IBM – CICS and Java Wildfire Workshop

L72 - CICS-Java-Servlet LINK to COBOL program

Lab Version V61.03.zVA

September 27, 2023

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

Overview

CICS supports Java in the Liberty profile, which runs Servlets, JSPs, Java Server Faces, JAX-RS, JAX-WS and supports local and remote databases. The Liberty profile provides an OSGi-based Java environment, or a non-OSGI-based Java environment. However, if you are going to invoke an OSGi-packaged Java program, the invoking program must be packaged with OSGi packaging

This lab exercise will have you compose a non-OSGI servlet used for presentation logic that invokes our existing business logic written in COBOL. The existing COBOL program uses a COMMAREA (CICS program communications area). We will generate an object from the COMMAREA (that is in COBOL), so that we can easily communicate between the Object orient world of Java and the series-of-bytes orientation of CICS.

The application we will use in this lab exercise is the Catalog Manager Application provided with CICS. It is a simple 3270-based application that allows you to order stationary items. This application ships with CICS and among other things, is intended to illustrate CICS web services. There is a web-front end provided with the application intended to run in WebSphere Application Server (or similar Servlet engine) and invoke CICS-based web services.

In this lab exercise we will use a modified version of the Servlet. We have modified the servlet so that is just LINKs to the business logic in COBOL instead of invoking a web service.

The purpose of this exercise is to illustrate the use of the Liberty Profile in CICS to run a servlet that provides the user interface to an existing COBOL program. We will use the IBM Record Generator for Java ability to create a Java data object from a COBOL data definition. The use of a COBOL program was arbitrary; the business logic could have been written in Java, PL/I, Assembler, C, or any programming language supported by CICS.

Scenario

In our scenario, you are a programmer at some company. You want to write a quick Java-based front-end to some existing business function on your CICS.

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 releases of CICS TS, but in lab environment, we use 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. The CICS-provided Catalog Manager Application has been installed for you.

- **Login:** TSO userids are available with appropriate passwords are provided, however the direction have you using the TSO userid with the z/OS Explorer.

- **CICS Explorer:** In the lab environment we have installed the CICS Explorer which includes JEE version of Eclipse, development tools, plus the Liberty profile was added.

Lab Step Overview

Part 1: Configuring the CICS Explorer connection to CICS

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

Part 2: Prepare the back-end CICS-provided Catalog Manager Application.

The Catalog Manager Application provided with CICS is supplied in various programming languages. We have arbitrarily chosen the COBOL version of this application. All versions of the application (written in various languages) have the same business logic and use the same COMMAREA.

Part 3: Prepare the 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 import the Servlet

In this part of the lab, you will create a Dynamic Web Project and import a Java servlet that LINKS to the CICS-provided business logic. This servlet is a variation of the servlet provided with CICS that is intended to run in WAS and uses web services to the CICS-based programs.

Part 5: Test the application on your desktop Liberty profile

This part of the lab exercise you will verify that you have implemented servlet correctly by executing the 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 a CICS Bundle 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 Servlet

In the part we will test the Servlet from a browser.

Part 10: Summary

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

Part 1: Configure the CICS Explorer Connection to CICS

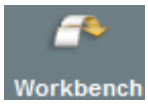
In this part of the lab exercise you will configure the connection between the CICS Explorer running on your workstation to CICS running on z/OS.

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. When you start the CICS Explorer **if** you are prompted for a workspace, 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 Explorer workbench. Then **maximize** the window.



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

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

Part 2: Prepare the Backend Catalog Manager Application

In our situation, the CICS-provided Catalog Manager Application has already been prepared for you using the COBOL version of the application. If you would like to install this CICS-provided application on your company's CICS, you can follow the instructions in…
https://www.ibm.com/docs/en/cics-ts/6.1?topic=samples-cics-catalog-manager-example-application

In the "Subtopics" part of the above web page, you only need to complete the first 3 subtopic areas.

➔ **Again, in our environment, we have installed the CICS-provided Catalog Manager Application for you.**

Part 3: Prepare the IBM Record Generator for Java classes

Our 'backend' business-logic program (the CICS-provided Catalog Manager Application) is an application, written in COBOL, that CICS introduced with CICS TS V3.1 in 2005. This lab will be working with a Java-based servlet that provides the user interface to this program. The version of the Java program that CICS provides was intended to run in WAS (WebSphere Application Server, or a JEE application server) and use web services to communicate to CICS. We have taken this Java 'front-end' (that previously used web services) and modified it so that it issues the CICS-provided JCICS equivalent of an EXEC CICS LINK command to pass a COMMAREA to the existing Catalog Manager business logic in COBOL.

In this part of the lab exercise we will generate a Java class that represents the Catalog Manager COMMAREA. This class will contain getter and setter methods for each field (for the Java side), plus a getByteBuffer() method to access the series of bytes that represents the entire field-oriented COMMAREA we will pass on the LINK command (because the JCICS LINK command passes a byte array). The copybook describing this COMMAREA contains both REDEFINES and OCCURS clauses. In addition to these COBOL keywords, the Record Generator for Java utility supports packed decimal data items and a COBOL OCCURS DEPENDING ON clause (see product documentation for restrictions). Note: we are using a very small part of the Record Generator for Java classes. The Record Generator for Java classes can also be used to read/write to PDSs and various other IBM Z specific functions.

___1. In the IBM Explorer for z/OS, RSE perspective, **right-click** on **USER1.CICSLAB.JCL(UPDJAVA8)** and from the context menu select **Submit**. Click **OK** or **Notify** on the pop-up window.

___2. You may want to look at what you have just submitted for execution. Double-click on **USER1.CICSLAB.JCL(UPDJAVA8)** member to open it in the editor and examine the contents.

Note on the EXEC COBGENIT statement, two symbolic overrides are provided for that JCL procedure: PGMNAME=DFH0XVDS and that PGMLOC=CICSTS61.CICS.SDFHSAMP. This is the CICS-provided backend DFH0XVDS program that contains a COPY statement for the COMMAREA layout, and that our Java program will use when it invokes the DFH0XVDS COBOL program. The COMMAREA was the data structure used to generate the Java class, but you can generate a class from any of the grouping elements.

Note that in the DATAIN DD statement (towards the bottom of the member), that the 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 DFH0XCP1_Datalayout 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 (the ~ indicates your home directory). Note that this was an arbitrary choice for the name of the package and containing directory.

___3. **Close** the UPDJAVA8 **editor**.

___**4.** From the **Remote System Explorer** perspective use the **JES** node to verify that the job ran successfully (must have all return code 00).

___**5.** Note that this Job generated Java source code, when we copy the source code into our Java project (in a later step) it will get compiled 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/DFH0XCP1_Datalayout.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.DFH0XCP1_Datalayout.java. The name of the Java source code is DFH0XCP1_Datalayout.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 most file systems.

This means that if someone told you to look at the com.library.cobol.records.DFH0XCP1_Datalayout.java program in your ~/cicslab/cobgen directory, you would look for
<Home_Directory>/cicslab/cobgen/com/library/cobol/records/DFH0XCP1_Datalayout.java

Note that there is a 'package' statement in DFH0XCP1_Datalayout.java that specifies that it resides in the com.library.cobol.records package. Therefore the Java file -must- be in the specified directory structure.

The usage of the com.library.cobol.records was an arbitrary choice. If you found out that you had generated the .java file into the wrong package, you could modify the control statements on the Job that generated the .java file, or you could modify the package statement at the top of the generated file. But, the package statement in the file must match the directory structure in which the file is stored.

Part 4: Create a Project and import the Java Program

In this part of the lab exercise we will use the CICS Explorer to create an Dynamic Web project and indicate that the project will contain Record Generator for Java capabilities. We will then add a pre-written Java program (very much like the sample JEE Java program that CICS provides to run in WebSphere Application Server) to the project. We will also add the Record Generator for Java class that we just generated.

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 the Windows workstation the IsCICS class will return 'false' and the program will process a dummy COMMAREA instead of LINKing 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 our business-logic COBOL program.

Running the program on the developer's workstation will provide immediate feedback on our Java program (easier, quicker development). Later 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. However, we could do all of our testing/debugging from CICS. From CICS we can debug our program by using CEDX, writing messages to the log, looking at trace, and looking at a dump as well as using the Java source-line debugger.

Create a Dynamic Web Project

___1. In the **IBM Explorer** open a **Java EE** perspective.

___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 **Next** button.

Field	Value
Project name	com.ddw.sample.cics.servlet
Target runtime	Liberty Runtime **
Dynamic web module version	3.1
Configuration	<custom>
Add project to an EAR	NOT Checked
EAR project name	

___4. From the **New Dynamic Web Project** dialog, the **Java** page, click the **Next** button.

___5. From the **New Dynamic Web Project** dialog, the **Web Module** page, **check** the **Generate web.xml deployment descriptor**, and press the **Finish** button.

Note that the 'com.ddw.sample.cics.servlet' is the context root (i.e. the first part of the path when executing the servlet).

___**6.** From the **Project Explorer** view, **double-click** on the **com.ddw.sample.cics.servlet > WebContent>WEB-INF > web.xml** file to open the file in an editor.

Click Welcome File List on the left, then notice the welcome-file-list on the right (under Details). This is a list of default names the environment should look for if the 'initial context' is entered by itself. (if the welcome-file-list is not defined, no problem, it can be added manually, if needed. We will be adding our own index.jsp later)

___**7.** **Close** the web.xml file. You shouldn't have made any changes to the file.

Add the Record Generator for Java-generated files to your project

___**8.** From the **Java EE** perspective, the **Project Explorer** view, **expand** your **com.ddw.sample.cics.servlet** project, then expand **Java Resources > src**.

___**9.** From the **Project Explorer** view, **right-click** on **src**, and from the context menu, select **New > Package.**

___**10.** From the **New Java Package** dialog enter the **Name** as **com.library.cobol.records** then click the **Finish** button.

___**11.** From the **Project Explorer** view, **right-click** on the **com.library.cobol.records** package and from the context menu select **New > Class**.

___**12.** From the **New Java Class** wizard, enter a **name** of **DFH0XCP1_Datalayout** and click the **Finish** button.

___**13.** From the CICS Explorer, 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 **DFH0XCP1_Datalayout.java** to open the file in an editor (this is your Record Generator for Java generated COMMAREA). After the file is open, press **Ctrl-A** to mark all the lines, then press **Ctrl-C** to copy the lines to the clipboard. **Close** the editor.

___**14.** **From the Java EE** perspective**, replace** the contents of the **DFH0XCP1_Datalayout.java** file by pressing **Ctrl-A** and **Ctrl-V** (to paste).

Note that there are <u>several</u> errors. We will 'fix' these errors in a later step.

___**15.** **Save** and **close** the DFH0XCP1_Datalayout.java editor.

The DFH0XCP1_Datalayout class is the COMMAREA representation that you developed in Part 3 of this lab exercise. 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 an equivalent getter or setter for each field.

In Java, you will set the byte array that corresponds to the CICS area (COMMAREA, records, etc) into the structure using the setByteBuffer() method. The Java program can then access the individual fields with getters and setters.

IBM offers two techniques that accommodate this situation: the Java Class Records in the Record Generator for Java, 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 Record Generator for Java Java Class Record. You run a wizard under RAD to develop CICS Java Data Bindings. There is no charge for Record Generator for Java. The generated classes from both products function very similarly.

Add the Java servlet to your project

___**16.** From the **Java EE** perspective, **Project Explorer** view, **right-click** on the **com.ddw.sample.cics.servlet > Java Resources > src** directory, and from the context menu, select **New > Package.**

___**17.** From the **New Java Package** dialog enter the **Name com.ddw.sample.cics.servlet** then click the **Finish** button.

___**18.** From the **Project Explorer** view, **right-click** on the **com.ddw.sample.cics.servlet** package and from the context menu select **New > Class**.

___**19.** From the **New Java Class** wizard, enter a **name** of **CatalogController** and click the **Finish** button.

___**20.** From **the CICS Explorer**, the **Remote System Explorer** perspective, the **Remote Systems** view, **double-click** on **C:\CICSLAB\CICS-Java-Servlet-to-COBOL\CatalogController.java** to open it in the editor. Press **Ctrl-A** to mark all the lines, then press **Ctrl-C** to copy the lines to the clipboard. **Close** the editor.

___**21.** Back in the **CICS Explorer, the Java EE** perspective**, replace** the contents of the **CatalogController.java** file by pressing **Ctrl-A** and **Ctrl-V** (to paste).

Note that this file also has errors, we will fix these in the next step.

___**22. Save** and **close** the CatalogController.java editor.

Fix the error indicators on your project…

___23. From the **Project Explorer view**, **right-click** on your **com.ddw.sample.cics.servlet** project, and from the context menu, select **Build Path > Configure Build Path**.

___24. From the **Properties for com.ddw.sample.cics.servlet** dialog, **on the left** select **Java Build Path**, and **on the right** select the **Libraries tab** (across the top right).

___25. 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**.

___26. **Again**, **back** on the **far-right** of the **Properties for com.ddw.sample.cics.servlet** 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 (not V8.0 with SR6 FP35)** is selected and click **Finish**.

___27. **Back** on the **Properties for com.ddw.sample.cics.servlet** dialog, press the **Apply and Close** button to dismiss the dialog.

NOTE that this should remove <u>all</u> of the errors (and warnings) 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 the Explorer (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 include the packages again.

Note: It is common to put dependent Jar file into the project's WebContent/lib directory. This is a way to specify dependencies specific to the servlet. In this lab we aren't doing this because the CICS JVM server already provides access to the required Java packages for JCICS and Record Generator for Java. However, we do need to indicate the existence of the jzos_recgen.jar.file to the Eclipse project so the Java program can be compiled (the project needs to be 'built'). Specifying these Jar files in the Build Path makes them available for compilation.

We will need to specify these Jar files when we run the servlet in our local Liberty environment (we will do this in the next section when we test the program under our local Liberty.

Include the rest of the application (JSPs, gif, etc)…

___**28.** From a **Windows Explorer window**, navigate to **C:\CICSLAB\CICS-Java-Servlet-to-COBOL\JSPsGIFsCSS** directory. Press **Ctrl-A** to mark all of the files, then press **Ctrl-C** to copy them onto the clipboard.

___**29.** **Back** in the CICS Explorer, the **Java EE** Perspective, the **Project Explorer** view, **right-click** on your **com.ddw.sample.cics.servlet > WebContent**, and from the context menu, select **Paste**.

Contact a lab instructor if you have any error indicators.

The program logic

The CatalogController responds to GET and POST HTTP methods. The first time the program is run from a browser a GET method is driven, the subsequent times a POST method gets driven. The method that is used is controlled by the <form> tag in the web pages. If the web page doesn't have a <form> tag, the web page will drive the GET method.

A servlet gets the request from the web page, instantiate a COMMAREA object, and passes the COMMAREA to a CICS COBOL program. When the COBOL program returns, the Java program invokes a JSP (JavaServer page) where the content from the COMMAREA is merge with HTML and a web page is produced.

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
- 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. 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, debug, code, test, debug, 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.

Note that desktop version of the Liberty profile isn't required. I find it very useful to do the initial coding on my desktop using the Liberty profile, and then move the application to z/OS when I want to test with CICS.

In this section 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, **right-click** on **Liberty Server at localhost** and from the context menu select **Add and Remove**.

___2. From the **Add and Remove** dialog **ensure** your **com.ddw.sample.cics.servlet** project is **on the right side** (under Configured).

Note if your **com.ddw.sample.cics.servlet** project isn't on the right side, on the **left** side, **click** on **the project**, then click the **Add** button in the middle of the dialog.

___3. **Still** on the **Add and Remove** dialog, click the **Finish** button.

___4. From the **Web** perspective, the **Servers view**, expand **Liberty Server at localhost**, then expand **Server Configuration**, and **double-click** on **Feature Manager**.

___5. From the **server.xml editor**, the **Design** tab (across the bottom), the **Feature Manager** section, ensure that feature list includes
jsp-2.3

___**6.** **Still** in the **server.xml** editor, click the **Source** tab (across the bottom), and **add the following** after the <httpEndpoint… > tag (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>
```

As stated earlier, we are pointing to these support files using this technique because we don't want to make the support files part of the project when we deploy the project to CICS.

___**7.** **Still** in the **server.xml** editor, **add the following** inside the webApplication tag.
NOTE: Just add the "classloader" and the "</webApplication>" parms to the existing parms and remove the "/" from the first line. (to have your application recognize the CLASSPATH):

Note that on the first line, the statement currently ends with … servlet"/> You need to take out the slash, and add the next two lines.

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

___**8.** **Save** and **close** the **server.xml** file.

Start the Liberty Profile

___**9.** **Back** on the **Java EE** Perspective, the **Servers** view, right-click on **Liberty Server at localhost** and select **Start**.

___**10.** In the **Project Explorer** view, **right-click** on **index.jsp** (in the WebContent directory) and from the **context menu**, select **Run as > Run On Server**.

___**11.** From the **Run On Server dialog**, press the **Finish** button (may take a few seconds for the server to start).

___**12.** If the application does not display, consult a lab instructor.

Try your application

___**13.** The action you took in step 10 above should have started the **CICS Example – Catalog Application**, but if not, you should be able to specify **http://localhost:9080/com.ddw.sample.cics.servlet/** in **Firefox** or a different browser.

___**14.** Press the **LIST ITEMS** button and you should go to a page where you can specify the starting Catalog Reference number for the browse. When you press the **SUBMIT** button you should see two items (remember, since we are not running this on CICS, the application provides dummy data).

___**15.** You can **press** one of the **radio buttons** and press the **SUBMIT** button to see an order screen. You can fill in the order screen and you should see "The order took place".

___**16.** You can try more options on this very small application, but remember, the application isn't running in CICS, so it is returning dummy data.

___**17. Next**, you will get the servlet running on CICS, and you will get 'real' data.

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

In this step you will perform the steps to start a Liberty profile (JVM) 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 JVM named DDWWLP using a JVM profile named DDWWLP – then skip to step 13 in this section.**

Copy the Liberty Profile JVMProfile into your USS files

___1. From the **CICS Explorer**, the **Remote System Explorer** perspective, click **Window > Preferences** in the tool bar at the top. **Navigate** to **Remote Systems > Files**. On the right side of the panel, click **Determine from target environment.**

___2. Click **Apply and Close** to close the dialog.

___3. From **the CICS Explorer**, the **Remote System Explorer** perspective, the **Remote Systems** view, **right-click** on **C:\CICSLAB\Java\JVMProfiles\DDWWLP** and select **copy** from the popup menu.

___4. From the **CICS Explorer**, the **Remote System Explorer** perspective, the **Remote Systems** view, **right-click** on **z/OS UNIX Files > My Home > cicslab > JVMProfiles** and from the context menu select **Paste**.

NOTE: the JVMProfile needs to be DDWWLP.jvmprofile. The .jvmprofile extension is a required.

___5. **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**.

___6. 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

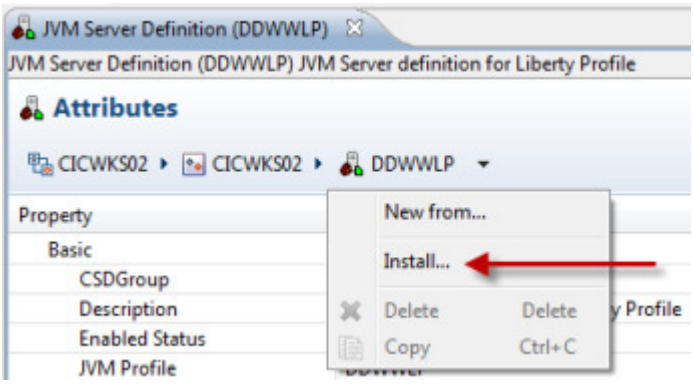
___**7.** From the **CICS Explorer**, the **CICS SM** perspective, from the **menu bar** click on **Definitions > JVM Server Definitions**.

___**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 Group	WORKSHOP
Name	DDWWLP
Description	JVM Server Definition for Liberty Profile
Enabled Status	ENABLED
JVM Profile	DDWWLP
Open editor	Checked

___**10.** 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**.



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

___**12.** **Close** the **JVM Server Definition(DDWWLP)** 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.

___**14.** In **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 (if it isn't already there):

```
<feature>jsp-2.3</feature>
```

___**15. Save** and **close** the server.xml file.

___**16. Note** that when you save the server.xml, the Liberty profile will automatically pickup the changes.

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

When deploying a Java program to CICS, you define a CICS BUNDLE and place the EAR file or WAR file containing your Java 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. CICS TS V5.3 and above supports either WAR or EAR files.

Create a CICS BUNDLE containing your Java EAR file on your workstation

___1. From the **CICS 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**. Then click the **Next** button.

___2. From the **CICS Bundle Project** dialog, type in a name of **com.ddw.sample.cics.servlet-BUNDLE**, then click the **Finish** button.

___3. Right-click your **com.ddw.sample.cics.servlet-BUNDLE**, select **New > Dynamic Web Project Include**.

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

___5. **Close** the **com.ddw.sample.cics.servlet-BUNDLE** 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.cics.servlet-BUNDLE** 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.cics.servlet-BUNDLE**

___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.cics.servlet-BUNDLE_1.0.0**

___**12.** Check the **Clear existing contents of Bundle directory** box.

___**13. 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 Java program into your CICS region.

___1. From the **CICS Explorer**, 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	BUSERV00
Description	Java Servlet front-end for the Catalog App
Bundle Directory	/u/user1/cicslab/bundles/com.ddw.sample.cics.servlet-BUNDLE_1.0.0 Note: use the Browse button on the right.
Open editor	Checked

___3. From the **CICS Explorer**, the **BUSERV00 (Bundle Definition)** editor, under **Attributes**, click the **down-arrow** to the right of **BUSERV00** (on the top), and select **Install**.

___4. From the **Perform INSTALL Operation** pop-up dialog, click **OK**.

___5. **Close** your **BUSERV00 (Bundle Definition) editor**.

___6. From the menu bar, select **Operations > Bundles**, then verify that the BUSERV00 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 Catalog Servlet

In this part you will use a web browser to access your CICS-based Catalog servlet.

___1. **Start** the **Firefox** browser.

From **Firefox**, enter a URL of
http://wg31.washington.ibm.com:1424/com.ddw.sample.cics.servlet

and press **Enter**.

___2. Click the **LIST ITEMS** button, and on the next page press the **SUBMIT** button. You should see a listing of the current inventory.

___3. Try the various paths through the application, and order some items.

If you have any problems with the application, consult a lab instructor.

Part 10: Summary

Congratulations, you have modified a Java CICS-provided front-end that invokes a backend COBOL program.

In this lab you performed the following steps:

- Defined a Dynamic Web Project using the CICS Explorer
- Used the Record Generator for Java utilities to create a Java equivalent to the COMMAREA
- Coded a servlet, which used the IBM Record Generator for Java generated COMMAREA to invoke a CICS COBOL program
- You tested the servlet in the Liberty profile on your desktop
- You defined a CICS bundle on your desktop
- You defined a JVM server
- You defined a CICS BUNDLE resource and included your Dynamic Web Project in the BUNDLE
- You tested the servlet under CICS