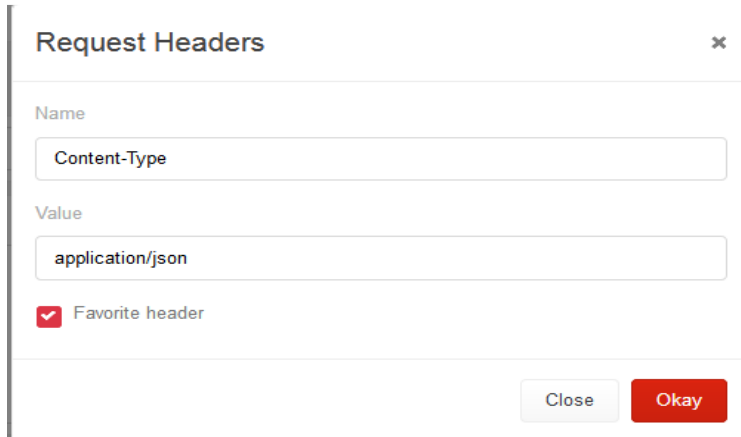
If you get the above message, on the “Preview” tab, skip step 4.

- ___4. You should see a status '200 OK' in the "Headers" tab and the following in the "Preview" tab.

```
{"id":"00000001","price":"1.23","name":"Some Fund","rating":"A","comment":"Fund information retrieved"}
```

If you don't see the results of your query, consult a lab instructor.

- ___5. From the **RESTClient**, add a **Content-Type** of **application/json**. Select **Headers > Custom Header**.



Enter the following in the "Body" section

```
{ "name":"Fred","price":"1.23","rating":"A" }
```

Then with a URL of **wg31:1424/com.ddw.sample.jaxrs.json4j.service/data/rest/00000002**

Select the **POST** method and click **SEND**.

- ___6. You should see '201 Created' in the "Headers" tab and the following in the "Preview" tab:

```
{"id":"00000002","price":"1.23","name":"Fred","rating":"A","comment":"Fund added "}
```

- ___7. You can try the GET (retrieve), PUT (update), and DELETE (delete) options

If you don't see the results of your query, consult a lab instructor.

Part 10: Summary

Congratulations, you have created a REST program in Java using JAX-RS, IBM Record Generator for Java, JSON4J, and JCICS to run under the Liberty profile under CICS.

In this lab you performed the following steps:

- Defined a Dynamic Web Project in The IBM Explorer
- Coded the RESTful Servlet
- You tested the RESTful Servlet in the Liberty profile on your desktop
- You define connections to z/OS and your CICS region to your IBM Explorer
- You defined a CICS JVMServer with a JVMProfile that invoked the Liberty profile
- You defined a CICS BUNDLE resource and included your Dynamic Web Project in the BUNDLE
- You tested the RESTful Servlet under CICS