IBM – CICS and Java Wildfire Workshop

L70-Creating a Servlet and JSP for CICS TS V5

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
lcompton@us.ibm.com

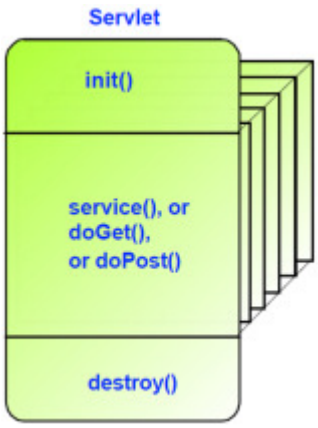
Overview

The purpose of this exercise is to experience Servlet and JSP programming in CICS TS. A subset of the Liberty profile provides a Servlet and JSP engine (JavaServer Page) to the CICS environment. See the CICS TS documentation for a list of the CICS supported parts of the Liberty Profile. We are exposing the Servlet as part of an EAR file. Most Java programmers package Servlets as part of EAR files.

A Servlet is a Java-based server-side web technology. A Servlet extends some prewritten code in the javax.servlet classes. Extending these classes provides standard methods that a Servlet may use to respond to web requests. Servlet methods get invoked during its lifetime to get initialized and destroyed (the init() and destroy()), and the Servlet gets invoked when HTTP requests arrive (doGet(), doPut(), doPost(), doDelete(), etc).

The init() method (optional) gets invoked once when your Servlet is loaded and is available if you want to perform any 'first time thru' processing. Likewise, a destroy() method (optional) gets invoked as your Servlet shuts down.

A doGet() and a doPost() methods (the two most common forms of input from a web browser), are available for GET and POST processing from a web page. Similar to the doGet() and doPost(), a doPut(), doDelete(), doTrace(), etc methods are available if you wish to code them. Since a REST request uses GET, PUT, POST, and DELETE processing, a Servlet can also respond to a REST request.

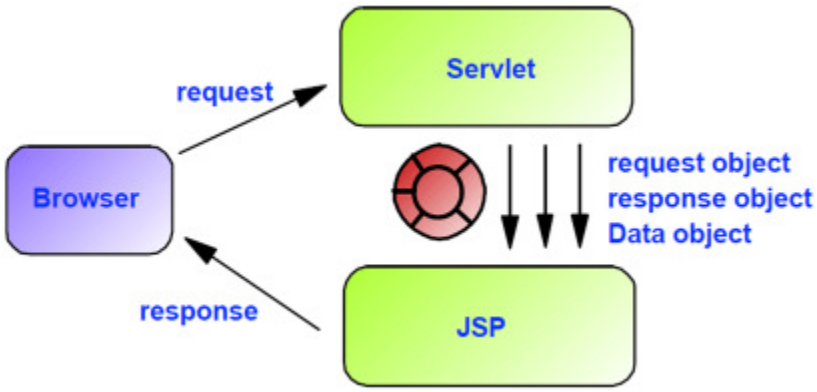


The graphic above is intended to illustrate a Servlet and its lifecycle. The init() method is invoked once, when your Servlet is loaded, at the first request. Your Servlet runs in a multi-threaded environment, and services all of the doGet(), doPost(), doPut(), doDelete(), doTrace(), etc requests received. Multiple web browsers can utilize a single instance of your Servlet at the same time so Servlet code must be threadsafe.

Web browsers are stateless, so Servlets include a way to maintain state between interactions using an HttpSession object. Additionally, there is a Cookie API that allows you to store data in a web browser.

See the ATestServlet.java program that is part of this lab exercise for more details.

When writing large systems, it is easiest to write them using the model-view-controller paradigm. A Servlet receives your request and figures out what to do. It then invokes the business logic to accommodate the request. Then the Servlet passes a Data Bean with the results to a JSP (JavaServer page) to create output to be returned to the web browser.



A JSP's purpose is to render output. A JSP, which is a modified form of Servlet, is loaded and dynamically compiled into a Servlet at first reference. In a JSP, you can code a web page with HTML, and where you would like variable contents you can use '<%' and '%>' symbols to indicate that you want to 'enter Java'. For example, if the above Servlet passed a variable called theDate to a JSP, a simple JSP could be coded as such:

In the Servlet to invoke the JSP (partial program) (OutPutPage.jsp is the target JSP):

```
…
String theDate = "2012-08-28";
request.setAttribute("theDate", theDate);
RequestDispatcher rd = getServletContext().getRequestDispatcher("OutPutPage.jsp");
rd.forward(request, response);
…
```

In the OutPutPage.jsp JSP (complete program):

```
<jsp:useBean id="theDate" class="java.lang.String" scope="request"/>
<html>
<head><title>The Date passed from a Servlet</title></head>
<body>
<b>Date: </b><%=theDate %>
</body>
```

See the OutPutPage.jsp program that is part of this lab exercise for more details.

The Servlet 1.0 API was available in June 1997. Servlet 3.1 was available as of May, 2013.

Lab Requirements

This exercise is performed using the CICS Explorer, with the Java plugins. This exercise will present just one of the ways this can be done. This lab exercise assumes you are using our CICS TS lab environment. The directory names are associated with our environment. Our environment has the CICS Explorer on the desktop.

Lab Step Overview

Part 1: Create a Servlet and a JSP

In this part you will create a simple Servlet and JSP.

Part 2: Run your Servlet in the Liberty Profile on your Desktop

In this part your will test your Servlet and JSP in the Liberty Profile running on your desktop.

Part 3: The Structure of a Servlet

This part will explain the common methods in a Servlet.

Part 4: Create Needed Connections to z/OS

In this part you will create a z/OS connection in the Remote System Explorer perspective of the CICS Explorer. This connection will allow you to copy your Servlet and JSP to z/OS. You will also create a connection from the CICS Explorer to your CICS region. This connection will allow you to define CICS resources that will allow your Servlet and JSP to run in a CICS maintained Servlet environment.

Part 5: Define a CICS JVMServer with the Liberty Profile

This part of the exercise you will create a JVMServer with the Liberty Profile to CICS. Depending on how you organize your Servlets, you may only need to do this step once for several Servlets.

Part 6: Create and Export the CICS Bundle containing your Servlet to z/OS

In this part of the exercise you will create a CICS Bundle and will deploy the Bundle to z/OS.

Part 7: Create and Install the CICS Bundle Definition

In this part, you will create and install a Bundle resource definition for the Servlet and JSP you transferred to z/OS in part 6.

Part 8: Test your Servlet under CICS

This part of the exercise has you invoke your CICS-based Servlet and JSP.

Part 9: Optional: Change the Heading on your CICS-based JSP

This optional part of the lab exercise has you change the heading on your JSP. To see your changes, you will have to go through all of the steps a developer would when they make a change to their program.

Part 1: Create a Servlet and JSP

In the workbench provided, you will create a Dynamic Web Project. This is the type of Eclipse project that is used to create a Servlet and a JSP.

___1. **Start** the **CICS Explorer.**

___2. Open a **Java EE** perspective.

___3. From **Project Explorer** view, **right-click** in an open area and **select New > Dynamic Web Project**.

___4. From the **Dynamic Web Project** dialog, **type** the **following**, then **click** the **Finish** button:

Field	Value
Project name	**ACICSServlet**
Use default location	Checked
Target runtime	Liberty Runtime
Dynamic web module version	**4.0** ←
Configuration	<custom>
Add project to an EAR	**NOT Checked**
EAR project name	

___5. **Right-click ACICSServlet > Java Resources > src** and **select New > Package**.

___6. In the **New Java Package** dialog, under **Name**, **type com.ibm.acicsservlet**. Then click **Finish**.

___7. From a **Windows Explorer** window, using **copy-and-paste**, **copy** the **C:\CICSLAB\CICSServlet01\ATestServlet.java** program **into** your **CICS Explorer** ACICSServlet project's **com.ibm.acicsservlet** package.

Note: Ask a lab instructor if you're not sure how to open a **Windows Explorer** window.

___8. From the CICS Explorer, the **Project Explorer** view, **expand** the **com.ibm.acicsservlet** package, and **double-click** on the **ATestServlet.java** program to view it. Close the ATestServlet.java file when you are done looking at it. Contact a lab instructor if you have any questions.

The Servlet: Note that the line and column numbers are at the bottom-right of the Eclipse perspective by default.

Line 34 is an annotation declaring the name of the Servlet. On line 35 is the declaration of the class and an indication that this Servlet 'extends' HttpServlet. This means that this program extends (or includes) the pre-written code that is in the HttpServlet class.

On line 60 is the init() method, and line 72 is the destroy() method. Line 82 and 95 are the doGet() and doPost() methods (which are invoked when receiving a 'GET' or 'POST' method from the web browser). These methods must have the indicated 'signature', i.e. lines 60, 72, 82, and 95 must be coded as you see them.

NOTE: If you would like line numbers on each line of code, you can…

From the menu bar, click Window > Preferences. Then on the left click General > Editors > Text Editors. On the right check the box next to 'Show line numbers'.

___**9.** From a **Windows Explorer** window, using **copy-and-paste**, **copy** the **C:\CICSLAB\CICSServlet01\OutPutPage.jsp** program **into** your **CICS Explorer** ACICSServlet project's **WebContent** directory.

___**10.** From the CICS Explorer, the **Project Explorer** view, **expand** the **WebContent** folder, and **double-click** the **OutPutPage.jsp** JSP.

A JSP gets interpreted into a Servlet and is normally used to display output (usually a web page). The Servlet we copied earlier performs the business logic (actions on your data) and invokes your JSP to display the data (presentation logic). A JSP makes displaying a web page easier.

The JSP contains references to the response object, and uses <jsp:useBean …> tags to accept the objects passed to it. You can print the value of a passed variable, e.g. with a <%=aCounter %>, or you can use <% and %> to enter Java and put actual Java in your web page. The page is evaluated before it is send to the browser.

The 4th line, response.addHeader, tells the JSP to add a header to the web page so that it will not be cached in the web browser.

The <jsp:useBean …> tags cause the page to accept the objects that that were passed to it. The contents of these objects will be displayed by the page.

After the end of the 'useBean' tags is somewhat normal HTML except that it contains items like "<%=aCounter %>". The indicates the value of the variable aCounter, passed in the useBean tags, should be placed here.
A few lines down, Java code is inserted between a <% and a %> tag.

The JSP is automatically compiled the first time is it sent to a browser.

It is easy to see that if we were to put the 'System.out.println' statements for the web page into our initial Servlet, it would be more difficult to read and to manage than having a JSP that is mostly HTML with a few extra lines inserted.

___**11.** **Close** your **OutPutPage.jsp** file by clicking on the x to the right of the name (at the top of the editor). You should not have changed anything in the file, so if you are prompted to save, click the '**No**' button.

NOTE: See REDP-4880 - Developing Web Applications using JavaServer Pages and Servlets for more details on coding Servlet and JSPs.

Part 2: Run your Servlet in the Liberty Profile on your Desktop

This section is optional. In this section we will run the Servlet in the Liberty profile on your desktop.

___**1.** In the **Servers** view, right-click on **Liberty Server at localhost** and select **Start**.

___**2.** To **run** your **ACICSServlet project** in the **Liberty profile** on your desktop, just **right-click** on the **ATestServlet.java** file and from the **context menu**, select **Run As > Run on Server**.

Note that this is possible because we have installed the Liberty Profile in the CICS Explorer.

___**3.** **If** this is the first time you have run the Liberty profile for this project, you will receive a **Run On Server** panel. **Ensure** that **Liberty Server at localhost** is **highlighted**. Click the **Finish** button.

___**4.** **After a moment,** you will see the **Liberty profile startup messages** in the **Console view**, and the results of the execution of your Servlet will be placed in a **web browser window**. Ensure that your URL in the browser is **localhost:9080/ACICSServlet/ATestServlet**. Contact a lab assistant if your Servlet does not start.

URL should look like this in the browser…
http://localhost:9080/ACICSServlet/ATestServlet

___**5.** This Servlet and JSP don't contain any business logic, but just displays various items about the infrastructure, like the URL, the method, parameters, headers, the name of the Servlet, and the name of the JSP.

___**6.** **Note** the Submit button under the heading on the web page. If you click this button, the form will be submitted to the Servlet, the JSP will be resent, and the counter should be incremented.

___**7.** **Note** the Console view (below the web browser view). The messages you see on the console view are from the Servlet. To write to the console you can use a System.out.println() method.

When running your Java program under CICS in production, you would typically only write error messages to your console since our CICS Java sysout would not be seen by our console operators. Additionally, the sysout file would get very large and may run out of space. In this application, these messages are intended for debugging, and would be removed when moving the application to 'production'. You can change the printMsgLevel variable in the program to eliminate the messages. There 'print' classes are classes that you can get off of the Internet that will allow you to very quickly turn your messages to sysout on and off.

___**8.** **Note** that the Servlet is 'automatically' storing a Cookie in your browser; the name of the Cookie is JSESSIONID which has a long value. The JSESSIONID value is the key to a table of values within the server. The Servlet is storing value of the 'counter' variable (for this session) in the HttpSession object. This Servlet accesses the current value of the counter by requesting a myHttpSession.getAttribute() method. The Servlet adds one to the value and places the new value into the session object under the name 'numberOfTimesThisSession'. The value is passed back to the Servlet the next time it is invoked. Although you could store variable values in the browser cookie, or use hidden fields on the web page, this is the architected way of storing Servlet variables. The HttpSession object, by default, expires in 30 minutes (30 minutes of inactivity) and its values go away.

___**9.** **Optional:** Modify the JSP (and Servlet depending on how advanced you are) and put your name in the page heading and test your change. Note that the environment will automatically restart your Servlet for you when you save your Servlet.

___**10.** Stop your Liberty server on your desktop by clicking on the Servers view, then right-clicking on the Liberty Server at localhost and from the contest menu choose Stop.

Contact a Lab Instructor if you have any problems with this step.

Part 3: The Structure of a Servlet

This section discusses the standard structure of a Servlet and JSP.

When the Servlet is first instantiated, the init() method is invoked. The init() method will only be invoked once. When the Servlet is unloaded the destroy() method is invoked (once). When requests arrive at the Servlet, the request will invoke the doGet(), doPost(), etc method that corresponds to the HTTP method. Servlets are multi-threaded, meaning that multiple requests can be made to the Servlet at the same time. Because a single Servlet can handle multiple requests at the same time, a Servlet must be threadsafe.

For this Servlet and JSP, the user types in the URL of the Servlet into a web browser. The request flows to the Servlet and JSP engine. The doGet() method of the Servlet is invoked and the response data is formulated. The Servlet then invokes the JSP passing it the data to display. The JSP then formulates the data to be displayed on the web browser.

The output page has a Submit button at the top and when pressed, sends a POST request to the Servlet, which follows the same procedure and displays a page (with the counter incremented).

The doGet() and doPost() methods both retrieve information from the Servlet's 'request' object. The doGet() method is invoked when the web browser sends in a 'GET' request and the doPost() method is invoked then the web browser sends in a 'POST' request.

Data passed to the Servlet comes in the query string for a doGet(), and in the body of the HTTP transmission for a doPost() request. The Servlet API is the same to access parameters, no matter how the data arrived (i.e. doGet() or doPost()). On the 'request' object, you can getParameter("theName") to access a variable called 'theName'. The parameters passed to this Servlet are displayed in the section labeled 'Parameters'. The parameters are name/value pairs. I chose the name of the parameters, and the values where specified on 'hidden' parameters.

The page also displays the inbound HTTP headers, plus the Cookie data that was passed from the web browser to the Servlet. The Cookie API is used to access the Cookie data. In this case, the important Cookie data is the JSESSIONID. To save data between browser interactions, you place the data in an HttpSession object. The environment will place the content of the HttpSession object in memory, and assign an id to the data. This id is automatically (done for you) placed in the JSESSIONID which is placed in the Cookie and sent to the web browser. When a subsequent request is sent in from the web browser, the Cookie data associated with this session is also sent in, and if requested, this Cookie JSESSIONID is used as a key to look up the HttpSession associated with this browser request.

We have programmed the Servlet, for test purposes, to print out the request number, and the time it took for the various requests. The output from the Servlet is various variables available to the Servlet and is sent to the JSP to be displayed.

The first time the Servlet (and corresponding JSP) is invoked, the execution time will include a compilation of the JSP. This is interesting because unlike the Servlet which is complied before it is placed in the Servlet engine, the JSP is dynamically compiled on first access.

When you write larger Servlets or systems of Servlets, you will want to divorce the Servlet containing the input logic, from the business logic, and from the output which you place in a JSP (JavaServer page).

Part 4: Create Needed Connections to z/OS

You will need two connections—a CMCI connection to your CICS region and an RSE connection to access files and the spool on z/OS—from your CICS Explorer. If you already have these connections, skip to Part 5.

___1. 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 5: Define a CICS JVM Server with the Liberty Profile

In this step you will perform the steps to put your CICS JVMProfile in place and to create a CICS JVMServer. You can define one JVMServer for multiple Servlets.

Copy the CICS Liberty Profile JVMProfile into your USS files

___1. If you have already defined a Liberty JVMServer (DDWWLP), then you can skip to step 13.

___2. 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 dialog.

___3. From the **CICS Explorer**, the **Remote System Explorer** perspective, the **Remote Systems** view, **right-click** on **MyzOS > z/OS UNIX Files > My Home > cicslab > JVMProfiles**, and from the context menu select **Paste**.

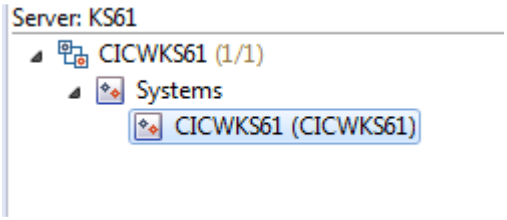
NOTE: The name of JVMProfile contains the file extension of ".jvmprofile". Earlier releases of CICS did not use a file extension. (DDWWLP, not DDWWLP.jvmprofile). The .jvmprofile extension is a required addition in CICS TS V5.2+.

___4. **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**.

___5. 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

___**6.** From the **CICS Explorer**, the **CICS SM** perspective, from the **CICSplex Explorer** view, expand **CICS1** and click on **CICS1**.



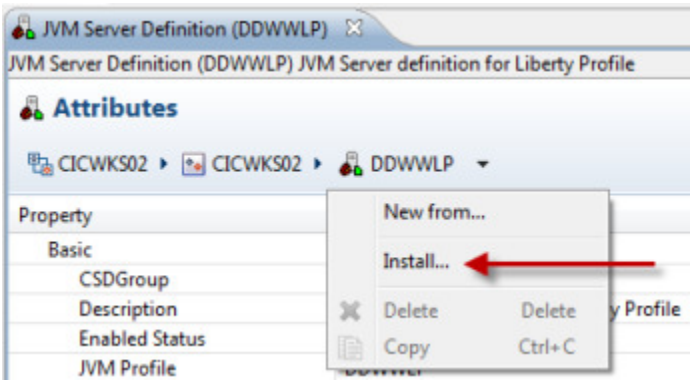
___**7.** From the **CICS SM** perspective, from the **menu bar** click on **Definitions > JVM Server Definitions**.

___**8.** **Right-click** in the **open area** in the JVM Server Definitions panel and select **New**.

___**9.** In the **New JVM Server Definition** dialog, specify the following, the click the **Finish** button.

Field	Value
Resource./CSD Group	WORKSHOP
Name	DDWWLP
Description	JVM Server Definition for Liberty Profile
JVM Profile	DDWWLP
Open editor	Checked

___**10.** From the **DDWWLP (JVM Server Definitions) editor**, under **Attributes, click** the down arrow after your new JVM Server (DDWWLP) and from the context menu select **Install**.



___**11.** From the **Perform INSTALL Operation** pop-up dialog, **highlight** your **CICS region** and click the **OK** button.

___**12.** **Close** the **JVM Server DDWWLP (JVM Server Definitions)** editor.

___13. 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.

Part 6: Define and Export a CICS Bundle to z/OS

When deploying a Servlet to CICS, you define a CICS BUNDLE and place your Servlet 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 Servlet

___1. 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 **BUNDLE-ACICSServlet**, then click the **Finish** button. Note that a **BUNDLE-ACICSServlet mainifest** editor automatically opened.

___3. From the **BUNDLE-ACICSServlet manifest editor**, under **Defined Resources**, click the **New** button, and from the context menu, select **Dynamic Web Project Include**.

___4. In the **Dynamic Web Project Include** dialog, **highlight** your **ACICSServletEAR** project at the top. Then type in a **JVM Server** name of **DDWWLP**, then click the **Finish** button.

___5. **Close** the **BUNDLE-ACICSServlet** manifest editor.

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

___6. In the **Project Explorer** view, **right-click** on your **BUNDLE-ACICSServlet** project and 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 be **BUNDLE-ACICSServlet**

___**10. Still** on the **Export Bundle** page, in the **Parent Directory**, ensure it is /u/user1/cicslab/bundles/

___**11. Still** on the **Export Bundle page**, in **Bundle Directory**, ensure it is /u/user1/cicslab/bundles/BUNDLE-ACICSServlet_1.0.0

___**12. Still** on the **Export Bundle** page, click the **Finish** button

Part 7: Define and Install a CICS Bundle Definition

In this part of the exercise you will define and install a CICS Bundle Definition for your Servlet in your CICS region.

___**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	BUNAPP01
Description	Bundle definition for Servlet
Bundle Directory	/u/user1/cicslab/bundles/BUNDLE-ACICSServlet_1.0.0
Open editor	Checked

___**3.** From the **CICS Explorer**, the **BUNAPP01 (Bundle Definition)** editor, under **Attributes**, click the **down-arrow** to the right of **BUNAPP01** (on the top), and select **Install**.

___**4.** From the **Perform INSTALL Operation** pop-up dialog, click **OK**.

___**5.** **Close** your **BUNAPP01 (Bundle Definition) editor**.

___**6.** From the menu bar, select **Operations > Bundles**, and verify that the BUNAPP01 BUNDLE is **ENABLED**. **Note** that if your Bundle view was already open, you will have to click the refresh button ( ).

Part 8: Test the CICS-based Servlet

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

___1. **Try** the **Servlet** by typing in the following from a browser:
http://wg31.washington.ibm.com:1424/ACICSServlet/ATestServlet

Note that port 1424 was specified in the JVM profile file. This port is specific to your Servlet.

___2. **Question:** So why did you type in the URL in step 1 above ?

Answer: ACICSServlet is the name of your Servlet project. When the project was defined, the 'context root' (the first part the path (after the port)) was defined as the same name as the project. **ATestServlet** is the name of your Servlet. The 'context root' was defined for you, but you can change it.

___3. **Question:** So why did you use port 1424 to access your Servlet?

Answer: This is the port your JVM server was listening on. The port number is in the DDWWLP JVMProfile file. We defined this for you. Take a look at MyzOS > z/OS UNIX Files > My Home > cicslab >JVMProfiles > DDWWLP.jvmprofile under the `-Dcom.ibm.cics.jvmserver.http.port=` parameter.

Part 9: Optional: Change the Heading on your CICS-based Servlet

Optional: In this part you will change the heading on your Servlet and see the changes in a web browser.

___**1.** Change the heading of your Servlet

(Hint: the heading is in the JSP. Go to the Java EE perspective, your ACICSServlet project, and expand WebContent. The edit the OutPutPage.jsp and change the head as desired).

___**2.** To see your changes in your CICS region, re-export your Servlet to CICS by selecting Export Bundle Project to z/OS UNIX File System. Note: you will have to check the box on the Export Bundle page that says 'Clear existing contents of Bundle directory'.

___**3.** Then disable and discard your current Servlet, then install your updated Servlet into your CICS region. Note: You have to discard you Servlet to remove it from the JVM server.

___**4.** Test your Servlet in your browser. Note that the browser may cache your Servlet output, so it may be necessary to hold down the left Control key while you are clicking the Refresh icon.

___**5.** In your DDWWLP.jvmprofile file, there is a WORK_DIR=. We have set have value of that to MyzOS > z/OS UNIX Files > My Home > cicslab > logs so the stdout and stderr from the JVM Server go to that directory.

Part 10: Summary

Congratulations, you have created a basic Servlet with no CICS API and ran it 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 Servlet and JSP (you copied these in)
- You tested the 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 your CICS-provided Servlet
- Optionally: you changed the heading on your Servlet and redeployed it to CICS