

WebSphere Application Server for z/OS Version 8.5

## Liberty Profile

# Quick Start Guide

*Version Date:* July 25, 2016

See "Document Change History" on page 39 for a description  
of the changes in this version of the document

IBM Advanced Technical Skills  
**Gaithersburg, MD**

**WP102110** at  
[ibm.com/support/techdocs](http://ibm.com/support/techdocs)  
© IBM Corporation 2012

Many thanks to **David Follis, Matt Sykes and Jeff Mierzejewski** of the WAS z/OS development staff.

The WAS z/OS team in ATS consists of John Hutchinson, Mike Kearney, Mike Loos, Louis Wilen, Lee-Win Tai and Don Bagwell. Glenn Fisher manages the team.

Mike Cox, Distinguished Engineer, serves as technical consultant to all our activities.

---

## Table of Contents

<b>Overview.....</b>	<b>4</b>
Objective of this document.....	4
WAS Version 8.5 Infomation Center.....	4
Other WP102110 documents at ibm.com/support/techdocs.....	4
<b>High-Level Overview of Key Concepts and Terminology.....</b>	<b>5</b>
Installation of WebSphere Application Server z/OS V8.5, including Liberty Profile.....	5
The Liberty Profile "install directory".....	5
The Liberty Profile "user directory".....	6
User directory within the install directory.....	6
User directory in a separate location from the install directory.....	6
User directory within a packaged and deployed Liberty Profile location.....	7
Starting a configured Liberty Profile server from the UNIX command line.....	8
Starting the server as z/OS started task?.....	8
Section summary.....	9
<b>Liberty Profile and the UNIX Shell -- Step-by-Step Instructions.....</b>	<b>10</b>
Verifying where the Liberty Profile install directory is on your system.....	10
Create Liberty Profile user directory.....	10
Check for _BPXK_AUTOCVT=ON.....	10
Create and start your first Liberty Profile server.....	10
Create multiple servers and take advantage of shared applications.....	13
Introducing the server.env file and the jvm.options file.....	14
Use of <include> in server.xml.....	16
Enabling SSL using z/OS SAF as the keystore.....	18
Establishing shared LTPA keys for single sign-on between servers.....	20
Section summary.....	22
<b>Liberty Profile and z/OS -- Step-by-Step Instructions.....</b>	<b>23</b>
Before you get started.....	23
Copy out the sample JCL procs and customize.....	23
z/OS WLM Classification.....	24
Some closing notes about WLM classification.....	26
Overview of the "Angel" process.....	26
Setting up the Angel process.....	27
Setting up server ID access to the Angel process.....	27
Starting the Angel process and restarting the servers to access the Angel.....	27
Preparation for JDBC Type 2 to DB2 exercises.....	28
JDBC Type 2 to DB2 with RRS transaction support.....	29
SAF authentication and authorization.....	31
z/OS MODIFY.....	33
<b>Catalog of common error symptoms and their causes.....</b>	<b>34</b>
Errors starting server.....	34
Server name specified in PARMS= not found.....	34
Location of Java SDK not resolved.....	34
Location of Java SDK resolves to a 31-bit Java SDK.....	34
Required server.xml file not present in server directory.....	35
Errors related to server.xml configuration.....	35
File server.xml stored on z/OS in EBCDIC rather than ASCII.....	35
Incorrect XML syntax in server.xml.....	35
<b>z/OS Definitions and Other Information.....</b>	<b>36</b>
Userid and group.....	36
STARTED profiles.....	36
SERVER profiles (for access to Angel process).....	36
EJBROLE profiles (for security authentication and authorization).....	36
Installation of Liberty with WAS z/OS 8.5.5.....	36
<b>Information Center Links.....</b>	<b>38</b>
<b>Document Change History.....</b>	<b>39</b>

## Overview

The Liberty Profile is a new application server runtime model included with WebSphere Application Server V8.5. The Liberty Profile is available on all platforms supported by WebSphere Application Server itself.

The key points about the Liberty Profile to be aware of are:

- Unlike traditional WAS z/OS, the Liberty Profile server instance is a single JVM rather than the multiple-JVM model used by traditional WAS z/OS.
- The Liberty Profile is part of the packaging and delivery of WAS z/OS V8.5, but Liberty Profile server instances do *not* operate under the management of a Deployment Manager, nor are they part of the "cell." Liberty operates separately from traditional WAS z/OS<sup>1</sup>.
- The creation of Liberty Profile server instances is done with a supplied shell script. There is no need to use the customizaton tools (WCT) or run generated JCL batch jobs.
- Configuration of Liberty Profile server instances is done by editing the `server.xml` file and a small set of other optional files. There is no "Admin Console" for Liberty Profile.
- The Liberty Profile is designed to be composable, lightweight, dynamic and fast:
  - Composable** -- the function is very modular and flexibly decoupled, allowing you to specify just what function you need for the applications you are serving.
  - Lightweight** -- the Liberty Profile uses a number of approaches to optimize the loading of functions, which results in a footprint significantly less than traditional WebSphere Application Server.
  - Dynamic** -- many of the runtime feature sets may be dynamically updated by simply changing the configuration XML; applications may be added or updated by simply replacing the application file in the file system directory. Server restarts are *not* required.
  - Fast** -- due to the composable design and other factors, the Liberty Profile is able to start in a matter of seconds and in many cases perform better than traditional WAS.

### Objective of this document

The objective of this document is to help you achieve preliminary success with the Liberty Profile on z/OS as quickly as possible. It will do this by providing a series of step-by-step exercises that start with the simplest elements of the Liberty Profile, then build on that to introduce more complex topics.

### WAS Version 8.5 Information Center

This document does *not* try to be a single source of reference for the Liberty Profile, on z/OS or any platform. The Information Center serves that role.

This document will provide unique search strings to guide you to specific articles related to the topics being discussed.

### Other WP102110 documents at [ibm.com/support/techdocs](http://ibm.com/support/techdocs)

The WP102110 document at [ibm.com/support/techdocs](http://ibm.com/support/techdocs) is a collection point for documents related to WebSphere Application Server for z/OS Version 8.5 and the Liberty Profile.

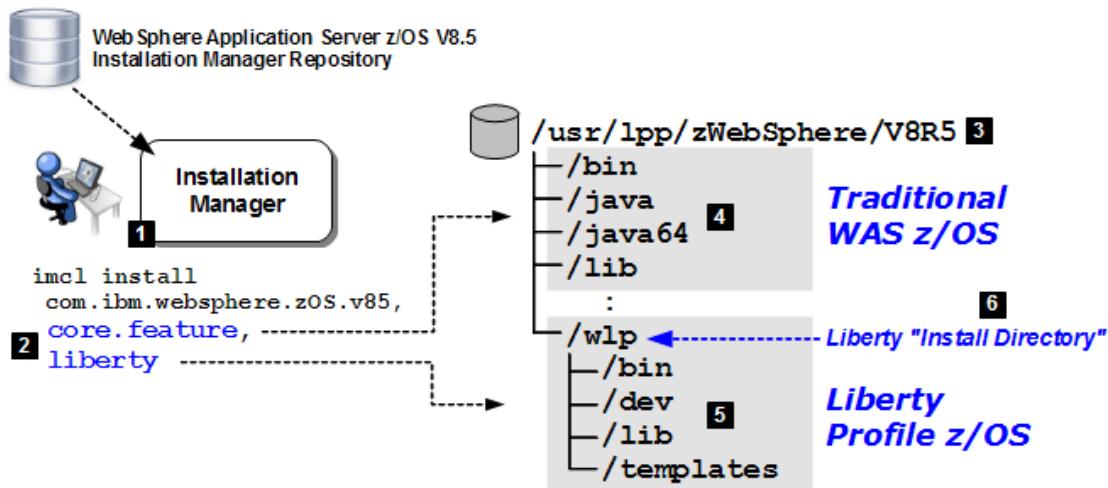
If you are viewing this document as a PDF, the Techdoc may be access with [this](#) link.  
Otherwise, go to [ibm.com/support/techdocs](http://ibm.com/support/techdocs) and search on WP102110.

<sup>1</sup> There is the capability of managing Liberty Profile server instances using the traditional WAS "Job Manager." That is something outside the scope of this document.

## High-Level Overview of Key Concepts and Terminology

In this section we'll simply convey key concepts and terminology. There are no hands-on exercises in this section. The step-by-step exercises come later.

### ***Installation of WebSphere Application Server z/OS V8.5, including Liberty Profile***



#### Notes:

1. The WAS z/OS V8.5 product code<sup>2</sup> is installed using IBM Installation Manager, as was true for WAS z/OS V8.0 as well. How to use the Installation Manager is beyond the scope of this document<sup>3</sup>. For this document we assume it has been installed.
  2. With WAS z/OS V8.5 you have the option in IM to install traditional WAS z/OS (`core.feature`) or Liberty (`liberty`) or both. The picture is illustrating installing *both*.
- Note:** The Liberty Profile components are relatively small -- about 60MB in total. If you are doing the WebSphere Application Server for z/OS V8.5 install we recommend you do both.
3. This is the installation directory where IM installs the product. This may be any location you choose, such as what is illustrated in the picture. Typically there will be one instance of this and it will be mounted Read-Only after IM has performed the installation.
  4. Within the installation directory will be a set of sub-directories that hold the *traditional* WAS z/OS product files and executables. The picture is somewhat simplified from what actually appears.
  5. When the Liberty Profile is installed it appears as a directory tree under `/wlp` as shown in the picture. The picture is somewhat simplified from what actually appears.
  6. The Liberty Profile "install directory" will be an important part of the discussion. In this picture the Liberty Profile install directory is:  
`/usr/lpp/zWebSphere/V8R5/wlp`

#### ***The Liberty Profile "install directory"***

The Liberty Profile "install directory" where the Liberty Profile product files and binaries may be found. In the picture above, that install directory is:

`/usr/lpp/zWebSphere/V8R5/wlp`

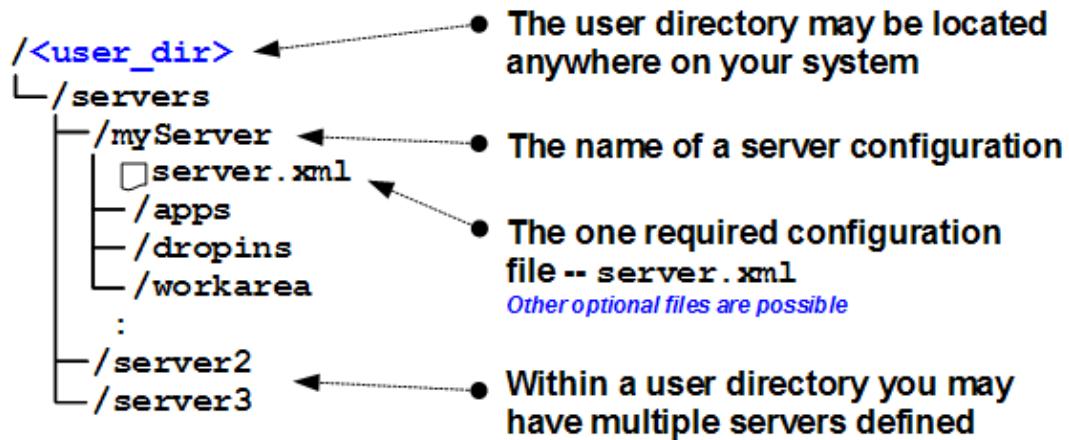
Your install directory may be different from that, however. So throughout this document we'll refer to this install direct as `<install_dir>`. Substitute your value in at that point.

2 With WAS z/OS 8.5.5 the installation of Liberty is a bit different. See "Installation of Liberty with WAS z/OS 8.5.5" on page 36.

3 See WP102014 at [ibm.com/support/techdocs](http://ibm.com/support/techdocs) for a document on installing and using IM on z/OS for installing and managing WebSphere Application Server for z/OS.

## The Liberty Profile "user directory"

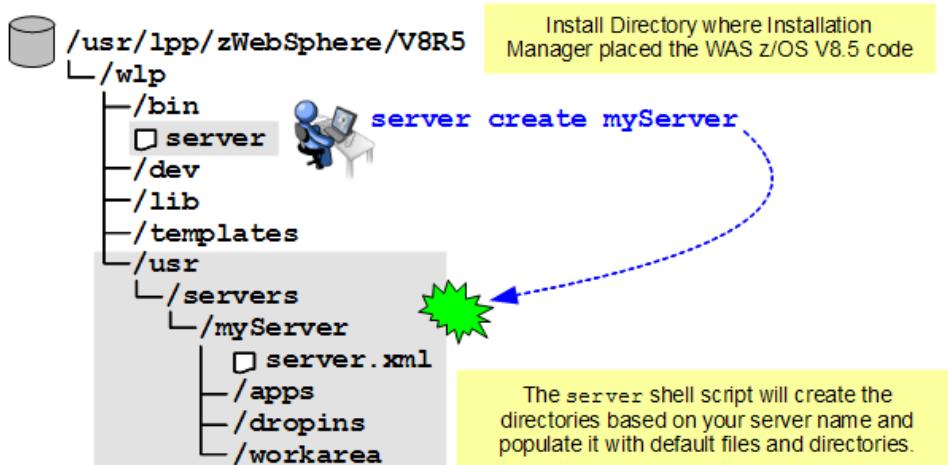
The "user directory" is where the configuration files for servers are maintained. At a very high level it looks something like this:



Throughout this document we will use the tag `<user_dir>` to refer to whatever user directory you are using. A question you may have is, "How is this user directory created?" We'll get to that next.

### User directory *within* the install directory

A server configuration is created using a supplied shell script called `server`. By default that will try to create the server configuration *within* the `<install_dir>` at the following location:



That works *if* your install directory is mounted Read-Write *and* you have write permission to that directory. If it's Read-Only or you don't have permission it won't work as shown.

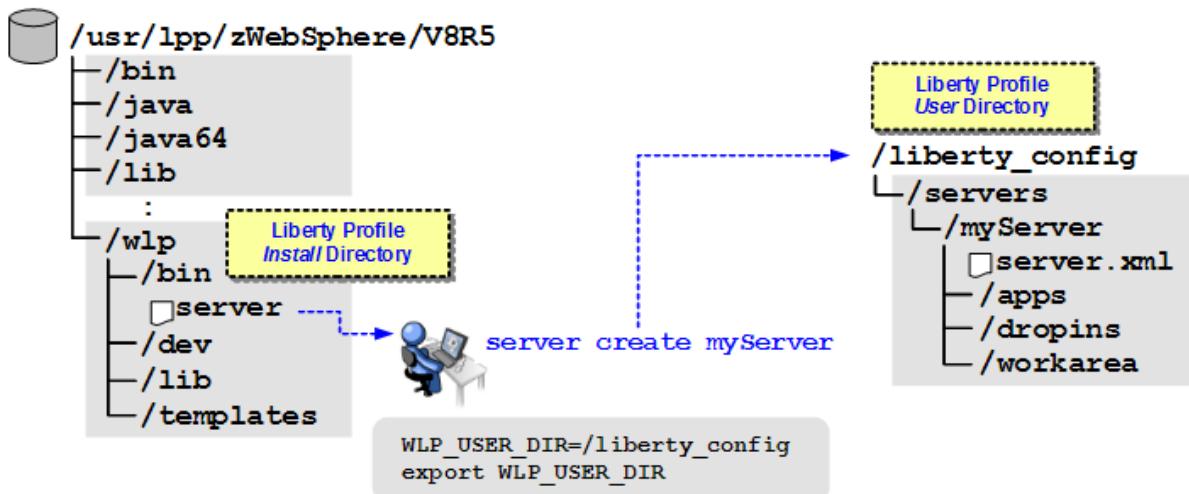
**Note:** For this document we will assume the install directory created by Installation Manager will be mounted Read-Only at your location, which is what is commonly done. So we will assume you would *not* be able to create a user directory within that install directory.

### User directory in a separate location from the install directory

An alternative is to create the user directory at some location outside the install directory. We have already mentioned that user directory may exist anywhere on your system.

For example, assume you create a directory called `/liberty_config` and you make it your<sup>4</sup> `<user_dir>` ... the picture then looks like this:

<sup>4</sup> You are not limited to a single `<user_dir>`. You may have multiple located at different locations. Here we illustrate one.

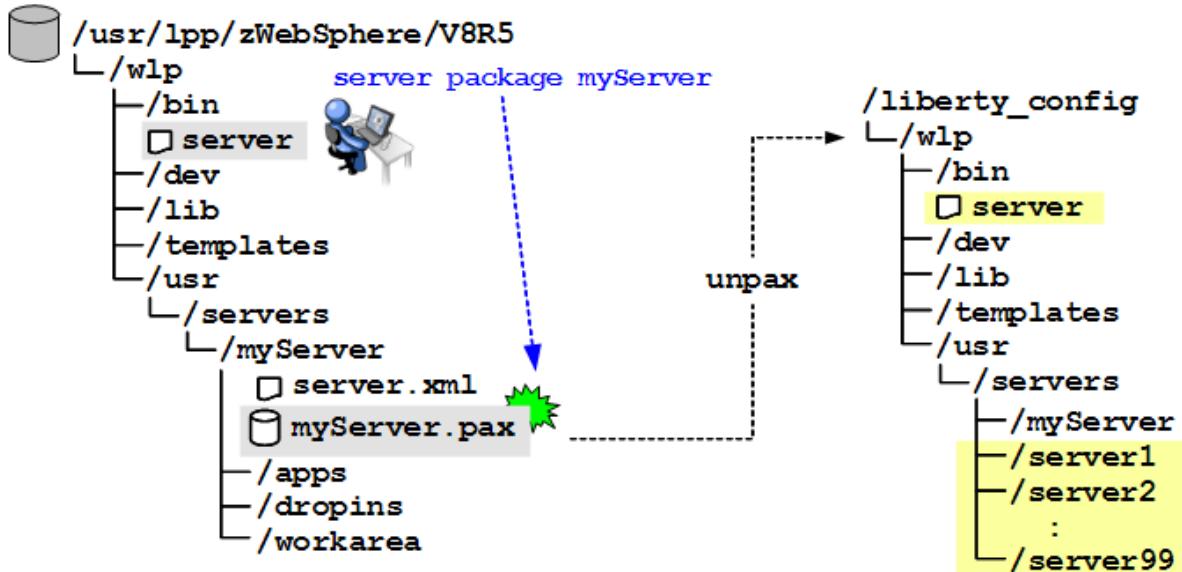


The key is to make the UNIX shell environment know where to create the user directory. That is accomplished by setting the environment variable `WLP_USER_DIR`. In this example we show it being set to `/liberty_config`. The `server` shell script creates the server directories and files in the specified directory.

### User directory within a packaged and deployed Liberty Profile location

The server shell script is capable of doing more than just creating server configuration directories. It is also capable of packaging up a server to be deployed somewhere else. You may package just the user definitions for a server or the user definitions as well as the Liberty execution binaries.

Assume the install directory has a server defined within it and is mounted Read-Write<sup>5</sup>.



The package action on the `server` shell script will create a pax archive that you may then unpax<sup>6</sup> at any other location<sup>7</sup>. Once unpaxed you will find the `/wlp` directory, the `/bin`

<sup>5</sup> We are showing one way of using this `package` action on the `server` shell script. Variations on this illustration are possible: you can indicate the pax archive is to be written to a location outside the `<install_dir>`. It is also possible to package a server configuration that is constructed outside the `<install_dir>`. For the purpose of this document simply be aware of the flexibility offered.

<sup>6</sup> `pax -r -ppx` to preserve some important z/OS extended attributes on a few files.

<sup>7</sup> That pax will not itself contain the JVM executable binaries. The Liberty Profile relies on a 64-bit Java 6 or Java 7 installation existing elsewhere and referenced with the `JAVA_HOME` environment value.

directory, the `/lib` directory -- just like what was in the original install directory. Then you would be able to use the `server` shell script and create additional servers as you pleased.

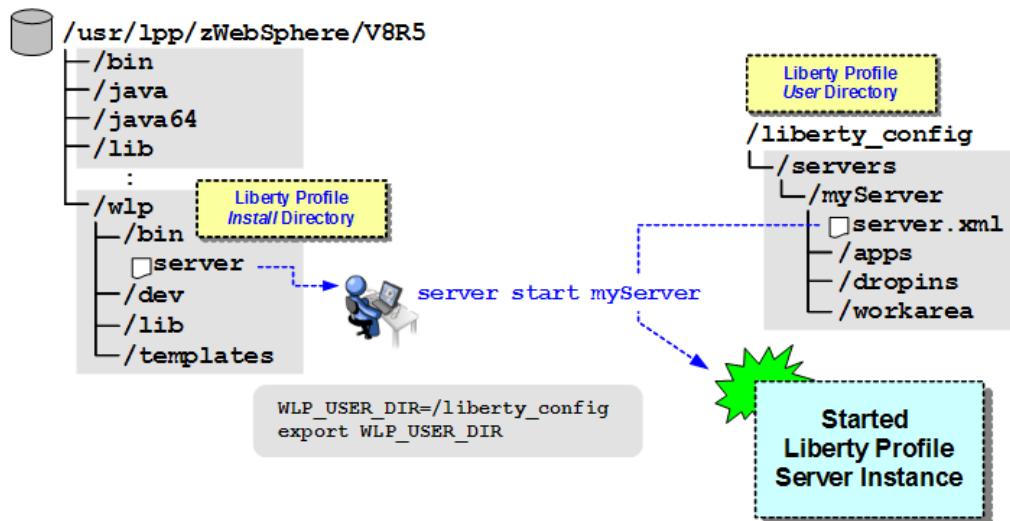
This mechanism allows you to create *ad hoc* install directories at whatever location you choose. It then allows you to create server definitions within that *ad hoc* install directory since you would likely have write access to it.

There is one thing to note about this approach -- maintenance updates managed by Installation Manager against the original installation directory would not apply to these *ad hoc* install directories. The responsibility to maintain your *ad hoc* Liberty Profile install directories would be yours, not Installation Manager's.

**Note:** We will *not* illustrate this process in this document. We will assume use of the install directory created by Installation Manager and a user directory separate from that install directory. We illustrated this under "User directory in a separate location from the install directory" on page 6.

### Starting a configured Liberty Profile server from the UNIX command line

Let's revisit our earlier picture of the user directory at a separate location from the install directory and show how easy it can be to start the configured server instance:



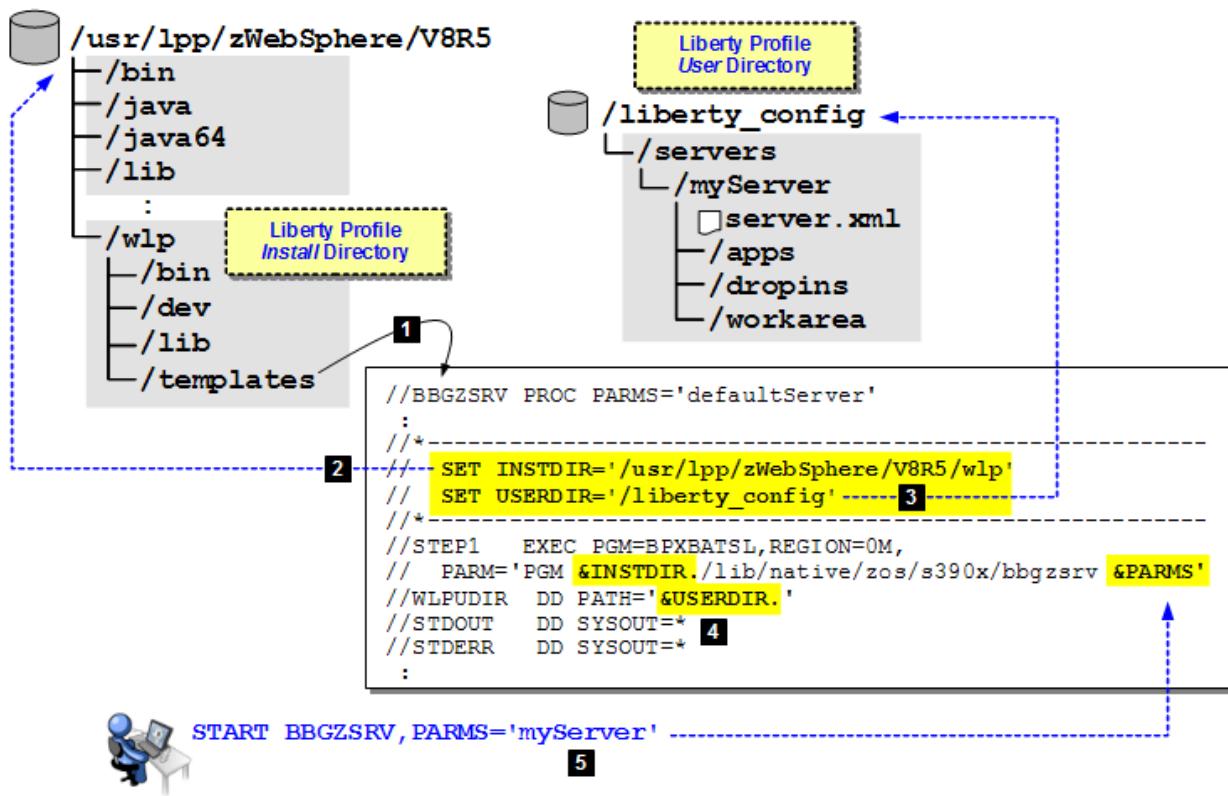
The `server` shell script comes into play once more -- this time with the start action specified.

In this picture the server shell script resides in the install directory and the UNIX shell environment is aware of the location of the user's server configuration directory based on the `WLP_USER_DIR` environment variable.

### Starting the server as z/OS started task?

You may wonder if a Liberty Profile server instance may be started as a z/OS started task, and the answer to that is yes. Within the Liberty Profile install directory there is supplied a set of sample JCL start procedures that may be customized and used to start Liberty as an MVS started task.

The following picture illustrates at a somewhat high-level how the JCL is structured to support install directories separate from user directories, and the passing in of the server name as a parameter on the START command:



### Notes:

- The sample JCL start procedures are found under the following directory:  
`/<install_root>/wlp/templates/zos/procs/bbgzsrv.jcl`  
**Note:** The illustration here is simplified with comments removed to save space.
- Two `SET` statements are included, one to set the path for the install directory (`INSTDIR`) and one for the user directory (`USERDIR`). Here we show `INSTDIR` pointing to the example installation directory.
- The second `SET` statement points to the user directory. Here we show `USERDIR` pointing to `/liberty_config`. You do *not* need to specify `/servers`, that is assumed.
- The two values are used lower in the JCL as substitutions variables.
- The `START` command includes a `PARMS=` value to name the specific server you wish to start within `USERDIR`. Note the single quotes. Server names relate to UNIX directories, which are case sensitive. The single quotes preserve the case. The `PARM=` value you supply is substituted in with the `&PARMS` variable as shown.

### Section summary

In this section we covered a few essential points about Liberty Profile, its configuration and its operations. There's a great many details we did not cover. Those details are what provide you greater and greater flexibility and control for maximizing the value of Liberty Profile to you and your organization.

But this was a "high level overview" section, and this document is intended to be a "quick start" guide, not an encyclopedia of all possible features and functions. The InfoCenter serves that role.

What we will do next is provide step-by-step instructions for creating and starting a Liberty Profile server ... first from the UNIX shell, and then as a z/OS started task.

## Liberty Profile and the UNIX Shell -- Step-by-Step Instructions

We will start with Liberty Profile from the z/OS UNIX shell because it is the best way to illustrate just how simple it is to get a Liberty Profile server working on z/OS. Starting on page 23 we will show how to operate Liberty Profile servers as z/OS started tasks.

### **Verifying where the Liberty Profile install directory is on your system**

- Check with your z/OS system administrator to find out where the Liberty Profile has been installed on your system. This will be the full path up to *and including* /wlp. Record the full path value here:

Liberty Profile <code>&lt;install_dir&gt;</code>	
---	--

### **Create Liberty Profile user directory**

- Open a Telnet session (or OMVS) to your z/OS system using your normal login ID. We wish to illustrate it does *not* require a special ID to use the Liberty Profile.
- Create a directory at whatever location you desire. Give that directory permissions that allow your ID to read and write (ex: 600). Record the full path value for that directory here:

Liberty Profile <code>&lt;user_dir&gt;</code>	
--	--

### **Check for `_BPXK_AUTOCVT=ON`**

**Note:** The Liberty Profile text based logs and configuration files created and managed by Liberty are tagged as text files with an appropriate code page. That means that in most cases if auto conversion is enabled for USS (`_BPXK_AUTOCVT=ON`), editors and script utilities should work. There would then be no reason to use viascii or to convert a file to EBCDIC with tools like a2e and convert them back. In fact, you should *not* do that -- if you do, you will break the tags, which makes the files harder to work with overall.

- From your Telnet session, issue the command: `echo $_BPXK_AUTOCVT`
- Then:  
If the response is ON then auto conversion is enabled for your shell environment. No further action required.  
If the response is OFF or the response is null, then auto conversion is *not* enabled. Enable it now with the following command:  
`export _BPXK_AUTOCVT=ON`

**Note:** If you need to tag a file, use this command: `ctag -t -c iso8859-1 <filename>`

To make that easier consider setting up an alias. The following is an example:

```
alias tagascii="ctag -t -c iso8859-1"
```

Then the command `tagascii <filename>` tags the file. If you are uncertain what files in a directory are tagged, the command `ls -lt` will display those tagged and those not.

T=on indicates those tagged, T=off those not.

### **Create and start your first Liberty Profile server**

- From your Telnet session, change directories to your Liberty Profile `/<install_dir>/bin` directory. That `/bin` directory should be immediately under `wlp`.
- Enter the following command<sup>8</sup>:  
`echo $JAVA_HOME`  
Do you get a value returned, and does that value point to a valid location for **64-bit Java 6 or Java 7?** If yes, then proceed to the next check-box exercise step.

<sup>8</sup> Throughout this document we assume a Bourne compatible shell.

If not, then do the following<sup>9</sup>:

- \_\_\_\_\_ Determine where a 64-bit Java 6 or Java 7 installation resides on your system<sup>10</sup>
- \_\_\_\_\_ Enter: `JAVA_HOME=<path_to_64bit_Java>`
- \_\_\_\_\_ Enter: `export JAVA_HOME`
- \_\_\_\_\_ Enter: `echo $JAVA_HOME` and verify the path that is returned

Enter the following commands:

`WLP_USER_DIR=/<user_dir>`

where `<user_dir>` is the user directory you created just a few moments ago.

Then:

`export WLP_USER_DIR`

Then:

`echo $WLP_USER_DIR`

and verify the location echoed back is indeed the full path to the Liberty user directory you created.

From the `/<install_dir>/bin` directory<sup>11</sup>, enter the following command:

`server create server1`

You should see the following response:

`Server server1 created.`

Now start that server. Enter the following command:

`server start server1`

You should receive a response *something* like this:

`Server server1 started with process ID 67371533.`

Congratulations! You have created and started your first Liberty Profile server instance. By default that server will listen on `localhost:9080`. It will be operating under your user identity. There aren't any applications deployed to that server, so there's not much you can do with that server ... yet. You're about to change that.

Take a look at the `/<user_dir>/servers/server1/server.xml` file. It should look something like this:

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>
    <httpEndpoint id="defaultHttpEndpoint"
                  host="localhost"
                  httpPort="9080"
                  httpsPort="9443" />
</server>
```

Leave the server running and make one small change<sup>12</sup> to that file:

```
<server description="new server">
    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
```

<sup>9</sup> It is possible to combine the commands: `export JAVA_HOME=<path>`

<sup>10</sup> If you have traditional WAS z/OS V8.5 installed along with Liberty, the Java supplied with traditional WAS z/OS will work perfectly well. The directory is `/java64` under the directory where Installation Manager installed WAS z/OS V8.5.

<sup>11</sup> Or you can add the Liberty install `/bin` directory to your PATH.

<sup>12</sup> The file is in ASCII but tagged, so with `_BPXK_AUTOCVT=ON` (page 10) tools such as vi, or OEDIT from OMVS or ISHELL should work.

```

</featureManager>
<httpEndpoint id="defaultHttpEndpoint"
               host="*"      ← Change that from "localhost" to an asterisk13
               httpPort="9080"
               httpsPort="9443" />
</server>

```

- The Liberty Profile server will *dynamically* pick up that change and put it into effect. That's one of the features of Liberty -- monitoring for changes and dynamic update.
- Browse the file `messages.log` (stored in ASCII) which is located at the following location:  
`/<user_dir>/servers/server1/logs`

You should see something like this (some lines trimmed here to save space):

```

The server server1 has been launched.
:
TCP .. is now listening for requests on host localhost (IPv4: 127.0.0.1) port 9080.
Monitoring dropins for applications.
Feature update completed in n.nnn seconds.
The server server1 is ready to run a smarter planet.
Starting server configuration update.
The server configuration was successfully updated in n.nnn seconds.
TCP .. stopped listening for requests on host localhost (IPv4: 127.0.0.1) port 9080.
TCP .. started and is now listening for requests on host * (IPv4) port 9080.

```

*The Liberty Profile detected your change to the server.xml file and dynamically updated the server.*

- From the WP102110 Techdoc at [ibm.com/support/techdocs](http://ibm.com/support/techdocs) you'll find a ZIP file called `WP102110-files.zip`. Download that file and extract the `ATS_servlet.war` file.
- Now upload that file in binary format to the `/<user_dir>/servers/server1/dropins` directory. Give the file 640 permissions. Again, the Liberty Profile server will recognize an update to the `/dropins` directory and will dynamically load the application.
- From a browser enter the following URL:

`http://<your_host>:9080/ATS_servlet/SimpleServlet`

Where `<your_host>` is the TCP host of your z/OS system. Case matters for the HTTP URLs, so be sure to match the case of `/ATS_servlet/SimpleServlet` as shown.

- You should get back the following:



IBM Advanced Technical Skills, Gaithersburg, Maryland

Very simple servlet to validate IBM WAS z/OS V8.5 Liberty

Current date and time is: Tue Apr 10 10:18:33 EDT 2012

Your browser's IP address is: 9.65.246.198

With the "Current date and time" equal to what your z/OS system says it is, and the "browser's IP address" being that of your workstation.

You have just invoked your first application running in a Liberty Profile server on z/OS! That application is a very simple servlet that produces a result as shown in the picture above.

<sup>13</sup> That change tells Liberty to listen on all adapters. That allows you to supply your z/OS system's host name in the browser URL and have it invoke the servlet.

- Now let's stop the server to see how that's accomplished. Enter the following command from the /<install\_dir>/bin directory:

```
server stop server1
```

You should see in response:

```
Server server1 stopped.
```

- While you're in that directory, enter the following command:

```
Server status server1
```

You should see:

```
Server server1 is not running.
```

### **Create multiple servers and take advantage of shared applications**

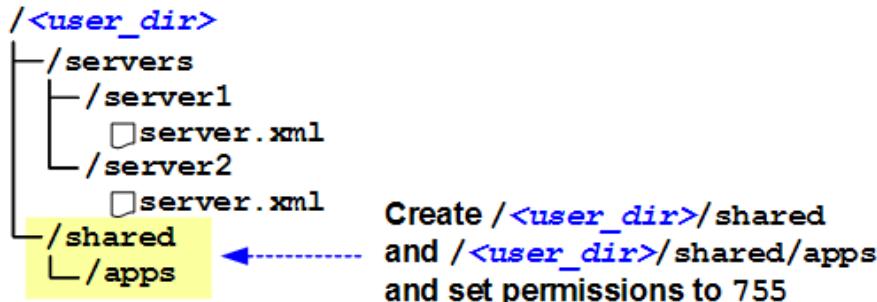
The Liberty Profile has a flexible set of built-in variables you may use to share assets, such as applications and configuration elements<sup>14</sup>, between multiple servers. In this section you'll make use of one of those variables -- \${shared.app.dir} -- so one copy of the ATS\_servlet.war file may be served from multiple servers.

- Delete the ATS\_servlet.war file from the /<user\_dir>/servers/server1/dropins directory.

- From the /<install\_dir>/bin directory enter the following command:

```
server create server2
```

- In your <user\_dir> create the following directories:



- Place the ATS\_servlet.war file in the /shared/apps directory with permission that allow your ID at least read to the file (example: 640)

- Modify the server.xml file for server1 and add one line:

```

<server description="new server">
  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
  </featureManager>
  <application location="${shared.app.dir}/ATS_servlet.war" />
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />
</server>

```

- Modify the server.xml for server2 and do the following:

- \_\_\_ Change host="localhost" to host="\*"
- \_\_\_ Change the httpPort= value to httpPort="9081" (increment by 1)
- \_\_\_ Add the exact same <application> line you added for server1

<sup>14</sup> We illustrate on page 16 the use of <include> in the server.xml file to show sharing configuration XML between servers.

- From the `/<install_dir>/bin` directory, enter the following two commands:

```
server start server1
server start server2
```

You should see a confirmation of server start for each server:

```
Server server1 started with process ID <pid>.
Server server2 started with process ID <pid>.
```

- From a browser, invoke the servlet in `server1`

```
http://<your_host>:9080/ATS_servlet/SimpleServlet
```

You should see the same servlet response as before, but with the current date and time.

- In a separate browser window or tab, invoke the servlet in `server2`

```
http://<your_host>:9081/ATS_servlet/SimpleServlet
```

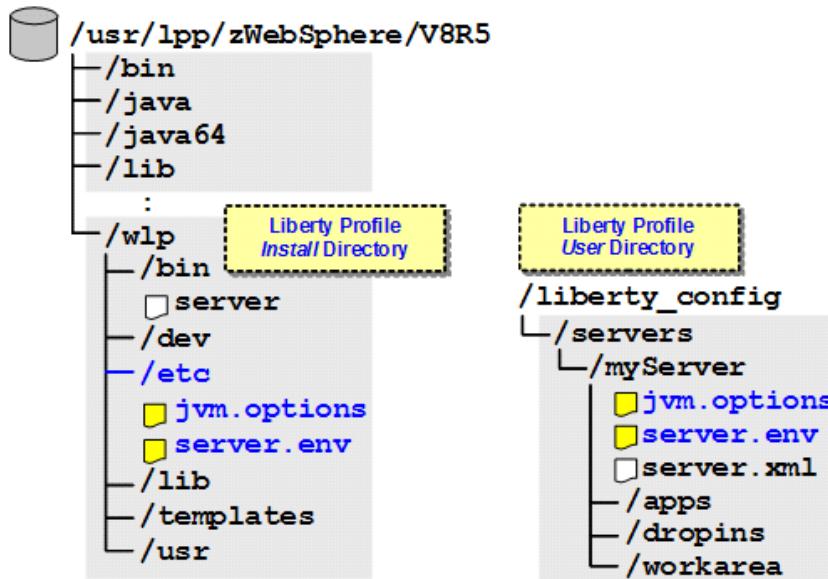
You should see the servlet response as well, but because you pointed at the port for `server2` the request went to that server rather than `server1`.

You have seen how easy it is to create a new server. And you have seen how it is possible to share an application across those servers with a simple update to `server.xml` that uses a built-in variable for the shared applications directory.

### ***Introducing the `server.env` file and the `jvm.options` file***

The `server.env` file is used to specify environment variables and their values to the server runtime environment. For example, the location of the 64-bit Java SDK to use. The `jvm.options` file is used to specify options used by the JVM.

The files may reside in one of two locations:



If files are present in both locations then the contents of the two files will be merged, with the server-level file content taking precedence over the installation level file.

**Notes:** By default the `/etc` directory under `<install_dir>` will not exist. If you wish to locate a copy of `server.env` there, you must create the directory.

In this document we are assuming the `<install_dir>` location is a Read-Only file system, so for this document we will focus on the server level files.

In this exercise you will set `JAVA_HOME` value in the `server.env` file<sup>15</sup> and the servers' HTTP ports as options in the `jvm.options` file. Then you will use variables in the `server.xml` that resolve the HTTP ports at runtime to the values set in the options file.

- From the `/<install_dir>/bin` directory, enter the commands:

```
server stop server1
server stop server2
```

The Liberty Profile does not automatically detect the presence of a new `server.env` file, nor does it dynamically pick up changes in the `server.env` file. So for this exercise a server restart is necessary.

- Create a `server.env` file<sup>16</sup> in the `/<user_dir>/servers/server1` directory and supply the following line:

```
JAVA_HOME=<path_to_64bit_Java>
```

Where `<path_to_64bit_Java>` is the same you used back on page 11. In your Telnet session enter `echo $JAVA_HOME` to see what that value was. Give it permissions 640.

- Copy that file to the `/<user_dir>/servers/server2` directory.

**Note:** There are many ways to make a valid 64-bit `JAVA_HOME` value applicable to the Liberty Profile server environment. Here we are showing each server with its own copy. Other mechanisms exist -- a shared copy under `/<install_dir>/etc` ... part of your ID's `.profile` ... or part of the `/etc/profile` system-wide file.

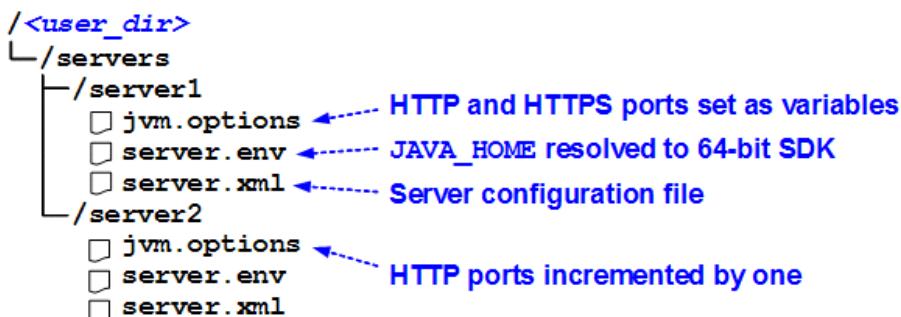
- Create a `jvm.options` file in the `/<user_dir>/servers/server1` directory, either in EBCDIC or ASCII (if ASCII, then tag `-- chtag -t -c iso8859-1 <filename>`). Give it permissions 640. Populate that file with two lines:

```
-Dhttp.port=9080
-Dhttps.port=9443
```

- Copy that file to the `/<user_dir>/servers/server2` directory. Edit the file and change the port values by incrementing each by one:

```
-Dhttp.port=9081
-Dhttps.port=9444
```

At this point your server file structure should look like this:



- Update the `server.xml` for `server1` and make two minor changes in the `httpEndpoint` section of the XML:

```
<httpEndpoint id="defaultHttpEndpoint"
              host="*"
              httpPort="${http.port}"
              httpsPort="${https.port}" />
```

<sup>15</sup> Up to this point you have relied on the `JAVA_HOME` value in your user ID's UNIX shell environment to be correct. That worked and you could continue with that approach if you wished. We will set `JAVA_HOME` in `server.env` to illustrate how it works there as well.

<sup>16</sup> Either ASCII or EBCDIC. Liberty will work with either. If ASCII, tag the file: `chtag -t -c iso8859-1 <filename>`

- Make the *exact same* change in the `server.xml` for server2.
- From the `/<install_dir>/bin` directory, enter the commands:  
`server start server1`  
`server start server2`
- From separate browser screens, invoke the servlet in each server using the unique TCP port each is listening on:

```
http://<your_host>:9080/ATS_servlet/SimpleServlet
http://<your_host>:9081/ATS_servlet/SimpleServlet
```

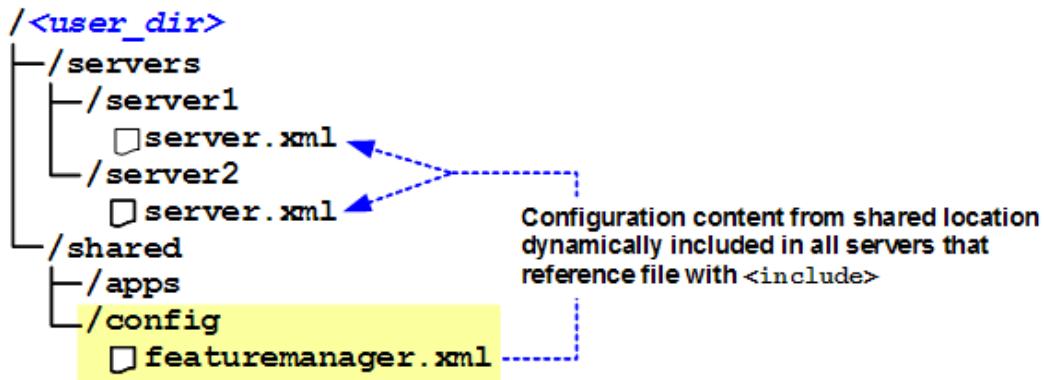
You should see the servlet response as you saw before, with the current date and time displayed. That validates that both servers are up, listening on separate TCP ports, serving the same application.

Further, each server's `server.xml` is identical at this point. The HTTP ports, which was unique in each configuration before, are now referenced with a variable set in `server.env`. That's what allows the two `server.xml` files to be the same.

Further still, both servers are referencing the *exact same* deployed application WAR file using the `<application>` tag in the `server.xml`. Remember that the Liberty Profile allows for dynamic application updates -- upload a new copy of the application WAS into the `/shared/apps` directory and Liberty will detect the new file and dynamically update the application running in *both servers*.

### Use of `<include>` in `server.xml`

Liberty Profile permits the use of an `<include>` element in the `server.xml` which allows XML to be dynamically incorporated into the configuration file. This exercise will have you replace the `<featuremanager>` section of the `server.xml` with an include, with Liberty pulling the feature definitions from a common, shared location. In other words, this:



Earlier you created the `/<user_dir>/shared/apps` directory and stored the servlet application WAR file there. In this exercise you will create the `/config` directory and place the XML file to be included in that directory.

- Leave the servers running ... the changes you are about to make can be done dynamically.
- Change directories to `/<user_dir>/shared` and create a directory called `config` with permissions that allow your ID to read and write (ex: 640).
- Create a file called `featuremanager.xml` and in that file add the following:

```

<server>
  <featureManager>
    <feature>jsp-2.2</feature>
  </featureManager>
</server>
  
```

- Store that file *in ASCII* in /<user\_dir>/shared/config with permissions that allow your ID to read the file (ex: 640).
- Tag that file: chtag -t -c iso8859-1 <filename>
- Modify the server.xml file of server1 **and** server2 and delete the XML indicated by strikethrough and add the bold blue line:

```
<server description="new server">

    <!-- Enable features -->

    <featureManager>
        <feature>jsp-2.2</feature>
    </featureManager>

    <include location="${shared.config.dir}/featuremanager.xml" />

    <application location="${shared.app.dir}/ATS_servlet.war" />

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="${http.port}"
                  httpsPort="${https.port}" />

</server>
```

**Notes:** We have introduced another built-in variable: \${shared.config.dir}, which resolves to the /shared/config directory under your <user\_dir>.

It is possible to hard-code an absolute path, but here we show the use of the variable to resolve the file location.

It is possible to <include> more than just one section of the XML. You could if you wished <include> the entire content of the server.xml. The key is understanding that the server.xml file must start and end with <server></server>, and the XML being included must *also* start and end with <server></server>.

- Go to the /<user\_dir>/servers/server1/logs directory and browse the file messages.log file. You should see something like this at the bottom of the file:

```
Starting server configuration update.
Processing included configuration resource: file:/<user_dir>/shared/config/featuremanager.xml
Feature update started.
The server configuration was successfully updated in n.n seconds.
Feature update completed in n.n seconds.
```

This is one more illustration of the dynamic nature of the Liberty Profile.

The ability to <include> gives you considerable flexibility to compose your server.xml file from different sources. In a multi-server environment this provides the ability to have some elements in the server.xml file common and some unique.

## **Enabling SSL using z/OS SAF as the keystore**

The next exercise will enable the use of SAF keyrings as the location for the SSL server certificate and the "Certificate Authority" (CA) used to sign the server certificate.

**Note:** The illustration here assumes IBM RACF, but RACF is *not* required to use the Liberty Profile for SAF work. Any SAF product will suffice.

This exercise calls for several RACF commands to create the certificates and the keyring. Work with your security administrator to determine whether entering these commands is required at your location.

- Work with your security administrator and enter the following RACF<sup>17</sup> commands, where *aaaaaaa* is the userid you logged on with, which is userid under which your Liberty Profile servers are currently running.

**Note:** Each RACDCERT command is entered as one line.

```
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(aaaaaaa) ACCESS(READ)
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty') OU('LIBERTY'))
WITHLABEL('LibertyCA.LIBERTY') TRUST NOTAFTER(DATE(2018/12/31))
RACDCERT ADDRING(Keyring.LIBERTY) ID(aaaaaaa)
RACDCERT ID (aaaaaaa) GENCERT SUBJECTSDN(CN('yourhost.com') O('IBM')
OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY') SIGNWITH(CERTAUTH
LABEL('LibertyCA.LIBERTY')) NOTAFTER(DATE(2018/12/31))
RACDCERT ID(aaaaaaa) CONNECT (LABEL('DefaultCert.LIBERTY')
RING(Keyring.LIBERTY) DEFAULT)
RACDCERT ID(aaaaaaa) CONNECT (RING(Keyring.LIBERTY)
LABEL('LibertyCA.LIBERTY') CERTAUTH)
SETROPTS RACLIST(FACILITY) REFRESH
```

---

<sup>17</sup> Or other SAF vendor equivalent commands.

- Update the server.xml of server1 with the following<sup>18</sup>:

```
<server description="new server">

    <!-- Enable features -->

    <include location="${shared.config.dir}/featuremanager.xml" />

    <application location="${shared.app.dir}/ATS_servlet.war" />

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="${http.port}"
                  httpsPort="${https.port}" />

    <sslDefault sslRef="DefaultSSLSettings" />
        <ssl id="DefaultSSLSettings"
             keyStoreRef="CellDefaultKeyStore"
             trustStoreRef="CellDefaultTrustStore" />
        <keyStore id="CellDefaultKeyStore"
                  location="safkeyring:///Keyring.LIBERTY"
                  password="password" type="JCERACFKS"
                  fileBased="false" readOnly="true" />
        <keyStore id="CellDefaultTrustStore"
                  location="safkeyring:///Keyring.LIBERTY"
                  password="password" type="JCERACFKS"
                  fileBased="false" readOnly="true" />

</server>
```

- Enable server2 with SSL by updating its server.xml with the same updates as you did for server1.
- Update the shared featuremanager.xml file you created earlier with the following:

```
<server>
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>zosSecurity-1.0</feature>
        <feature>ssl-1.0</feature>
    </featureManager>
</server>
```

- Look in the messages.log file for either (or both!) of your servers and you should see:

```
Creating the LTPA keys. This may take a few seconds.
The security service is starting...
The server installed the following features: [zosSecurity-1.0, ssl-1.0].
Feature update completed in n.nnn seconds.
TCP Channel defaultHttpEndpoint-ssl has been started and is
now listening for requests on host * (IPv4) port 9443.
LTPA keys created in n.nnn seconds. LTPA key file:
    <user_dir>/servers/server1/resources/security/ltpa.keys
LTPA configuration is ready after n.nnn seconds.
```

Which illustrates that changes to the shared include file are detected and incorporated, *including updates that result in new features that are then dynamically installed*.

- From a browser, invoke the servlet in server1

[https://<your\\_host>:9443/ATS\\_servlet/SimpleServlet](https://<your_host>:9443/ATS_servlet/SimpleServlet)

**Note:** that "s" on https is important.

---

<sup>18</sup> The password="password" string is exactly as shown ... the literal string "password" is required as the password.

Your browser will most likely challenge you because the CA certificate that signed the server certificate is not "well known" -- that is, it's not a well known CA such as VeriSign. The CA we used was self-signed certificate.

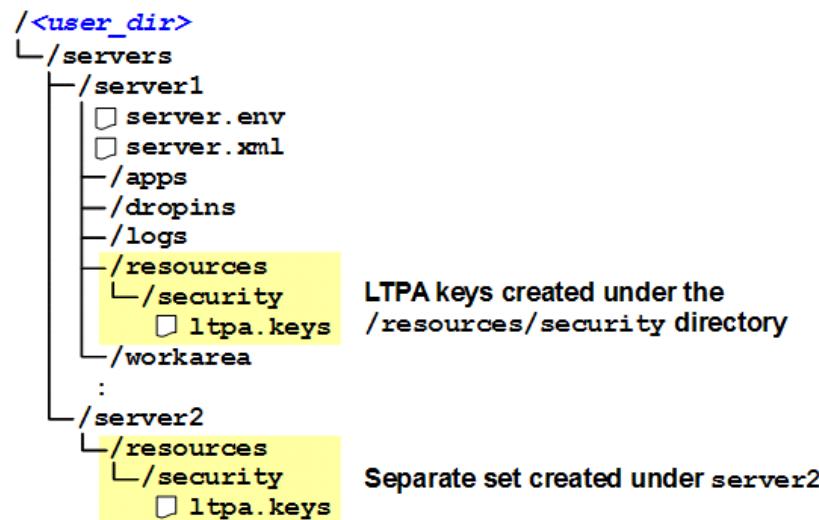
Accept the challenge and you should then see the servlet screen as you saw earlier. But this time the content will have been encrypted using SSL between the Liberty Profile server and the browser.

- Try the other server -- `https` against port 9444.

### **Establishing shared LTPA keys for single sign-on between servers**

LTPA stands for "Lightweight Third-Party Authentication." It is a mechanism used to authenticate across different physical servers without having the user re-enter ID and password each time. LTPA keys are used to encrypt a token that is passed between a client and server once identity authentication is established.

In that last exercise *each* server ended up with its own set of LTPA keys:



That is not itself a problem *unless* your goal is to treat the two servers as logically associated for the purpose of single sign-on<sup>19</sup>. If that was your goal, then you would want the two servers to *share* the LTPA keys. That way an LTPA token created by one server would be understood and trusted by the other -- because both are using the same LTPA keys and thus both are able to *decrypt* the token and trust it.

**Note:** The sample servlet application does not perform any authentication so LTPA is not actually *used* in these exercises. We are simply illustrating how to share LTPA keys across servers if that was your design intent.

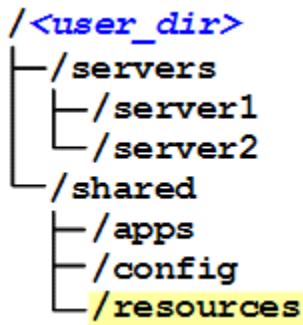
In this section you will see how to share a single `ltpa.keys` file between the servers<sup>20</sup>.

Further, you will discard the default LTPA keys and create new keys using your own password. Like the other exercises, this will be done *without* requiring a server restart.

- Create a directory called `/resources` under the `/shared` directory from earlier:

<sup>19</sup> For those interested in the topic of authentication and Liberty, see the InfoCenter, search string `cwlpa_authentication`

<sup>20</sup> You will share between two servers, but it could just as easily be 200.



Give that directory permissions to allow your ID at least read (ex: 640).

- Invoke the /<install\_dir>/wlp/bin/securityUtility shell script with the following command:

securityUtility encode <password>

where <password> is some password string of your choosing. The utility will return a string that starts with {xor} ... copy the full string to the clipboard.

- Update the server.xml for each server with the following XML:

```

<ltpa keysFileName="\${shared.resource.dir}/ltpa.keys"
      keysPassword=""
      expiration="120" />
  
```

**Note:** \${shared.resource.dir} is another built in variable that resolves to the /shared/resource directory of the <user\_dir> in which the servers are operating.

- In the messages.log file for the *first* server updated you should see:

Creating the LTPA keys. This may take a few seconds.

LTPA keys created in n.nnn seconds. LTPA key file: /<user\_dir>/shared/resources/ltpa.keys  
LTPA configuration is ready after n.nnn seconds.

The second server updated will simply show:

LTPA configuration is ready after 0.0 seconds.

- Delete the ltpa.keys file from each server's /resources/security directory.
- Important** -- when you are ready to move to the next section, stop both servers. In the next section you will start them as z/OS started tasks. If you left them up there would be a TCP port conflict when you started them *again* as a started task.

## Section summary

An objective of this section was to start you out with the simplest of environments to illustrate that getting started with the Liberty Profile server can be quite easy:

- You used your own ID
- You created the configuration user directory in your ID home directory
- You started the server

And you were on your way.

We then built upon that by deploying an application by simply dropping it into the `/dropins` directory, adding a second server, and introducing variables and includes in the `server.xml`.

We finished up by enabling SSL on your server using a z/OS SAF keyring as the keystore and truststore for the digital certificates. We followed that up with showing how the LTPA keys can be shared across servers, allowing for single sign-on.

Next we will take you into running Liberty Profile servers as z/OS started tasks.

**Note:** In the next section we will illustrate the use of JDBC Type 2 using RRS for transaction support as well as illustrate the ability to classify work and have WLM create enclaves that map to service classes and reporting classes.

*Those functions are also possible running Liberty from the UNIX shell.*

*You do **not** need to run Liberty as a z/OS started task to use those z/OS features. They are available from servers started from the UNIX shell or as a z/OS started task.*

*You may need to have the z/OS Angel process<sup>21</sup> running. We discuss that starting on page 26.*

Next up ... Liberty Profile as a z/OS started task.

---

<sup>21</sup> The Angel process provides access to z/OS *authorized* services. Not all z/OS functions require authorization. You just saw how using SAF for keyring support does not (the Angel process was not present for the earlier exercises, and none of the SAF SERVER profiles were established to provide access control). The Angel process is discussed in more detail starting on page 26.

## Liberty Profile and z/OS -- Step-by-Step Instructions

### Before you get started

In this section we will guide you through starting the Liberty Profile server as a z/OS started task. The following assumptions are in place:

- The <install\_dir> you used for the UNIX shell section will be used here as well
- The <user\_dir> you created and used for the UNIX shell section will be used here as well
- The same server definitions you created earlier will be used here as well
- The started tasks will operate under the same ID as you used in the UNIX shell section.

**Note:** The following exercises are built on the previous four assumptions being true. If you decide to use another ID or create a different server you will need to take that into account while following the instructions that come next.

You *could* use separate <install\_dir>, <user\_dir> and userid if you wished. We assume the same artifacts are re-used here to keep things simple.

The z/OS definitions needed to create a separate ID and give it access to the Angel process SERVER profiles is provided under "z/OS Definitions and Other Information" starting on page 36.

### Copy out the sample JCL procs and customize

- Locate the sample JCL start procedures at the following location. You should find *two* sample JCL start procedures<sup>22</sup>:

```
/<install_dir>/templates/zos/procs/bbgzangl.jcl  
/<install_dir>/templates/zos/procs/bbgzsrv.jcl
```

- Copy both to your preferred JCL start procedure library as BBGZANGL and BBGZSRV.
- Edit the BBGZSRV start procedure and do the following:

```
//BBGZSRV PROC PARMS='defaultServer'  
:  
//  SET INSTDIR='/<install_dir>'      Note 1  
//  SET USERDIR='/<user_dir>'        Note 2  
:
```

#### Notes:

- 1 Set this to your <install\_dir>, including the /wlp designation
- 2 Set this to your <user\_dir>

- If the server you intend to start has a `server.env` file with `JAVA_HOME` defined (as illustrated back on page 14), then you may leave the JCL's `STDENV DD` commented out. Otherwise, uncomment `STDENV DD` and point to a file containing `JAVA_HOME=` that points to the location of a 64-bit SDK.
- Enter the following RACF<sup>23</sup> commands to create the `STARTED` profile and assign your userid to the profile:

```
RDEF STARTED BBGZSRV.* UACC(NONE) STDATA(USER(cccccc)  
                                GROUP(aaaaaaaa) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))  
SETROPTS RACLIST(STARTED) GENERIC(STARTED) REFRESH
```

Where `cccccc` is the userid you used in the UNIX shell exercises and `aaaaaaaa` is the group. The `RDEF` command is entered as one line.

22 The file `bbgzsrv.jcl` holds the server start procedure. The file `bbgzangl.jcl` holds the "Angel" server proc. The Angel server provides access to authorized z/OS services. It is only needed when certain z/OS-exclusive Liberty Profile functions are employed. Initially all you will need is `bbgzsrv.jcl`, but eventually you'll need both for these exercises. Copy *both* to your proclib.

23 Or equivalent commands for SAF product from another vendor.

- Start the server with the following command:

```
START BBGZSRV,PARMS='server1'
```

PARMS= resolves to a case sensitive UNIX directory, so be certain the case you provide is correct. Be certain to enclose in single quotes as shown.

- From a browser, invoke the servlet over http and https:

```
http://<your_host>:9080/ATS_servlet/SimpleServlet
```

```
https://<your_host>:9443/ATS_servlet/SimpleServlet
```

- Stop the server with a normal MVS stop command: P BBGZSRV

- Start the server again, this time give it a unique JOBNAME value:

```
START BBGZSRV,JOBNAME=LIBSERV1,PARMS='server1'
```

- Now start the other server, using the same JCL start procedure, but with a different JOBNAME and a different PARMS= value:

```
START BBGZSRV,JOBNAME=LIBSERV2,PARMS='server2'
```

Both should see both start but under different JOBNAME values.

- From a browser, invoke the servlet in both servers over http and https:

*First server:*

```
http://<your_host>:9080/ATS_servlet/SimpleServlet
```

```
https://<your_host>:9443/ATS_servlet/SimpleServlet
```

*Second server:*

```
http://<your_host>:9081/ATS_servlet/SimpleServlet
```

```
https://<your_host>:9444/ATS_servlet/SimpleServlet
```

This illustrates that you may use the same Liberty Profile server configuration and start from either the UNIX shell or as a z/OS started task. The BBGZSRV start procedure has a relatively simple structure that points to <install\_dir> and <user\_dir>. The RACF STARTED profile is used to assign a userid to the started task.

## ***z/OS WLM Classification***

The Liberty Profile for z/OS has the ability to assign to work a transaction classification and pass that into z/OS WLM for mapping to service classes and reporting classes.

- Leave your server **active**. The changes you are about to make may be done dynamically.
- Update the `featuremanager.xml` file you created earlier in the `/shared/config` directory under your `<user_dir>`. Add the following line:

```
<server>
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
    <feature>zosWlm-1.0</feature>
  </featureManager>
</server>
```

In response you should see the following in the `messages.log` file:

```
The server installed the following features: [zosWlm-1.0].
```

- Update the `server.xml` file for `server1` with the following:

```
<server description="new server">
```

```
  <!-- Enable features -->
```

```

<include location="${shared.config.dir}/featuremanager.xml" />

<application location="${shared.app.dir}/ATS_servlet.war" />

<httpEndpoint id="defaultHttpEndpoint"
               host="*"
               httpPort="${http.port}"
               httpsPort="${https.port}" />

<wlmClassification>
  <httpClassification transactionClass="LBRTSERV"
                        host="*" port="*"
                        resource="/ATS_servlet/*" />
</wlmClassification>

[the SSL definitions found here but trimmed out of this example to save space in document]

</server>
```

The application URI<sup>24</sup> of /ATS\_servlet/SimpleServlet will match the resource= value of /ATS\_servlet/\* and be assigned a Transaction Class of LBRTSERV. The next step is to update WLM to include a rule to map that TC to a service class and reporting class.

- Work with your security administrator to make certain the ID under which your Liberty Server is operating has READ to FACILITY class BPX.WLMSERVER.<sup>25</sup>
- Work with your WLM administrator to create the following CB subsystem rule in WLM:

Action	Type	-----Qualifier-----	Name	Start	Service	-----Class-----	Report
____	1	TC <b>1</b>	LBRTSERV	____	LBRTSERV	____	<b>2</b> LBRTSERV

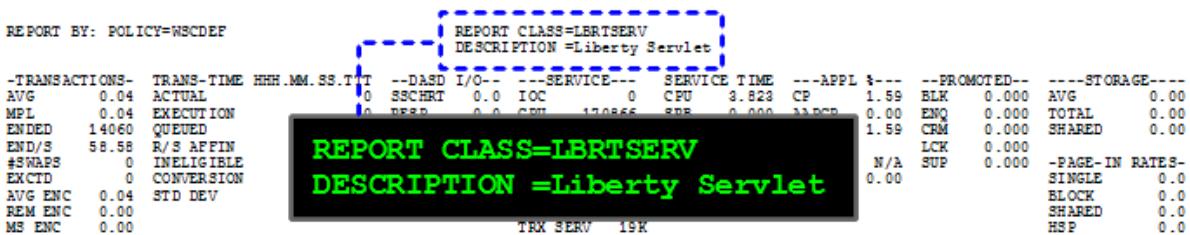
**Notes:**

1. The TC value matches the transactionClass="LBRTSERV" value in server.xml.
2. It is not necessary to use these exact service class or report class names. You may use existing class names during your testing.

- Save the rule and then install and activate the policy.
- Optional exercise* -- If you have a workload driver tool such as JMeter handy you could drive the servlet and perhaps<sup>26</sup> catch an enclave in flight using =SDSF.ENC. If so, you would see this:

NAME	SSType	Status	OwnerJob	SrvClass	Per PGN	RptClass
4400000B1E	CB	ACTIVE	BIGZSERV1	<b>LBRTSERV</b>	1	<b>LBRTSERV</b>

- Optional exercise* -- an alternative is to use RMF to report based on the report class:



<sup>24</sup> If you wished to match all URIs for an application, including compound URIs, then /contextroot/\*\*/\* would be the matching pattern.

<sup>25</sup> This is because we are *not* operating with the Angel process for this exercise. If the Angel was started *and* your ID had access to the Angel (see "Setting up server ID access to the Angel process" on page 27) then READ to BPX.WLMSERVER not required.

<sup>26</sup> The sample servlet is very simple. The WLM enclave will come and go very quickly. You need a tool to invoke the servlet many times to catch an enclave inflight such as this.

## Some closing notes about WLM classification

- In this example we illustrated WLM classification without the Angel process (which we introduce in the next section). That means the access was operating unauthorized. Each call results in SAF check. Running authorized with the Angel process has the potential to perform more efficiently as the SAF check is performed once.
- As we noted earlier, you may perform WLM classification with a Liberty Profile server started in the UNIX shell. It is not necessary to run as a z/OS started task to take advantage of WLM classification.
- The XML tag `<zosWorkloadManager collectionName="<value>" />` may be used to pass in a CN different from your server name. That would provide the ability for multiple servers using the same XML to collect under the same CN= in WLM.
- If your interest was in classifying all work in a server with the same TC regardless of the URI, then something like this would provide that:

```
<wlmClassification>
  <httpClassification transactionClass='ALLWORK' />
</wlmClassification>
```

Values for host, method and resource will default to matching everything. And then in WLM a TC= rule for ALLWORK would map to whatever service class and reporting class you desired.

## Overview of the "Angel" process

Earlier you copied the BBGZANGL JCL start procedure to your proclib. Until now you have not used it. For the next exercise -- JDBC Type 2 with RRS -- you will need it. This will imply stopping the Liberty Profile server, starting the Angel process and then re-starting the server.

The Angel process is a very lightweight function that's used as an anchor point for access to z/OS authorized services. At a very high level it starts, establishes some control blocks and then does very little other CPU-consuming work.

Here are the key points about the Angel process we wish to convey in this document:

- It is not always required. As you have already seen, the Liberty Profile server is able to start and operate without the Angel process present. In this document you have already seen SSL with SAF and WLM classification work *without* the Angel process.
- It is only *required* when an Liberty Profile server seeks to use z/OS authorized services for which there is no unauthorized path.

For example, using z/OS RRS for JDBC Type 2 transaction coordination requires authorized access, and as such the Angel process is necessary. In contrast, z/OS WLM provides both an authorized and unauthorized path, so the Angel process is not strictly required for that. However, the Liberty Profile server performance is better with WLM when the Angel is present -- a SAF authorization check is not needed for each call.

- The Angel process provides access to z/OS authorized services for Liberty Profile servers started from the UNIX shell or as z/OS started tasks.
- Only one Angel process is needed for any given z/OS operating system image. That is true regardless of the number of Liberty Profile servers you wish to run on that z/OS image.
- The Angel process has no configuration files and uses no TCP ports. This is very much *in contrast* to the traditional WAS z/OS Daemon which has both.
- The Angel process is code level neutral, which means the code the Angel is using does *not* need to be equal or higher than the code used by any of the supported Liberty Profile servers. The code used by the Angel process may also be updated *without* having the recycle the Angel process.

- During testing of Liberty it was discovered that a z/OS Language Environment (LE) APAR is needed. That APAR is OA39035. Make certain your system has that APAR before proceeding.

### **Setting up the Angel process**

- Edit the BBGZANGL JCL start procedure you copied into your proc library and make one change:

```
//BBGZANGL PROC PARMS='',COLD=N
//*
-----  
// SET ROOT='<install_dir>' ← update with your install directory, including /wlp  
:
```

- Enter the following RACF<sup>27</sup> commands to create the STARTED profile and assign your userid to the profile:

```
RDEF STARTED BBGZANGL.* UACC(NONE) STDATA(USER(cccccc)
GROUP(aaaaaaaa) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
SETROPTS RACLIST(STARTED) GENERIC(STARTED) REFRESH
```

Where `ccccc` is the userid you used in the UNIX shell exercises and `aaaaaaaa` is the group. The RDEF command is entered as one line.

### **Setting up server ID access to the Angel process**

- Access to the Angel process services is protected by a set of SAF SERVER profiles<sup>28</sup>.

Work with your security administrator to have the following definitions entered with access granted to your Liberty Profile server ID<sup>29</sup>:

```
RDEF SERVER BBG.ANGEL UACC(NONE)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(cccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(cccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.TXRRS CLASS(SERVER) ACCESS(READ) ID(cccccc)
SETROPTS RACLIST(SERVER) GENERIC(SERVER) REFRESH
```

### **Starting the Angel process and restarting the servers to access the Angel**

Servers gain access to the Angel process at server startup. A server started before the Angel is started will not have access until it is restarted.

- Stop your Liberty Profile servers that were started as z/OS started tasks earlier.
- Start the Angel process:

```
START BBGZANGL
```

Look for the following message in the held output for the Angel started task:

```
CWWKB0056I INITIALIZATION COMPLETE FOR ANGEL
```

- Restart your Liberty Profile servers. You should see the following messages in the messages.log file indicating successful access to the Angel for TXRRS<sup>30</sup>:

```
Authorized service group TXRRS is available.
Authorized service group SAFCRE is not available.
Authorized service group ZOSDUMP is not available.
Authorized service group ZOSWLM is not available.
```

You are now ready to proceed to the JDBC Type 2 and RRS exercise.

27 Or equivalent commands for SAF product from another vendor.

28 We will show only those SERVER profile to get started with JDBC Type 2 and RRS. For the full set of Angel process SERVER profile definitions, see "z/OS Definitions and Other Information" on page 36.

29 Again, this applies to server started in the UNIX shell or as a z/OS started task.

30 The others -- SAFCRE, ZOSDUMP and ZOSWLM -- are not available because to this point you haven't created the SERVER profile for them or granted the server ID READ to them.

## Preparation for JDBC Type 2 to DB2 exercises

The JDBC sample application you will use has the following front navigation screen:

**Dynamic create, populate, display and drop Table in DB2 z/OS**  
Click [here](#) to run the servlet.

**Insert Records into LIBERTY.RECORDS table in DB2 z/OS**  
Click [here](#) to run the servlet.

**Select and display Records from LIBERTY.RECORDS table in DB2 z/OS**  
Click [here](#) to run the servlet.

**Delete Records from LIBERTY.RECORDS table in DB2 z/OS**  
Click [here](#) to run the servlet.

**These three function require the LIBERTY.RECORDS database table to exist ahead of time**  
**The DDL to create the table is provided below.**

**This function does not require database tables to be present ahead of time**  
**However, your Liberty ID must have authority to CREATE, INSERT, SELECT and DROP to perform this function.**

- If you wish to use the first option, which dynamically creates and drops a data table, then work with your DB2 administrator to make sure your ID has authority to perform the functions shown in the chart.
- If you wish to use the functions that use the pre-existing LIBERTY.RECORDS table, then work with your DB2 administrator to create the artifacts using this sample DDL:

```
--DROP DATABASE LIBDB;
--DROP STOGROUP LIBSTOGP;
--COMMIT;
CREATE STOGROUP LIBSTOGP VOLUMES(<volume>) VCAT <vcat>;
CREATE DATABASE LIBDB;
COMMIT;
--
CREATE TABLESPACE LIBSPACE IN LIBDB
    USING STOGROUP LIBSTOGP
        PRIQTY      1500
        SECQTY      50
        ERASE       NO
        LOCKSIZE    ROW;
--
GRANT USE OF TABLESPACE LIBDB.LIBSPACE TO PUBLIC;
COMMIT;
CREATE TABLE LIBERTY.RECORDS
    (NAME VARCHAR(20),
     CITY VARCHAR(20),
     STATE VARCHAR(2),
     ZIP VARCHAR(5))
    IN LIBDB.LIBSPACE;
GRANT ALL ON LIBERTY.RECORDS TO PUBLIC;
COMMIT;
```

## JDBC Type 2 to DB2 with RRS transaction support

The Liberty Profile server supports TCP-based JDBC Type 4 as well. Here we will focus on JDBC Type 2 because cross-memory database access provides demonstrable benefits over a remote JDBC Type 4 call<sup>31</sup>.

**Note:** JDBC Type 2 with RRS will work from a server started in the UNIX shell. Here we are illustrating the server as a z/OS started task. The important thing to understand is that JDBC Type 2 with RRS has no unauthorized path, so the Angel process is required. That is true whether UNIX shell or z/OS started task.

- Leave your servers running. The following changes can be made dynamically. The key is having access to the Angel process established ahead of time, as you have just done.
- Update the `featuremanager.xml` file you created earlier in the `/shared/config` directory under your `<user_dir>`. Add the following line:

```
<server>
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
    <feature>zosWlm-1.0</feature>
    <feature>jdbc-4.0</feature>
    <feature>zosTransaction-1.0</feature>
  </featureManager>
</server>
```

In response you should see the following in the `messages.log` file:

The server installed the following features: [jdbc-4.0, zosTransaction-1.0, jndi-1.0].

- Add to the `server.xml` the following XML<sup>32</sup>:

```
<jdbcDriver id="DB2T2" libraryRef="DB2T2LibRef" />

<library id="DB2T2LibRef">
  <fileset dir="/<db2_install_path>/jdbc/classes/" includes="db2jcc4.jar db2jcc_license_cisuz.jar sqlj4.zip" />
  <fileset dir="/<db2_install_path>/jdbc/lib/" includes="libdb2jcct2zos4_64.so" />
</library>

<dataSource id="ds1" jndiName="jdbc/exampleDS" jdbcDriverRef="DB2T2">
  <properties.db2.jcc driverType="2"
    databaseName="<DB2_location_name>" />
</dataSource>
```

- Add the following<sup>33</sup> to the `server.env` file as a *separate line* under `JAVA_HOME`, which should be there from earlier:

`LIBPATH=/<db2_install_path>/jdbc/lib`

- Create a file called `db2jcc.properties` (at any location you prefer) and populate that file with a single line:

`db2.jcc ssid=<SSID>`

Where `<SSID>` is the DB2 subsystem ID your Liberty server will connect to.

31 See [ibm.com/support/techdocs/atstr.nsf/WebIndex/WP101476](http://ibm.com/support/techdocs/atstr.nsf/WebIndex/WP101476)

32 Here we are showing only the XML to be added, not the whole XML file, which is fairly long at this point in time.

33 Failure to have a LIBPATH environment entry results in this error: Failure in loading native library db2jcct2zos4\_64, java.lang.UnsatisfiedLinkError: db2jcct2zos4\_64 (Not found in java.library.path): ERRORCODE=-4472, SQLSTATE=null

- Add the following<sup>34</sup> to the `jvm.options` file as a *separate line*:

```
-Ddb2.jcc.propertiesFile=/<path>/db2jcc.properties
```

Where `<path>` indicates where you created the `db2jcc.properties` file.

- Determine if the DB2 z/OS SDSNEXIT, SDSNL0D2 and SDSNLOAD modules are available via LINKLST, or if you have to add them via STEPLIB. Use the following to determine how to provide STEPLIB based on how you are starting the server:

Started Task	<i>In the JCL start proc:</i>
	<pre>//STEPLIB DD DISP=SHR,DSN=&lt;hlq&gt;.SDSNEXIT //          DD DISP=SHR,DSN=&lt;hlq&gt;.SDSNL0D2 //          DD DISP=SHR,DSN=&lt;hlq&gt;.SDSNLOAD</pre>
UNIX process	<i>Export STEPLIB to your shell environment prior to issuing 'server start':</i> <pre>export STEPLIB=&lt;hlq&gt;.SDSNEXIT:&lt;hlq&gt;.SDSNL0D2:&lt;hlq&gt;.SDSNLOAD</pre>

- Stop and restart the server to pick up the changes to `server.env` and `jvm.options`.
- In the `messages.log` file you should see the following indication of successful update:
 

The `jdbcDriver DB2T2` is available.  
The `dataSource ds1` is available as `jdbc/exampleDS`.
- From the WP102110 Techdoc at [ibm.com/support/techdocs](http://ibm.com/support/techdocs) you'll find a ZIP file called `WP102110-files.zip`. Download that file and extract the `ATS_jdbc.war` file.
- For the sake of simplicity<sup>35</sup>, simply drop the `ATS_jdbc.war` file (in binary format) into the `/dropins` directory for `server1` with permissions 640. You should see the following in the `messages.log` file:
 

Application `ATS_jdbc` started in `n.nnn` seconds.

- You are now ready to drive the JDBC application. To get to the main screen enter the following URL:

`http://<your_host>:9080/ATS_jdbc/sample.html`

You should get back a screen that looks like this:



**IBM Advanced Technical Skills, Gaithersburg, Maryland**

### ATS JDBC Type 2 Sample Applications

**Dynamic create, populate, display and drop Table in DB2 z/OS**  
[Click here](#) to run the servlet.

**Insert Records into LIBERTY.RECORDS table in DB2 z/OS**  
[Click here](#) to run the servlet.

**Select and display Records from LIBERTY.RECORDS table in DB2 z/OS**  
[Click here](#) to run the servlet.

**Delete Records from LIBERTY.RECORDS table in DB2 z/OS**  
[Click here](#) to run the servlet.

<sup>34</sup> Failure to properly specify the DB2 SSID results in an error message like this: T2zos exception: [jcc] [T2zos]T2zosReusableConnection.flowConnect:initRRSAFAttach:2451: RRS Identify failed,Return Code=12, Reason Code=X00F30006,Subsystem ID:DSN ,Plan Name:,Pklist:NULLID.\*

<sup>35</sup> Or you could code an `<application>` tag and put this application in the `/shared/apps` directory as was done with `ATS_servlet.war`. That too would be dynamically detected and started.

- If the ID under which your Liberty Server is running was given authority to CREATE, INSERT, SELECT and DROP ... then click on the first hyperlink. You should see a result that looks like this:

Servlet for validating JDBC Type 2 with [IBM WAS z/OS V8.5 Liberty](#)

This servlet dynamically creates a table, populates it, selects from it, then drops it

Connection to DB2 successfully achieved

Creation of table successful, will now populate ...

Table populated ... will now select and display results

```
Table Record = MyCity0, M0
Table Record = MyCity1, M1
Table Record = MyCity2, M2
Table Record = MyCity3, M3
Table Record = MyCity4, M4
Table Record = MyCity5, M5
Table Record = MyCity6, M6
Table Record = MyCity7, M7
Table Record = MyCity8, M8
Table Record = MyCity9, M9
```

End of records

Database table successfully dropped

Connection to DB2 closed

- If you built the LIBERTY.RECORDS table then you may invoke the other four links. They produce results similar to what's shown above, except the table is fixed rather than being dynamically created and dropped.

### **SAF authentication and authorization**

- Work with your security administrator to set up the following SAF definitions<sup>36</sup>:

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC (NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS (SERVER) ACCESS (READ) ID (cccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC (NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSDUMP CLASS (SERVER) ACCESS (READ) ID (cccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC (NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSWLM CLASS (SERVER) ACCESS (READ) ID (cccccc)
SETROPTS RACLIST (SERVER) GENERIC (SERVER) REFRESH
```

- Stop and restart the Angel process and the Liberty Profile server.

- Look in the server's messages.log file. You should see the following<sup>37</sup>:

```
Authorized service group SAFCRED is available.
Authorized service group TXRRS is available.
Authorized service group ZOSDUMP is available.
Authorized service group ZOSWLM is available.
```

- Have your security administrator create the following SERVER profile to set up the default prefix for Java EE application roles as BBGZDFLT:

```
RDEF SERVER BBG.SECPFX.BBGZDFLT UACC (NONE)
PERMIT BBG.SECPFX.BBGZDFLT CLASS (SERVER) ACCESS (READ) ID (cccccc)
SETROPTS RACLIST (SERVER) GENERIC (SERVER) REFRESH
```

<sup>36</sup> The SAFCRED profile is the one needed for this section. We're adding ZOSDUMP in anticipation of the next section. ZOSWLM is there simply for completeness. We saw earlier how WLM integration worked. Angel process authorization is not required, but is beneficial from an efficiency perspective.

<sup>37</sup> Now that the SERVER profiles are set up and the ID granted READ, these services are available.

- Update the `featuremanager.xml` file you created earlier in the `/shared/config` directory under your `<user_dir>`. Add the following line:

```

<server>
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>zosSecurity-1.0</feature>
        <feature>ssl-1.0</feature>
        <feature>zosWlm-1.0</feature>
        <feature>jdbc-4.0</feature>
        <feature>zosTransaction-1.0</feature>
        <feature>appSecurity-1.0</feature>
    </featureManager>
</server>

```

In response you should see the following in the `messages.log` file:

The server installed the following features: [appSecurity-1.0].

- Now add the following to the `server.xml` file for the server:

```

<safRegistry id="saf" />
<safAuthorization id="safAuthz" />

```

**Notes:** Those are *in addition* to what's already in the `server.xml` file.

The `safRegistry id=` value may be anything you like.

The `safAuthorization id=` value may be anything you like.

- Have your security administrator create the following `EJBROLE` profile and grant your server ID<sup>38</sup> READ access to it<sup>39</sup>:

```

RDEF EJBROLE BBGZDFLT.ATS_secure_servlet.servletRole UACC (NONE)
PERMIT BBGZDFLT.ATS_secure_servlet.servletRole CLASS (EJBROLE)
                                         ACCESS (READ) ID (cccccc)
SETROPTS RACLIST (SERVER) GENERIC (SERVER) REFRESH

```

- In the ZIP file that's included with the WP102110 Techdoc there is an application WAR file called `ATS_secure_servlet.war` ... drop that into your server's `dropins` directory so Liberty may auto-deploy it.
- You are now ready to drive the application and see the authentication mechanism work. To get to the main screen enter the following URL:

`http://<your_host>:9080/ATS_secure_servlet/SimpleSecureServlet`

- You should see an "Authentication Required" panel:



<sup>38</sup> Really whatever ID you wish to use when authenticating into the application. Here the Liberty server ID will work for this demonstration.

<sup>39</sup> The SAF commands are entered as one line. EJBROLE definitions are *case sensitive*.

- Enter the ID and password that was granted `READ` to the `EJBROLE` you set up earlier. You should then get a screen that looks a lot like the simple servlet from earlier:



**IBM Advanced Technical Skills, Gaithersburg, Maryland**

Very simple servlet to validate **IBM WAS z/OS V8.5 Liberty**

Current date and time is: **Tue Dec 04 17:24:05 GMT 2012**

Your browser's IP address is: **9.80.13.190**

- Close the browser, re-open it and try the URL again, this time providing an ID and PW that's *not* allowed access to the application `EJBROLE`. You should see "Error 403: Authorization Failed" on the browser, and (if RACF) an `ICH408I` message on the console indicating a failed access intent.

### ***z/OS MODIFY***

Three actions are enabled with the `MODIFY` command:

- Change the trace specification level.
- Request an `SVCDUMP`
- Request a Java transaction dump (`TDUMP`)

Check with your security administrator to make certain the ID under which the Liberty Profile server is running has the authority to allocate and catalog the dump data sets. If you see `ICH408I` errors in the system log it indicates a problem with insufficient access authority.

Two simple exercises:

- Issue the following command:  
`F <jobname>, SVCDUMP`  
Look for the resulting dump data set.
- Issue the following command:  
`F <jobname>, TDUMP`  
Look for the resulting dump data set.

---

## Catalog of common error symptoms and their causes

### *Errors starting server*

#### **Server name specified in PARMS= not found**

##### *External Symptom:*

Server fails to start. RC=768.

##### *Output:*

```
in STDERR:  
CWWKB0222E Server configuration directory <dir> does not exist  
CWWKB0209E Unable to determine SERVER_CONFIG_DIR
```

##### *Cause:*

The essential problem is that the path to the server directory is not found. There are many reasons why this might occur:

- The string supplied to PARMS= is misspelled
- No PARMS= was supplied on the START command and default was taken
- There is a case mis-match between string in PARMS= and actual directory
- The server definition simply doesn't exist
- The SET USERDIR= variable in the BBGZSRV proc points to a user directory different from where the server exists
- There is a permissions error and the ID of the job can't access the directory

#### **Location of Java SDK not resolved**

##### *External Symptom:*

Server fails to start. RC=1024.

##### *Output:*

```
in STDERR:  
CWWKB0234E JAVA_HOME location <java_dir> does not exist  
CWWKB0210E Failed to resolve JAVA_HOME
```

##### *Cause:*

Liberty requires access to a valid 64-bit Java SDK. Liberty uses the value of JAVA\_HOME in the UNIX shell environment to determine where Java is located. The variable JAVA\_HOME may be set in server.env file, or set in the profile of the ID under which Liberty is running. It must point to a valid installation of a 64-bit Java 6 or Java 7 SDK.

#### **Location of Java SDK resolves to a 31-bit Java SDK**

##### *External Symptom:*

Server fails to start. RC=2560.

##### *Output:*

```
in STDERR:  
CWWKB0215E Failed to open /<java_dir>/libjvm.so : CEE3587S A call was  
made to a function in the AMODE 31 DLL /<java_dir>/libjvm.so from an  
AMODE 64 caller.
```

##### *Cause:*

Liberty requires 64-bit Java. The JAVA\_HOME environment variable is resolving to a 31-bit Java.

## Required `server.xml` file not present in server directory

*External Symptom:*

Server fails to start. RC=768.

*Output:*

in STDERR:

```
CWWKB0224E Server configuration /<path_to_server>/server.xml does not exist  
CWWKB0209E Unable to determine SERVER_CONFIG_DIR
```

*Cause:*

The `server.xml` file is a required configuration file. It must be present and accessible by the ID under which the server operates.

## Errors related to `server.xml` configuration

### File `server.xml` stored on z/OS in EBCDIC rather than ASCII

*External Symptom:*

Server starts but otherwise does not serve applications

*Output:*

in STDERR:

```
CWWKG0014E: The configuration parser detected an XML syntax error while  
parsing the root of the configuration and the referenced configuration  
documents. Error: An invalid XML character (Unicode: 0x4c) was found in the  
prolog of the document. File: file:/<path>/server.xml Line: 1 Column: 1
```

*Cause:*

The `server.xml` file is a required configuration file and is expected to be in ASCII.

### Incorrect XML syntax in `server.xml`

*External Symptom:*

Server starts but otherwise does not serve applications

*Output:*

in STDERR:

```
CWWKG0014E: The configuration parser detected an XML syntax error while  
parsing the root of the configuration and the referenced configuration  
documents. Error: Element type "featureManager" must be followed by either  
attribute specifications, ">" or "/>". File: file:/<path>/server.xml Line:  
5 Column: 9
```

*Cause:*

In this example the error in the `server.xml` was the lack of a closing > bracket on the `<featureManager>` tag. The message provides a pretty clear indication of the nature of the error and the location within the file.

## **z/OS Definitions and Other Information**

### **Userid and group**

```
ADDGROUP aaaaaa OMVS(GID(bbbbbbb))
ADDUSER ccccccc DFLTGRP(aaaaaaaa) OMVS(UID(ddddddd) HOME(/eeeeeee)
                           PROGRAM(/bin/sh)) NAME('Liberty') NOPASSWORD NOOIDCARD
ALU ccccccc PASSWORD(cccccccc) NOEXPIRE
```

**Note:** The two ADDUSER command is issued as one line.

### **STARTED profiles**

```
RDEF STARTED BBGZANGL.* UACC(NONE) STDATA(USER(cccccccc)
                                             GROUP(aaaaaaaa) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
RDEF STARTED BBGZSRV.* UACC(NONE) STDATA(USER(cccccccc)
                                             GROUP(aaaaaaaa) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
SETROPTS RACLIST(STARTED) GENERIC(STARTED) REFRESH
```

**Note:** The two RDEF STARTED commands are issued as one line.

### **SERVER profiles (for access to Angel process)**

```
RDEF SERVER BBG.ANGEL UACC(NONE)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(cccccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(cccccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER) ACCESS(READ) ID(cccccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSWLM CLASS(SERVER) ACCESS(READ) ID(cccccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.TXRRS CLASS(SERVER) ACCESS(READ) ID(cccccccc)
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSDUMP CLASS(SERVER) ACCESS(READ) ID(cccccccc)
SETROPTS RACLIST(SERVER) GENERIC(SERVER) REFRESH
```

### **EJBROLE profiles (for security authentication and authorization)**

```
RDEF SERVER BBG.SECPFX.BBGZDFLT UACC(NONE)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(cccccccc)
RDEF EJBROLE BBGZDFLT.ATS_secure_servlet.servletRole UACC(NONE)
PERMIT BBGZDFLT.ATS_secure_servlet.servletRole CLASS(EJBROLE)
                                         ACCESS(READ) ID(cccccccc)
SETROPTS RACLIST(SERVER) GENERIC(SERVER) REFRESH
```

### **Installation of Liberty with WAS z/OS 8.5.5**

With WebSphere Application Server for z/OS 8.5.5 the installation of Liberty Profile changes a little bit. It is still included as a part of the WAS z/OS license, but the location is different and the Installation Manager syntax changes.

Earlier in this document we showed Liberty as an Installation Manager *feature*, and an installation location of */wlp* under wherever you installed WAS z/OS.

With 8.5.5 Liberty is called out as a separate package, and you specify its own location:

```
Separate package name
imcl install com.ibm.websphere.liberty.zos.v85,
liberty,embeddablecontainer,extprogmodes +
-installationDirectory /usr/lpp/zWebSphere/Liberty
Separate install directory from WAS itself
```

What then gets installed is essentially the same as shown before, with a /bin directory, a /lib directory, a /templates directory and so forth:

```
_ Dir bin
_ Dir clients
_ File Copyright.txt
_ Dir dev
_ Dir java
_ Dir lafiles
_ Dir lib
_ File README.TXT
_ Dir templates
```

**Note:** The `java` directory was something we added to our installation, but it does not automatically get created when you install Liberty 8.5.5.

Liberty requires a 64-bit SDK, and you may use whatever 64-bit z/OS Java installation you wish.

In this case we performed a separate IM install for Java and placed the install path *inside* our Liberty install location. That simply located the Java files with the Liberty files into a handy co-located bundle. There's no functional advantage to this ... it's just something we did.

All the steps we outlined earlier to create the server, start and stop the server, and use z/OS functions is the same.

---

## Information Center Links

<i>Topic</i>	<i>Specific Search String</i>
Liberty Profile Directories	rwlp_dirs
Liberty Profile server command	rwlp_command_server
XML elements in server.xml	rwlp_metatype_4ic
Features in <featureManager>	rwlp_feat
WLM classification	rwlp_wlmclassification
JDBC Type 2 and RRS	twlp_using_DB2JDBCtype2drv_zos
SAF definitions for authorized access	twlp_config_security_zos

## Document Change History

Check the date in the footer of the document for the version of the document.

---

<i>April 23, 2012</i>	Original document at time of Techdoc creation
<i>April 25, 2012</i>	Techdoc number added and PDF republished to WP102110
<i>May 16, 2012</i>	Fixed minor error where RACF RDEF definition did not exactly match the PERMIT that followed. This was under "Setting up server ID access to the Angel process" on page 27.
<i>June 1, 2012</i>	Several updates based on feedback from early reviewers <sup>40</sup> : <ul style="list-style-type: none"> <li>• Clarified how to specify <code>JAVA_HOME</code> when using JCL</li> <li>• Corrected JDBC Type 2 information to show how to specify <code>LIBPATH</code> to the native modules of DB2 as well as to show how to specify the DB2 SSID value</li> </ul>
<i>July 4, 2012</i>	Miscellaneous updates related to typographical errors <sup>41</sup> .
<i>November 27, 2012</i>	Fixed incorrect reference to <code>jvm.properties</code> ... proper file name is <code>jvm.options</code> .
<i>December 4, 2012</i>	A small handful of trivial typo corrections, plus the addition of the section on SAF application authentication and authorization. Many thanks to Elisa Ferracane of WebSphere Security Development for her guidance and expertise on that topic.
<i>July 17, 2013</i>	Added a footnote and section on installation of Liberty for 8.5.5. The syntax in Installation Manager (IM) is a bit different and the install location becomes configurable rather than <code>/wlp</code> under the WAS install root. See "Installation of Liberty with WAS z/OS 8.5.5" on page 36.
<i>July 28, 2014</i>	Modified the XML syntax for SAF-based SSL (page 19). Original document had:
	<pre>&lt;sslDefault sslRef="DefaultSSLSettings" &gt;   : &lt;/sslDefault&gt;</pre>
	and the proper syntax is:
	<pre>&lt;sslDefault sslRef="DefaultSSLSettings" &gt;   : &lt;/sslDefault&gt;</pre>
	The earlier syntax was incorrect but overlooked by Liberty Profile in earlier releases. With later releases this surfaced as an issue that prevented proper operations. Therefore, the correct syntax – for all releases of Liberty Profile – is what we now show.
<i>July 25, 2016</i>	Updated the SAF SSL section to add the command for LISTSTRING access. Also updated the JDBC T2 section with instructions for STEPLIB to the DB2 modules. Credit to Andrew Mattingly of IBM Australia for those updates.

---

End of WP102110

<sup>40</sup> Thanks to Carl Farkas, Paola Bari and Judy Viccica

<sup>41</sup> Thanks to Edward McCarthy