



## **L92 – CICS-Java REST services using JAX-RS (non-OSGi)**

*Lab Version V61.01.zVA*

---

***September 26, 2023***

Please send any comments on this lab exercise to:  
Leigh Compton and Steve Fowlkes  
[fowlkes@us.ibm.com](mailto:fowlkes@us.ibm.com)  
[lcompton@us.ibm.com](mailto:lcompton@us.ibm.com)

## **Overview**

CICS supports Java in the Liberty profile, as well as JAX-RS (Java Extension for RESTful Services). The Liberty profile provides a non-OSGi-based Java environment, or an OSGi-based Java environment. You can provide REST from a CICS application under the Liberty Profile with or 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 very small hard-coded message in JSON format. The Java front-end program will invoke our business logic, also written in Java. Our simple scenario won't have a Java backend (because this is a simple scenario).

The purpose of this exercise is to illustrate the use and preparation of a program that uses JAX-RS in a CICS-based Liberty profile.

## **Scenario**

In our scenario, you are a programmer at a mutual fund company. You want to write a quick REST service in CICS to verify the Liberty profile.

## **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 CICS 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 all supported versions CICS TS, but in the lab environment, we are using CICS TS V6.1. 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:** TSO userids are available with appropriate passwords are provided, however the direction have you using the TSO userid with the z/OS Explorer.
- **The CICS Explorer:** In the lab environment we have installed the CICS Explorer with the JEE plug-in.

## **Lab Step Overview**

### **Part 1: Configure the CICS Explorer and the Remote Systems Explorer**

Although you can complete the lab without using the CICS explorer, in this part of the lab, you will configure the CICS Explorer. Likewise, you will find the Remote Systems Explorer simpler to use when editing z/OS UNIX System Services files in ASCII.

### **Part 2: 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 3: 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 4: 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 5: 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 and insert the Java project into the bundle.

### **Part 6: 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 7: 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 8: Summary**

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

## **Part 1: Configure the CICS Explorer Connection to CICS**

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

### **Start the CICS Explorer**

- \_\_\_1. From the **desktop**, **double-click** the **CICS Explorer** icon to start the CICS Explorer 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 CICS Explorer shows you a **Welcome page** click the **Workbench icon** in the upper-right corner to go to the workbench. Then **maximize** the Explorer window.



### **Verify that you have an FTP connection to z/OS in your CICS 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: Create a Project and Create a JAX-RS Servlet**

In this part of the lab exercise we will create a Dynamic Web Project in Eclipse and indicate that the project will contain JAX-RS capabilities. We will then add a pre-written Java program to the project.

In later parts of this exercise we will test the code using the Liberty profile on your desktop. Then we will define CICS resources and run the JAX-RS servlet on Liberty under CICS.

### **Create a Dynamic Web Project**

\_\_\_1. In the **Explorer**, open (or switch to) a **Java EE** perspective.

### **Create a JAX-RS enabled Dynamic Web Project**

\_\_\_2. In the **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.

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

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

\_\_\_4. Back on the **Dynamic Web Project** dialog, click the **Next** button.

\_\_\_5. From the **New Dynamic Web Project** dialog, the **Java** page, click the **Next** button.

\_\_\_6. 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.

\_\_\_7. From the **New Dynamic Web Project** dialog, the **JAX-RS Capabilities** page, **check** the box next to **Update Deployment Descriptor**.

\_\_\_8. Ensure that JAX-RS servlet class name is **com.ibm.websphere.jaxrs.server.IBMRestServlet**.

\_\_\_9. 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.json.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.json.service.SayHelloApplication</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 to edit the web.xml file.

\_\_\_ **14.** **Save** and **close** the web.xml file.

**Note** that you may have some **errors**.

## Add two classes to your project that use JAX-RS

\_\_\_ **15.** From the **Project Explorer** view, **expand** your **com.ddw.sample.json.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.json.service**, then click the **Finish** button.
- \_\_\_18. From the **Project Explorer** view, **right-click** on the **com.ddw.sample.json.service** package and from the context menu select **New > Class**.
- \_\_\_19. From the **New Java Class** wizard, enter a **name** of **SayHello** and click the **Finish** button.
- \_\_\_20. **Replace** the contents of the **SayHello.java** file with the following (note that you can copy the following text from the C:\CICSLAB\CICS-JSON-JAX-RS-HelloWorld\SayHello.java ).

```
package com.ddw.sample.json.service;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;
@Path("/hello")
public class SayHello {
    @GET
    public String helloGet() {
        return "{ \"HelloText\":\"Howdy Partner\" }";
    }
    @POST
    public Response helloPost() {
        return Response.ok("{ \"HelloText\":\"Hi, Neighbor\" }").status(201).build();
    }
}
```

- \_\_\_21. **Create** another Java Class and call it **SayHelloApplication**, with the following contents (note that you can copy the following text from the C:\CICSLAB\CICS-JSON-JAX-RS-HelloWorld\SayHelloApplication.java ).

```
package com.ddw.sample.json.service;
import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.core.Application;
public class SayHelloApplication extends Application {
    //List the JAX-RS classes that contain annotations
    public Set<Class<?>> getClasses() {
        System.out.println("SayHelloApplication: Instantiated");
        Set<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(com.ddw.sample.json.service.SayHello.class);
        return classes;
    }
}
```

---

## 22. Some Explanation of the JAX-RS parts:

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.json.service.SayHelloApplication (this Class must extend javax.ws.rs.core.Application).

The SayHelloApplication, which extends javax.ws.rs.core.Application put the name of the ‘real’ program to get control into a HashSet using an add() method (this program name is com.ddw.sample.json.service.SayHello.class). So, the SayHelloApplication is basically a router program.

The SayHello program will respond to GET and POST requests when it receives a path with ‘/hello’ in it (i.e. /jaxrsnonosgi/data/hello).

## **Part 3: 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 to support JAX-RS.

### **Configure the Liberty Profile**

1. From the **Java EE** perspective, the **Servers view**, expand **WebSphere Application Server Liberty Profile at localhost**, then expand **Server Configuration**, and **double-click** on **Feature Manager**.
2. From the **server.xml editor**, the **Feature Manager** section, ensure that feature list includes **jsp-2.3**, **json-1.0**, and **jaxrs-2.1**. Then **save** and **close** the **server.xml** file when you are done editing it.


### **Start the Liberty Profile**

3. From the **Java EE** perspective, the **Servers view**, **right-click** on **WebSphere Application Server Liberty Profile at Local**, and from the context menu, select **Add and Remove**.
4. From the **Add and Remove** dialog, in the left part of the dialog, select **com.ddw.sample.json.service** and press the **Add** button (in the center). Press the **Finish** button to dismiss the dialog.
5. From the **Servers view**, **right-click** on **WebSphere Application Server Liberty Profile at localhost**, and from the context menu, select **Start**.

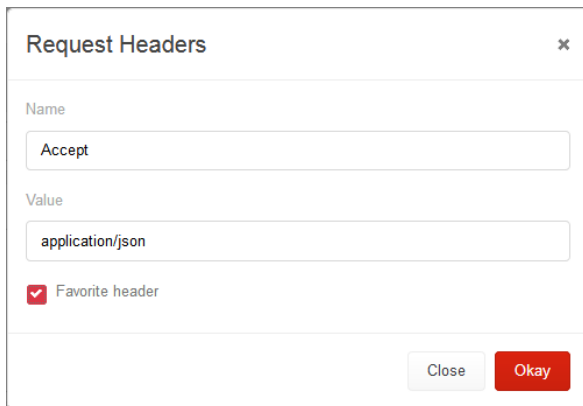


## Try your application

\_\_\_6. Start the **Firefox** browser.

\_\_\_7. From the **Firefox browser**, press the  button in the upper-right corner. If you hover over the button, it says “RESTClient”.

\_\_\_8. From the **RESTClient dialog**, add a header for **Accept – application/json**. Select **Headers > Custom Header**.



\_\_\_9. From the **RestClient dialog**, enter a URL of <http://localhost:9080/com.ddw.sample.json.service/data/hello> and click the **SEND** button. Ensure the Method is **GET** (try the POST and PUT methods also).

\_\_\_10. You should see a “Response” with a status ‘200 OK’. Click the “Preview” or “Response” tab to see the results.

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

## **Part 4: Define a CICS JVM with the Liberty Profile**

In this step you will perform the steps to put your JVMProfile in place and to create a JVMServer (JVM).

Note that you can define one JVMServer and use it for Servlets, REST, and web services.

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

➔ **If you already have a Liberty Profile (JVM) named DDWWLP using a JVM profile named DDWWLP – then skip to step 21 in this section.**

## Copy the Liberty Profile JVMProfile into your USS files

- \_\_\_ **11.** Switch to the **Remote Systems Explorer** perspective, the **Remote Systems** view. **Right-click** on **Local > Local Files > Drives > C:\CICSLAB\Java\ JVMProfiles\DDWWLP** and select **copy** from the popup menu.
- \_\_\_ **12.** From the **CICS Explorer**, the **Remote System Explorer** perspective, the **Remote Systems** view, **right-click** on **/var/cicsts/JVMProfiles** and from the context menu select **Paste**.

**NOTE:** The JVMProfile needs to be DDWWLP.jvmprofile, not DDWWLP. The .jvmprofile extension is a required.

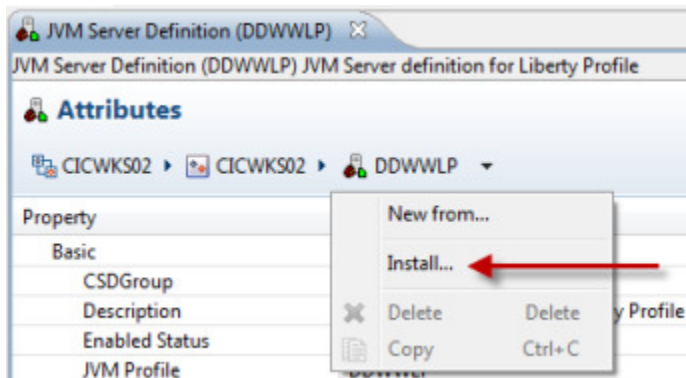
- \_\_\_ **13.** **Right-click** on **DDWWLP** (the file you just pasted in) and select **Rename** from the popup dialog. Enter a **New name** of **DDWWLP.jvmprofile** and click **OK**.
- \_\_\_ **14.** Double-click on **DDWWLP.jvmprofile** to open it in the editor. Ensure that it looks okay (otherwise contact a lab instructor). Close DDWWLP.jvmprofile.

## Define and Install a JVMServer into CICS

- \_\_\_ **15.** From the **CICS Explorer**, the **CICS SM** perspective, from the **menu bar** click on **Definitions > JVM Server Definitions**.
- \_\_\_ **16.** **Right-click** in the **open area** in the JVM Server Definitions panel and select **New**.
- \_\_\_ **17.** In the **New JVM Server Definition** dialog, specify the following, the 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

- \_\_\_18. 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**.



- \_\_\_19. From the **Perform INSTALL Operation** pop-up dialog, click the **OK** button.
- \_\_\_20. Close the **DDWWLP (JVM Server Definition)** editor.
- \_\_\_21. 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`.

## Modify server.xml to include the jaxrs-2.1 feature

- \_\_\_22. From the **Remote System Explorer** perspective, the **Remote Systems** view, **double-click** on `/u/user1/cicslab/logs/CICS1/DDWWLP/wlp/usr/servers/defaultServer/server.xml` file. In the `<featureManager>` tag, add the following and **Save**.

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

**Your running Liberty profile should automatically pick up the change when file is saved**

## **Part 5: Define and Export a CICS Bundle to z/OS**

You deploy a Java REST program to CICS by creating a CICS BUNDLE, and placing your Java 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 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.json.serviceBUNDLE**, then click the **Finish** button.
- \_\_\_3. From the **com.ddw.sample.json.serviceBUNDLE** manifest editor, the **Defined Resources** section, click the **New** button, and from the context menu select **Dynamic WebProject Include**.
- \_\_\_4. In the **Dynamic Web Project Include** dialog, **highlight** the **com.ddw.sample.json.service** project at the top. Then type in a **JVM Server** name of **DDWWLP**, then click the **Finish** button.
- \_\_\_5. Close the **com.ddw.sample.json.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.json.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.json.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.json.serviceBUNDLE\_1.0.0**
- \_\_\_12. Still on the **Export Bundle** page, click the **Finish** button.

## **Part 6: 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 Servlet into your CICS region. The CICS Bundle Definition will point at the CICS bundle on z/OS USS.

- \_\_\_ 1. From the **CICS SM** perspective, from the **menu bar**, select **Definitions > Bundle Definitions**, and from the Bundle Definitions view, **right-click** in and 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	BUREST01
Description	REST Servlet
Bundle Directory	/u/user1/cicslab/bundles/com.ddw.sample.json.serviceBUNDLE_1.0.0
Open editor	Checked

- \_\_\_ 3. From the **CICS Explorer**, the **BUREST01 (Bundle Definition)** editor, under **Attributes** (at the top of the editor), click the **down-arrow** to the right of **BUREST01** (on the top), and select **Install**.
- \_\_\_ 4. From the **Perform INSTALL Operation** pop-up dialog, click **OK**.
- \_\_\_ 5. Close your **BUREST01 (Bundle Definition)** editor.
- \_\_\_ 6. From the **Operations > Bundles** view, verify that the BUREST01 BUNDLE is **ENABLED**.  
**Note** that if your Bundle view was already open, you will have to click the refresh button (🔄).

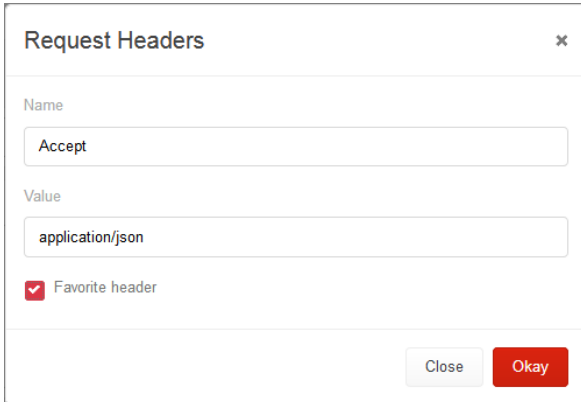
## **Part 7: Test the CICS-based REST Servlet**

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. Try the **Servlet** by typing in the following from a browser:  
<http://wg31.washington.ibm.com:1424/com.ddw.sample.json.service/data/hello>
- \_\_\_ 3. You should see the message come up in your browser. Consult a lab instructor if you don't see the message.

\_\_\_4. From the **Firefox browser**, press the  button in the upper-right corner to start the RestClient.

\_\_\_5. From the **RESTClient dialog**, add a header for **Accept – application/json**. Select **Headers > Custom Header**.



\_\_\_6. From the **RestClient dialog**, enter a URL of <http://wg31.washington.ibm.com:1424/com.ddw.sample.json.service/data/hello>

and click the **SEND** button. Ensure the Method is **GET** (try the POST and PUT methods also).

\_\_\_7. You should see a “Response” with a status ‘200 OK’. Click the “Response” or “Preview” tab to see the results.

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

## **Part 8: Summary**

**Congratulations**, you have created a RESTful Servlet with using JAX-RS to run under the Liberty profile under CICS.

In this lab you performed the following steps:

- Define a Dynamic Web Project in the CICS Explorer
- Code up 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 CICS 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

