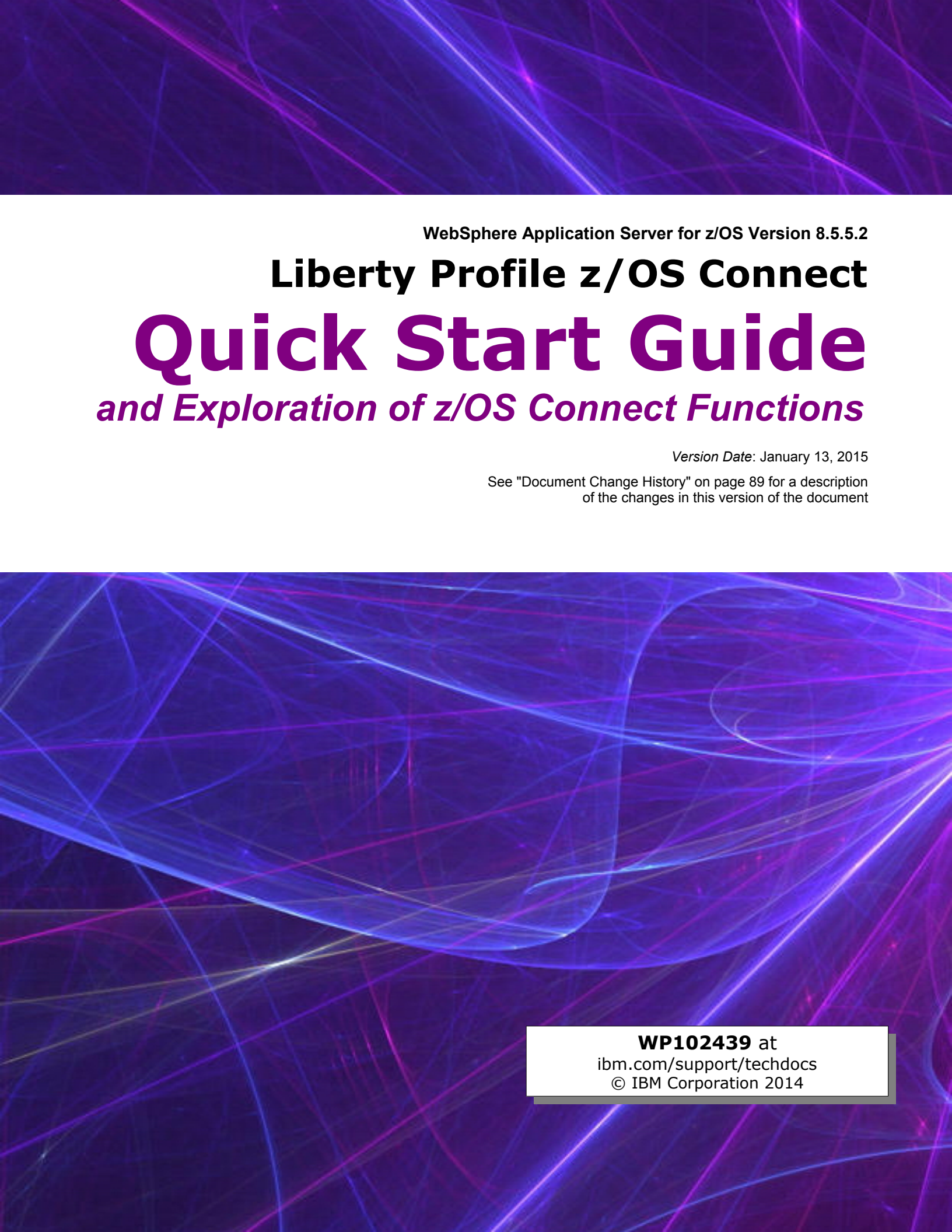




WebSphere Application Server for z/OS Version 8.5.5.2

Liberty Profile z/OS Connect

Quick Start Guide

and Exploration of z/OS Connect Functions

Version Date: January 13, 2015

See "Document Change History" on page 89 for a description
of the changes in this version of the document

WP102439 at
ibm.com/support/techdocs
© IBM Corporation 2014

Many thanks to **Ed Mezarina, David Follis, Jim Mulvey, Dan Belina, Rob Alderman, Dan Kulik** and **Dave Sudlik** of the IBM WebSphere z/OS Development and Test Lab.

Thanks to **Carl Farkas** of IBM France for review and feedback.

The IBM Gaithersburg WAS z/OS support team consists of **Mike Cox, Mike Kearney** and **Don Bagwell**

Table of Contents

Overview	5
What is z/OS Connect?	5
Three delivery paths and the focus of this document	5
Important prerequisites	6
How this document is structured	6
Some sections are required, others optional	6
The IBM Knowledge Center	6
Section 1: Preliminary Setup and Feature Validation	7
A little background in Liberty Profile for z/OS	7
Install 8.5.5.2 and required iFixes	7
Plan the Liberty Profile server environment	8
Install features using featureManager	8
Create server	10
Setup Liberty Profile server and Angel process to run as a z/OS started tasks	12
Create the IDs and Group	12
Change Liberty Profile configuration directory and file ownerships	12
Copy JCL start procedures to proclib and modify	13
Create server.env and provide JAVA_HOME variable to environment	13
Create SAF STARTED profiles for Angel process and Server	14
Start the Angel process as a started task	14
Start server as a started task	14
Reserve the TCP ports	15
Status at this point	15
Section 2: Enabling WOLA	16
Enable WOLA	16
Configuration updates to support WOLA	16
SAF SERVER updates to support WOLA	17
Start server and check for key validation messages in log	18
Sample Java application to validate WOLA	18
Use sample COBOL batch to register into Liberty using WOLA	19
Copy WOLA modules from file system to load library data set	19
Compile sample COBOL batch	21
Create SAF CBIND profile	21
Run COBOL batch and validating using sample Java program	22
Enable WOLA support in a CICS region	24
Configuration checkpoint: server.xml at this point	27
Recap of the WOLA validation	28
Status at this point	28
Section 3: Exploring z/OS Connect Functionality	29
Level-set before we begin	29
Set up basic registry and validate access to z/OS Connect	29
Slight detour: install HTTP client that can send REST with JSON	31
Configure the COBOL batch as a z/OS Connect service	32
Update server.xml with service and service provider definitions	32
Discover your new service	33
Run the batch job and have it register into the Liberty Profile z/OS server	33
Invoke service using REST client	34
Configure service with invokeURI	35
Invoke CICS OLACB01 using REST client	37
Data transformation and CICS as a backend	39
Overview of the process	40
BBGLS2JS not supplied with 8.5.5.2	41
Set up data conversion utility structure and generate bind files	41
Update server.xml for and provide data conversion for the CICS service	42
Validate using REST client	43
Exercise some of the z/OS Connect REST APIs	44
Enable the audit interceptor	47

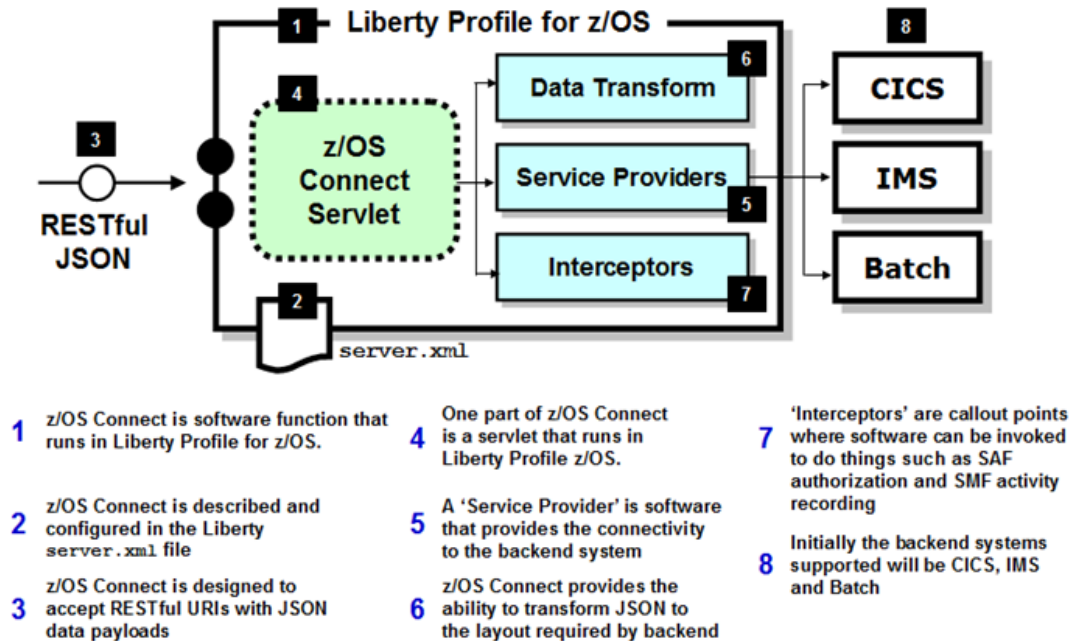
Verify system requirements to write SMF 120.11 records.....	47
Update server.xml with interceptor information.....	47
Drive the REST service a few times.....	49
Use SMF browser to see the records.....	49
Configuration checkpoint: server.xml at this point.....	51
Status at this point.....	51
Section 4: More security - SAF, Authentication, and Client Certificates.....	52
Use SAF for SSL keystore and user registry.....	52
Enable SAF as keystore for CA and server certificates.....	52
Remove basicRegistry and use SAF for registry and zosConnectAccess role checking.....	53
Enable the authorization interceptor.....	57
Review: a more granular security model.....	57
Userids and access to the zosConnectAccess role.....	58
Create Authorization Groups.....	58
Review: first set of authorization exercises.....	59
Connect users to the authorization groups.....	59
Changes to the server.xml in support of the global authorization scenario.....	59
Validation of global authorization interceptor.....	61
Service-level authorization.....	65
Configuration checkpoint: server.xml at this point.....	68
Use SAF and client certificates.....	68
Create client certificate and export certificates from SAF.....	68
Download and import CA certificate and client certificate into workstation.....	69
Validate browser recognizes the certificates.....	71
Update server.xml to prompt for certificate.....	72
Validate use of certificate to authenticate into z/OS Connect.....	73
Configuration checkpoint: server.xml at this point.....	75
Miscellaneous Information.....	76
SAF definitions.....	76
Overview of the SAF security model.....	76
ID and Group.....	77
STARTED for Liberty Profile Angel process and server.....	77
SERVER profiles.....	77
CBIND for registration into server.....	78
SAF CA Certificate and Server Certificate signed by CA.....	78
APPL class profiles and the unauthenticated userid and group.....	78
EJBROLE profiles to control access.....	78
CA certificate and the Server certificate.....	79
Client certificate and exporting certificates for download to workstation.....	79
The security profile prefix value.....	79
Workaround to do basic authentication using local repository.....	80
Angel process and the new WOLA support.....	81
Reference: server.xml examples.....	82
Checkpoint 1: End of WOLA Enablement.....	82
Checkpoint 2: After audit interceptor configuration.....	82
Checkpoint 3: after authorization interceptor configuration.....	84
Checkpoint 4: after client certificates for authentication.....	85
Knowledge Center Links.....	88
Document Change History.....	89

Overview

The purpose of this guide is to provide you a single document that takes you through the configuration, validation and simple usage¹ of WebSphere Liberty z/OS Connect. This guide is designed to provide detailed step-by-step instructions along with validation activities to insure you're on the right track. Along the way we'll show pictures illustrating key elements of the process.

What is z/OS Connect?

z/OS Connect is a feature² of WebSphere Liberty Profile z/OS that provides a REST and JSON interface (or a “gateway”) to z/OS programs and applications³:



That picture comes from the WP102439 Techdoc⁴ at ibm.com/support/techdocs, specifically the “z/OS Connect Presentation” located there. That Techdoc provides information that explains the *what* and *why* questions more fully. This document is dedicated to the *how* question – *how* to implement and use.

Three delivery paths and the focus of this document

z/OS Connect is provided under the license entitlement of three different products:

1. With WebSphere Application Server for z/OS Version 8.5.
2. With CICS Transaction Server 5.2. See IBM US Software Announcement 214-107. This function was delivered to CICS TS 5.2 via DType APAR PI25503.
3. With IMS. See IBM US Software Announcement 214-220.

This document is focused on implementing for the *first* delivery mechanism – that is, you have a license for WebSphere Application Server for z/OS V8.5.

Note: z/OS Connect is *in essence* the same across all three. However, some of the step-by-step activities to implement may be different. The delivery with CICS is somewhat unique in that it will operate in Liberty *inside* a CICS region, and use the CICS JCICS interface to access CICS programs. The delivery with IMS is very similar to WAS, with the exception being the means of accessing IMS will be through IMS Connect rather than the local adapters.

¹ This document will *not* attempt to show every configuration and use case. Some, yes; most or all, no.

² It is a no-fee *feature* of WebSphere Liberty Profile z/OS, not a separate product. Liberty Profile is itself entitled with WAS z/OS license.

³ “Batch” in this picture is a long-running started task that uses the WOLA “host a service” API. z/OS Connect does not start the batch job; the “batch” in this picture must be started and registered into Liberty Profile using WOLA.

⁴ <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102439>

Important prerequisites

Liberty Profile z/OS has a requirement for z/OS 1.11 or later.

If you are using z/OS 1.11, 1.12 or 1.13⁵, there is an important APAR you must make sure you have on your system: APAR 0A3905. See the following website for more on that APAR:

<http://www.ibm.com/support/docview.wss?uid=isg10A39035>

In addition, WebSphere Liberty Profile z/OS 8.5.5.2 or higher is required, along with two interim fixes (iFixes)⁶. See “Install 8.5.5.2 and required iFixes” on page 7 for more on those iFixes.

How this document is structured

This document will have a combination of information and 'to-do' activities. You can spot the to-do activities because they will have a checkbox next to them:

- ☐ Indicates some action you are to take
 - ☐ A sub-action to take under the square checkbox activity

Informational text will not have checkbox or check-circles.

Some sections are required, others optional

To help you understand what sections of this document are “required” and which are “optional,” the following guide is offered:

Section	Page	Comments
Section 1: Preliminary Setup and Feature Validation	7	This is essential setup work to get z/OS Connect and the Local Adapters feature installed and operational in your Liberty Profile environment.
Section 2: Enabling WOLA	16	WOLA is used by z/OS Connect to access backend regions, so enabling and validating WOLA is an essential step as well.
Section 3: Exploring z/OS Connect Functionality	29	This is where z/OS Connect is actually used. We show connectivity to both a long running COBOL process and a CICS region. We show the SMF 120.11 audit interceptor as well.
Section 4: More security - SAF, Authentication, and Client Certificates	52	This is somewhat optional. In this section we move some things from the Liberty server.xml down into SAF, and we explore the authentication interceptor function and show client certs rather than basic authorization.

The IBM Knowledge Center

Much of the material in this Quick Start was derived from the IBM Knowledge Center for WebSphere Liberty z/OS Connect.

The topic page for z/OS Connect can be found here:

ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_setup_zosconnect.html

That is a long URL. For those reading this from a PDF, here is a link → [Knowledge Center](#).

Under “Knowledge Center Links” on page 88 we offer a series of unique search strings that will take you to the specific topic page. The search strings may be entered in the Knowledge Center's search field. Then a link to the article is provided, for example:

[Using z/OS Connect REST APIs - IBM](#) 

Jun 9, 2014 ... [ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-mp&t](#)

Date: May 1, 2014 | Found in: [WebSphere Application Server Network Deployment 8.5.5](#)

⁵ The APAR fix is included in z/OS 2.1 and higher.

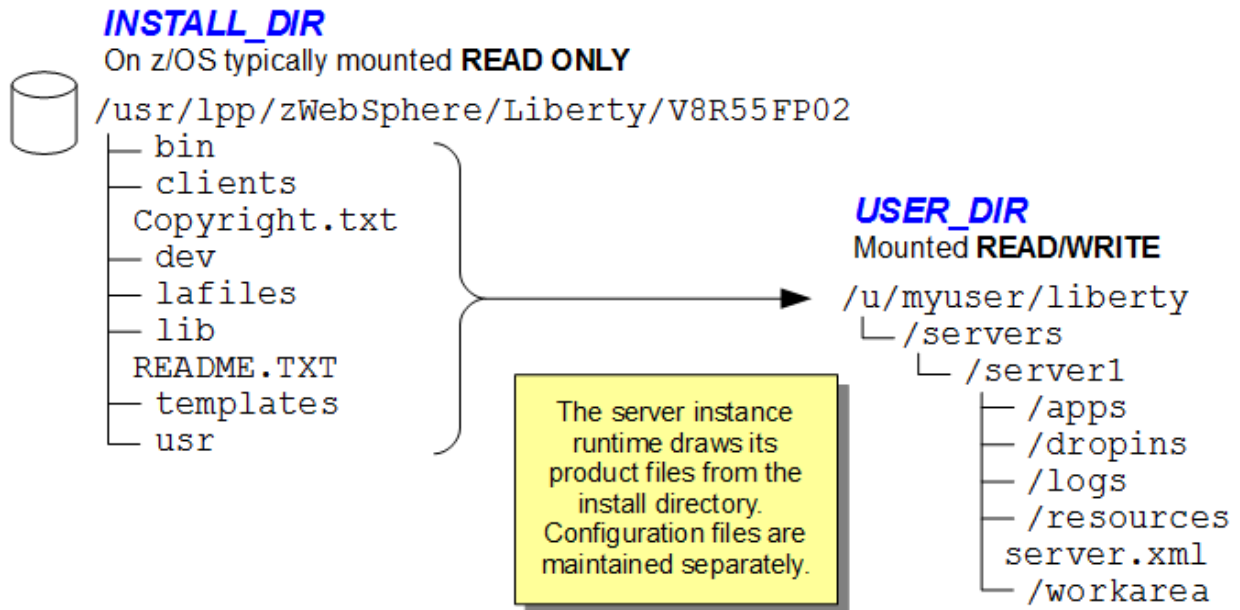
⁶ 8.5.5.3 is scheduled to GA on August 18, 2014. The iFixes will be included in that level of WebSphere Liberty Profile z/OS.

Section 1: Preliminary Setup and Feature Validation

The z/OS Connect feature (as well as the local adapter feature) are not automatically available to WebSphere Liberty Profile z/OS – they need to be enabled. It's a simple process which we'll cover in this section.

A little background in Liberty Profile for z/OS

When discussing Liberty Profile, it's necessary to keep separate the notion of the *install file system* from the *user directory*. The install file system is where the product files reside; the user directory is where a Liberty Profile server's⁷ configuration and runtime files reside:



A few notes on this:

- The WP102110 Techdoc⁸ covers Liberty Profile for z/OS, and provides a separate "Quick Start Guide" on Liberty Profile z/OS. If you're interested in digging deeper into Liberty Profile z/OS in general, consult that Techdoc.
- Liberty Profile provides a way to create an archive (for example, a PAX file) that contains *both* the configuration files *and* the install files in one portable file⁹. In this document we're not talking about that. In this document we're illustrating the install directory and user directory as separate.

The reason we highlight this is because as part of the initial setup work to enable z/OS Connect we will have you work against the **INSTALL_DIR** files. Then later we will have you work against the **USER_DIR** files. Keeping them separate in your mind helps.

Install 8.5.5.2 and required iFixes¹⁰

z/OS Connect is provided initially in WebSphere Application Server z/OS Liberty 8.5.5.2. To fully support using z/OS Connect, two interim fixes ("iFixes") are required – PI18279 and PI18379. From an IBM Installation Manager perspective, they are known as:

8.5.5.2-WS-WLP-OS390-IFP**I18279**_8.5.5002.20140522_1502

8.5.5.2-WS-WLP-OS390-IFP**I18379**_8.5.5002.20140522_1030

⁷ Or *servers*, plural ... it's possible to have multiple server instances defined under a single user directory.

⁸ <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110>

⁹ It's a handy way to distribute a complete Liberty Profile runtime to other servers, or in the case of z/OS, other LPARs.

¹⁰ Or, if at the time of reading 8.5.5.3 is available, then that will include the iFixes.

Do the following:

- ☐ Using IBM Installation Manager¹¹, install WebSphere Liberty Profile z/OS 8.5.5.2
- ☐ Using IBM Installation Manager, install the two iFixes listed earlier
- ☐ If your local conventions install first into a /Service location, copy the file system and mount **READ/WRITE** at the location that will be your `INSTALL_DIR`. If you install directly to your `INSTALL_DIR`, then make sure that's mounted READ/WRITE.

At this point you should have WebSphere Liberty z/OS 8.5.5.2 + the two iFixes.

Plan the Liberty Profile server environment

To make the configuration process easier, a little planning up-front will help. Come up with values for the following:

☐	ID under which Liberty Profile will run	<code>liberty_id =</code>
☐	ID under which the Angel process will run	<code>angel_id =</code>
☐	Group to which Liberty ID and Angel ID will connect	<code>lib_group =</code>
☐	UNIX path where Liberty configuration will reside	<code>WLP_USER_DIR=</code>
☐	UNIX path to valid 64-bit Java SDK	<code>JAVA_HOME=</code>

Notes:

- The IDs and Group may be new, or you may re-use existing.
- The UNIX path to the Liberty Profile configuration may reside anywhere you like. The ID you choose to run Liberty under must have WRITE to that directory.
- The 64-bit Java SDK may be IBM Java SDK6 or SDK7. It must be 64-bit.

Install features using featureManager

There's some work that needs to be done against the `INSTALL_DIR`, which is why we had you mount the file system READ/WRITE¹². That work involves installing two features into your installation of Liberty Profile – the z/OS Connect feature, and the Optimized Local Adapters feature¹³.

Do the following:

- ☐ Take a look at the ownership of the directories and files in your `INSTALL_DIR`. You will need an ID that has WRITE permission within that directory structure. Determine which ID you will use.
- ☐ Log into OMVS, Telnet or SSH session using the ID determined in the previous step.
- ☐ Go to the `/INSTALL_DIR/bin` directory.
- ☐ Enter the command `echo $JAVA_HOME`. Do you get a response to a valid 64-bit SDK? If not, then enter the command `export JAVA_HOME=<SDK_path>` to point to a 64-bit SDK installation on your system.
- ☐ Invoke the `featureManager` script with this command:

```
./featureManager featureList /<path>/features.txt
```

¹¹ See Techdoc <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102014> for more on IM and z/OS.

¹² Once the setup work is complete, the `INSTALL_DIR` may be mounted READ ONLY.

¹³ You don't need the Optimized Local Adapter function initially, but you will need it before the end of this document. We're having you install both at this point just to get that work done and out of the way.

- The file produced will be an ASCII XML file, and it will contain a listing of all the features for that installation instance of Liberty. Browse and look for the following two strings:

```
<feature name="zosConnect-1.0">
<feature name="zosLocalAdapters-1.0">
```

Note: Neither should be present. That indicates the features need to be installed. That is done with the featureManager script as well.

- Install the z/OS Connect feature with the following command¹⁴
`./featureManager install zosConnect-1.0 --when-file-exists=ignore`
 Then:
 - It will prompt you to accept the license ... enter 1
 - After a few moments it will go through a process of validating features¹⁵, and finally it will show

```
CWWKF1017I: The feature has been successfully installed.
```

- Install the z/OS Local Adapters feature with the following command:
`./featureManager install zosLocalAdapters-1.0 --when-file-exists=ignore`
 Then:
 - Enter 1 to accept the license
 - Look for the CWWKF1017I message indicating successful install.
- Run the command to list out the installed features, this time providing a new file name:
`./featureManager featureList /<path>/features2.txt`
- Browse that file and look for the following two strings:

```
<feature name="zosConnect-1.0">
<feature name="zosLocalAdapters-1.0">
```

This time you should see them:

```
</kernelFeature>
<feature name="zosConnect-1.0">
  <symbolicName>com.ibm.websphere.appserver.zosConnect-1.0</symbolicName>
  <displayName>z/OS Connect</displayName>
  <description>This feature allows users to access z/OS resources.</description>
```

and

```
<feature name="zosLocalAdapters-1.0">
  <symbolicName>com.ibm.websphere.appserver.zosLocalAdapters-1.0</symbolicName>
  <displayName>z/OS WebSphere Optimized Local Adapter Channel</displayName>
```

Note: That was simply a validation of the feature install. It wasn't strictly required.

- Look in the `/INSTALL_DIR/bin` directory for the sub-directory `/zosConnect`. That sub-directory was not present before installing the feature. Change to that directory.
- Enter the following two commands to set the 'execute' bits on two scripts¹⁶ in that directory:
`chmod ugo+rx bbgls2js`
`chmod ugo+rx bbgjs2ls`

¹⁴ This requires the system to have internet access.

¹⁵ You may see an indication of a failure validating `zosNoConsoleExtensions-1.0.mf`. Ignore that for this exercise.

¹⁶ These scripts are used to create the 'bind' files for data transformation.

- ❑ Verify the `chmod` by entering `ls -al` and looking at the bits:

```
-rwxr-xr-x  1 <ID> <group> 432 Jun 14 16:25 bbgjs2ls
-rwxr-xr-x  1 <ID> <group> 432 Jun 14 16:25 bbgls2js
```

- ❑ Re-mount the `INSTALL_DIR` file system so it is READ ONLY.
- ❑ Log out of the OMVS or Telnet session you were in.

Create server

In this section we will have you create and start a Liberty Profile server in which z/OS Connect will be configured and operated.

Do the following:

- ❑ Open an OMVS, Telnet or SSH session to your system and log on with any ID of your choosing.

Note: We'll start in the UNIX shell, and later go to z/OS started tasks. For this section any ID will work. When we move to using z/OS started tasks we'll have you `chown` the created directories and files to the ID you determined on page 8.

- ❑ Change directories to the `/INSTALL_DIR/bin` directory.
- ❑ Enter the following command:
`export JAVA_HOME=<Path_to_Java>`
 where `<Path_to_Java>` is the path to the 64-bit installation on your system.
- ❑ Verify that by entering the command `echo $JAVA_HOME`. It should return the value you just entered.
- ❑ Enter the following command:
`export WLP_USER_DIR=<User_Directory>`
 where `<User_Directory>` is the directory you chose as the location where the Liberty Profile server would be created.
- ❑ Verify that by entering the command `echo $WLP_USER_DIR`. It should return the value you just entered.
- ❑ Enter the following command to create the server:

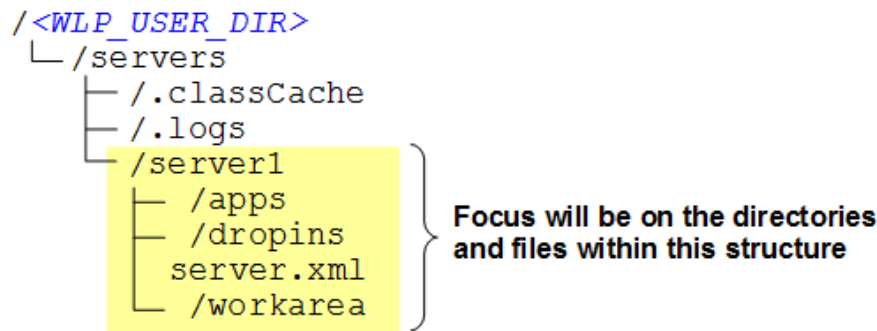
```
./server create server1
```

You should see the following response:

```
Server server1 created.
```

Note: The server configuration was created under the directory specified by the UNIX `WLP_USER_DIR` environment variable. That was the reason for setting that variable.

- Change directories to `WLP_USER_DIR` (whatever you set that value to be). You should see a structure something like this:



- Edit the file `server.xml`, which is in ASCII¹⁷. Add `host="*"` to the file as shown here:

```

000008      <!-- To access this server from a remote
000009      <httpEndpoint id="defaultHttpEndpoint"
000010                  host="*" ←
000011                  httpPort="9080"
000012                  httpsPort="9443" />

```

Change the port values as well if you think the default values will conflict. Save the file.

- From the `/INSTALL_DIR/bin` directory, enter the following command:

```
./server start server1
```

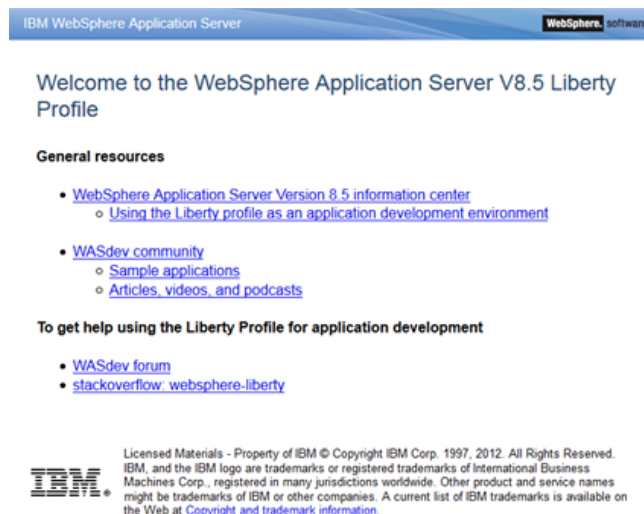
You should get an indication of the server start with process ID:

```
Server server1 started with process ID 50595976.
```

- From a browser, enter the following URL:

`http://<host>:9080/`

where `<host>` is the host on which the Liberty Profile server is running. You should get a page that looks like this:



¹⁷ However, it is "tagged" as ASCII and z/OS tools such as OEDIT should recognize it and auto-convert if your UNIX shell environment has `_BPXX_AUTOCVT=ON` set. See Liberty Profile Quick Start Guide for more – ibm.com/support/techdocs, search for WP102110.

- ❑ From the `/INSTALL_DIR/bin` directory, enter the following command:

```
./server stop server1
```

You should get an indication the server has been stopped.

The basics are now in place, but you haven't really *exercised* the z/OS Connect feature or the z/OS Local Adapters feature. That's coming up.

Setup Liberty Profile server and Angel process to run as a z/OS started tasks

Note: It is **not** a requirement of z/OS Connect that the Liberty Profile server in which it runs be a z/OS started task. However, you will require an Angel Process¹⁸ to use the Local Adapters support¹⁹, and *that* must run as a started task. It is expected most will operate z/OS Connect as a started task. Therefore, we will take you through the process of running the server as a started task.

Create the IDs and Group

Perform these steps if you wish to create new IDs and a new group. If you are re-using existing IDs and group, then skip to “Copy JCL start procedures to proclib and modify” on page 13.

Do the following:

- ❑ Work with your security administrator and create the following group and IDs:

```
ADDGROUP lib_group OMVS (GID (gid)) OWNER (SYS1)

ADDUSER angel_id DFLTGRP (lib_group) OMVS (UID (uid))
      HOME (angel_home) PROGRAM (/bin/sh) NAME ('Liberty Angel')
      OWNER (lib_group) NOPASSWORD NOOIDCARD

ADDUSER liberty_id DFLTGRP (lib_group) OMVS (UID (uid))
      HOME (server_home) PROGRAM (/bin/sh) NAME ('Liberty Server')
      OWNER (lib_group) NOPASSWORD NOOIDCARD
```

Where the highlighted values are substituted with the proper values.

Change Liberty Profile configuration directory and file ownerships

Earlier you created a Liberty Profile server from the UNIX shell using some ID you logged on with. Now is the time to change the ownership of the created directories and files to the ID you wish the Liberty Profile server to run under when operating as a started task.

Do the following:

- ❑ Log into the UNIX shell with an ID that has the authority to issue the `chown` command.
- ❑ Change directories to your `WLP_USER_DIR` directory.
- ❑ Issue the following command:

```
chown -R liberty_id:lib_group servers
```

where `liberty_ID` and `lib_group` are the ID and group values you decided upon back on page 8, and created in the previous section if you are going with new IDs and groups for this exercise.

- ❑ Verify the owner and group has been changed for the directory `servers` and the directories and files under it.

¹⁸ The Angel Process is a started task that provides an anchor point for access to z/OS authorized services, such as the local adapters support. It is *not* a server, per se: it has no TCP ports, it has no configuration file, and it uses virtually no CPU once started.

¹⁹ See “Angel process and the new WOLA support” on page 81 for more on the Angel and the new WOLA support.

Copy JCL start procedures to proclib and modify

Do the following:

- ☐ Locate the `bbgzsrv.jcl` file, which is in the following directory:

`/INSTALL_DIR/templates/zos/procs`

- ☐ Copy that file to your procedure library, for example from ISPF Option 6:

`OGET '/INSTALL_DIR/templates/zos/procs/bbgzsrv.jcl' 'SYS1.PROCLIB(BBGZSRV)'`

- ☐ Edit the JCL and change the following two lines as instructed:

```
000014 //*-----
000015 //  SET INSTDIR='/u/MSTONE1/wlp'      1
000016 //  SET USERDIR='/u/MSTONE1/wlp/usr'  2
000017 //*-----
```

Notes:

1. Set `INSTDIR` to your `INSTALL_DIR` location. Be sure to enclose the path in single quotes as shown above.
2. Set `USERDIR` to your `WLP_USER_DIR` location. Be sure to enclose the path in single quotes as shown above.

- ☐ Save the file

- ☐ Locate the `bbgzangl.jcl` file, which is in the following directory:

`/INSTALL_DIR/templates/zos/procs`

- ☐ Copy that file to your procedure library, for example from ISPF Option 6:

`OGET '/INSTALL_DIR/templates/zos/procs/bbgzangl.jcl' 'SYS1.PROCLIB(BBGZANGL)'`

- ☐ Edit the JCL and change the following one line as instructed:

```
000002 //*-----
000003 //  SET ROOT='/u/MSTONE1/wlp'      1
000004 //*-----
```

Notes:

1. Set `ROOT` to your `INSTALL_DIR` location. Be sure to enclose the path in single quotes as shown above.

- ☐ Save the file

Create `server.env` and provide `JAVA_HOME` variable to environment

When you started the server from the UNIX shell, it had knowledge of the `JAVA_HOME` because you set the UNIX environment variable. When run as a started task, you need to provide the `JAVA_HOME` value in a different way – with a `server.env` file.

Do the following:

- ☐ Create the following file:

`/WLP_USER_DIR/servers/server1/server.env`

Note: this is the same directory where the `server.xml` file is located.

- ☐ Edit that file²⁰ and enter one line:

`JAVA_HOME=<Path_to_Java>`

Note: this is the same location you captured earlier on page 8.

- ☐ Save the file.

- ☐ Change the ownership of the file to the Liberty Profile ID and group.

²⁰ For a started task, the content should be in EBCDIC.

Create SAF STARTED profiles for Angel process and Server

To get the IDs you planned earlier (page 8) to be assigned to these started tasks, you will need to create two SAF STARTED profiles²¹.

Do the following:

- ☐ Work with your security administrator and issue the following commands²²:

```
RDEF STARTED BBGZANGL.* UACC(NONE) STDATA(USER(angel_id)
                                GROUP(lib_group) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
RDEF STARTED BBGZSRV.* UACC(NONE) STDATA(USER(liberty_id)
                                GROUP(lib_group) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
SETROPTS RACLIST(STARTED) REFRESH
```

Start the Angel process as a started task

The Angel process is very simple and lightweight – it has no configuration, no TCP ports, and it consumes almost no CPU once started. Do the following:

- ☐ From the z/OS operator console, issue the following command:

S BBGZANGL

You should see the Angel come active very quickly:

```
S BBGZANGL
IRR812I PROFILE BBGZANGL.* (G) IN THE STARTED CLASS WAS USED
                                TO START BBGZANGL WITH JOBNAME BBGZANGL.
$HASP100 BBGZANGL ON STCINRDR
IEF695I START BBGZANGL WITH JOBNAME BBGZANGL IS ASSIGNED TO
                                USER angel_id, GROUP lib_group
$HASP373 BBGZANGL STARTED
CWWKB0056I INITIALIZATION COMPLETE FOR ANGEL
```

Start server as a started task

You're ready to run the server as a started task – you've created the JCL proc, created a STARTED profile, and created a `server.env` with `JAVA_HOME`.

Do the following:

- ☐ Open a 'System Command Extension' window²³.
- ☐ Enter the following command:

START BBGZSRV,PARMS='server1'

You should see the server come active:

```
JOBNAME  StepName ProcStep JobID
BBGZSRV  BBGZSRV  STEP1      STC04122
```

- ☐ Go back and edit the `server.xml` file and *add* the following two lines:

```
<feature>zosConnect-1.0</feature>
<feature>zosLocalAdapters-1.0</feature>
```

at the location shown here:

²¹ Early in this document we have you create the SAF profiles as you need them. If you would prefer to see the complete set of profiles needed and create them ahead of time, see "SAF definitions" on page 76.

²² Here we are showing IBM RACF commands. If you have another vendor's security product, use the commands appropriate for that vendor's product.

²³ A single slash will open the command extension. Why is this needed? Because by default the JCL will start a server with name `defaultServer`. Your server name is `server1`. To specify your server, you provide `PARMS=` on the `START` command. However, your server name is *lowercase*, and you must preserve the case on the `START` command. The command extension window will preserve the case. A simple `/START` from the operator console will not.

```

000003    <!-- Enable features -->
000004    <featureManager>
000005        <feature>jsp-2.2</feature>
000006        <feature>zosConnect-1.0</feature>
000007        <feature>zosLocalAdapters-1.0</feature>
000008    </featureManager>

```

Save the file.

- Because Liberty Profile is dynamic²⁴, the changes you made to the `server.xml` are automatically detected and implemented. Look at the log file²⁵:

`/WLP_USER_DIR/servers/server1/logs/messages.log`

In particular, look for the following messages which will be near the bottom:

```

:
J2CA7018I: Installing resource adapter ola.
J2CA7001I: Resource adapter ola installed in 0.496 seconds.
:
SRVE0169I: Loading Web Module: z/OS Connect.
SRVE0250I: Web Module z/OS Connect has been bound to default_host.

```

The first two messages (J2CA*) indicate the Optimized Local Adapter JCA resource adapter has been installed. That's a preliminary validation of the local adapters feature being operative. We'll validate it more fully under "Section 2: Enabling WOLA" starting on page 16.

The second two messages (SRVE*) indicate the z/OS Connect servlet web module is being loaded and bound. That's a preliminary validation of the z/OS Connect feature. We'll validate it more fully under "Section 3: Exploring z/OS Connect Functionality" starting on page 29.

- Enter the MVS command:

P BBGZSRV

The started task should stop.

- Edit the `server.xml` file and **remove** the `zosConnect-1.0` line:

```

000004    <featureManager>
000005        <feature>jsp-2.2</feature>
000006        <feature>zosConnect-1.0</feature>
000007        <feature>zosLocalAdapters-1.0</feature>
000008    </featureManager>

```

We'll put that back later. For now taking that out eliminates the need for SSL, which is a small distraction during the WOLA validation.

Reserve the TCP ports

Work with your network administrator to reserve your server's TCP ports for its use only²⁶. If later the ports are in use by something else, your Liberty Profile server will not start.

Status at this point

You have constructed the essential framework to begin testing and validating z/OS Connect.

You have not yet invoked any of the services in z/OS Connect. Your instance of Liberty Profile z/OS has the z/OS Connect servlet loaded, but there is nothing in the `server.xml` (yet) to tell it what to do. We'll do that under "Section 3: Exploring z/OS Connect Functionality" starting on page 29.

But first we need to set up a bit of infrastructure support – WOLA.

²⁴ Not all changes are dynamic, but many are.

²⁵ The `messages.log` file is in ASCII.

²⁶ Update TCPPARMS and SAF.

Section 2: Enabling WOLA

z/OS Connect uses the WebSphere Optimized Local Adapters – or WOLA²⁷ – to access backend systems such as CICS and batch programs. Initially we'll use COBOL batch because it's relatively simple, then on page 24 we'll turn to using CICS.

Enable WOLA

You did some of this earlier when you used the `featureManager` shell script to install the z/OS Connect and local adapters function (starting on page 8). You have the Angel process running (page 14). What's left to do is update the `server.xml` with a few things, add some SAF profiles and perform some validation.

Configuration updates to support WOLA

Do the following:

- ❑ Verify you have the `zosLocalAdapters-1.0` feature configured in your `server.xml`. Your copy of `server.xml` should look like this right now:

```
000001 <server description="new server">
000002
000003     <!-- Enable features -->
000004     <featureManager>
000005         <feature>jsp-2.2</feature>
000006         <feature>zosLocalAdapters-1.0</feature>
000007     </featureManager>
000008
000009     <!-- To access this server from a remote client
000010     <httpEndpoint id="defaultHttpEndpoint"
000011                 host="*"
000012                 httpPort="9080"
000013                 httpsPort="9443" />
000014
000015 </server>
```

- ❑ Add the following (see notes that follow):

```
000001 <server description="new server">
000002
000003     <!-- Enable features -->
000004     <featureManager>
000005         <feature>jsp-2.2</feature>
000006         <feature>zosLocalAdapters-1.0</feature>
000007     </featureManager>
000008
000009     <zosLocalAdapters
000010         wolaGroup="GROUP"
000011         wolaName2="NAME2"
000012         wolaName3="NAME3" />
000013
000014     <connectionFactory id="wolaCF" jndiName="eis/ola">
000015         <properties.ola />
000016     </connectionFactory>
000017
000018     <!-- To access this server from a remote client
000019     <httpEndpoint id="defaultHttpEndpoint"
000020                 host="*"
000021                 httpPort="9080"
000022                 httpsPort="9443" />
000023
000024 </server>
```

Notes:

- WOLA requires a three-part name to be used when external address spaces (CICS regions, batch programs) register into the Liberty Profile server. In full-function WAS z/OS that three part name is the cell, node and server short names. Liberty Profile has no cell or node constructs. To keep the WOLA programming model consistent

²⁷ For more on WOLA in general, see WP101490 at ibm.com/support/techdocs.

between full-function WAS z/OS and Liberty Profile z/OS the three-part name model was carried into Liberty Profile. The three-part name is accomplished with the XML you see in the diagram.

Note: The *values* are arbitrary. Each name component is limited to 8 characters alpha-numeric with no blank spaces. The names are case sensitive, so if you have mixed case in the `server.xml` then the outside program seeking to register into the Liberty Server would need to match the case.

Recommendation: make the values upper-case. It makes things easier when you create the `CBIND` profile (see page 21 for that).

The three-part name used by a Liberty Profile z/OS server instance *must be unique* on the z/OS LPAR.

- The `connectionFactory` XML defines the WOLA JCA resource adapter that is used for communication outbound from Liberty Profile to the external address space.

☐ Save the file.

SAF SERVER updates to support WOLA

The WOLA support requires the Angel process. But having the Angel running is not enough. You must also create a set of SAF `SERVER` profiles to grant your Liberty Profile server authority to use the services of the Angel²⁸.

Do the following:

- ☐ Work with your security administrator to see if the following SAF `SERVER` profiles already exist, and if not create them. Use the following commands:
 - ☐ `RDEF SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)`
 - ☐ `RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)`
 - ☐ `RDEF SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)`
 - ☐ `RDEF SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)`
 - ☐ `RDEF SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)`
 - ☐ `RDEF SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)`
 - ☐ `SETROPTS RACLIST(SERVER) REFRESH`

Note: If you've worked with Liberty Profile z/OS in the past, the first two may already exist²⁹. You do not need to recreate them. The last four are new to the WOLA support introduced in 8.5.5.2. Those will very likely need to be created.

- ☐ When you have all six of `SERVER` profiles verified or created, then grant your Liberty Profile server ID³⁰ `READ` access to each. Issue the following commands:
 - ☐ `PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(liberty_id)`
 - ☐ `PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(liberty_id)`
 - ☐ `PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS(SERVER) ACCESS(READ) ID(liberty_id)`
 - ☐ `PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS(SERVER) ACCESS(READ) ID(liberty_id)`
 - ☐ `PERMIT BBG.AUTHMOD.BBGZSCFM CLASS(SERVER) ACCESS(READ) ID(liberty_id)`
 - ☐ `PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS(SERVER) ACCESS(READ) ID(liberty_id)`
 - ☐ `SETROPTS RACLIST(SERVER) REFRESH`

²⁸ See "SAF definitions" on page 76 for a complete list of all the SAF profiles.

²⁹ And in fact you may have four other `AUTHMOD` profiles: `SAFCRED`, `TXRRS`, `ZOSDUMP`, `ZOSWLM`. Those were part of the original Liberty Profile z/OS support introduced with 8.5.0.0.

³⁰ See page 8 for the value you planned; see page 14 for the SAF `STARTED` profile that assigns the ID to the started task.

Start server and check for key validation messages in log

Do the following:

- ❑ From *the command extension*, issue the START command for the server:

```
S BBGZSRV,PARMS='server1'
```

- ❑ Go the `/WLP_USER_DIR/servers/server1/logs` directory and look at the contents of the `messages.log` file. Specifically, look for the following WOLA success indicators:

```
TRAS0018I: The trace state has been changed. The new trace state is *=info.
CWWKE0001I: The server server1 has been launched.
CWWKB0103I: Authorized service group LOCALCOM is available. 1
CWWKB0103I: Authorized service group WOLA is available. 2
CWWKB0104I: Authorized service group PRODMGR is not available. 3
CWWKB0104I: Authorized service group SAFCRE is not available.
CWWKB0104I: Authorized service group TXRRS is not available.
CWWKB0104I: Authorized service group ZOSDUMP is not available.
CWWKB0104I: Authorized service group ZOSWLM is not available.
:
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with
the Liberty profile server using the following name: GROUP NAME2 NAME3 4
:
J2CA7001I: Resource adapter ola installed in 0.327 seconds. 5
```

Notes:

1. The `LOCALCOM` service group being available is a key indicator that the Angel is running *and* your Liberty Server ID has `READ` to that `SERVER` profile.
2. The `WOLA` service group being available is another key indicator that the setup work was successfully completed.
3. The next file services groups being *not available* is okay for now³¹.
4. This message is showing the successful use of the WOLA three-part name values you provided in the `server.xml` file. This message indicates that an outside address space (we'll show using COBOL batch in a little bit) may register into the Liberty Profile server using this three-part name on the `BBOA1REG` API.
5. The WOLA JCA resource adapter is available.

The key messages we wanted to see here were 1, 2 and 4.

Sample Java application to validate WOLA

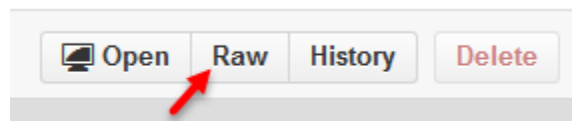
For *preliminary* validation of WOLA – that is, before we get z/OS Connect up and going – we're going to use a sample Java application provided with the Liberty Profile z/OS WOLA support.

Do the following:

- ❑ Go to the following URL:

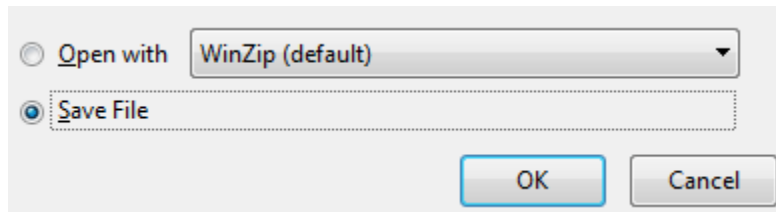
<https://github.com/WASdev/sample.wola/blob/master/OLASampleLiberty.ear>

- ❑ Click on the “raw” tab:



31 They're not available because the Liberty Profile server ID has not been granted `READ` to the respective `SERVER` profiles. If you see some of these being available, that's likely due to previous Liberty Profile z/OS work on this LPAR being done with those SAF profiles being set up with your server ID being granted `READ`.

- When prompted, choose to save the file:



- FTP the file *in binary* up to your system and place it in the `/dropins` directory³² with permissions that provides the Liberty ID at least READ to the file³³:

```

/<WLP_USER_DIR>
└─ /servers
   └─ /.classCache
   └─ /.logs
   └─ /server1
      └─ /apps
      └─ /dropins
      └─ server.xml
      └─ /workarea

```

The `/dropins` directory is located just under `/server1` directory

- Now look in the `messages.log` file, specifically for the messages indicating the application is being started, and it has completed starting:

```

CWWKZ0018I: Starting application OLASampleLiberty.
:
CWWKZ0001I: Application OLASampleLiberty started in 0.104 seconds.

```

Use sample COBOL batch to register into Liberty using WOLA

The Liberty Profile WOLA support is up and ready to go. Now all you need is something to talk to over WOLA. In this section we'll illustrate using a sample COBOL batch program. We start with that because it's relatively simple to set up and use. Later, on page 24, we'll show how to set up and use z/OS Connect with CICS.

Copy WOLA modules from file system to load library data set

The WOLA modules come with Liberty Profile z/OS 8.5.5.2. But for use by external address spaces such as batch programs or CICS regions those modules must be copied out to a load library. The load library may then be pointed to with `STEPLIB` or `DFHRPL`.

Do the following:

- Allocate a PDSE³⁴ with 30 tracks:

```

Average record unit
Primary quantity . . 30
Secondary quantity 2
Directory blocks . . 15
Record format . . . U
Record length . . . 0
Block size . . . . 32760
Data set name type LIBRARY

```

- Open a Telnet session to your system and change directories to the following location:

³² The `/dropins` directory is created when the server is started for the first time. Liberty Profile monitors the `/dropins` directory and will dynamically load any application files placed in the directory.

³³ Liberty Profile will throw an FFDC event if it doesn't have at least READ to the application file.

³⁴ A good practice is to provide the fixpack level of the modules in the name qualifier, for example: `LIBERTY.V855FP02.LOADLIB`. That helps reduce questions about what level of WOLA code is resident in the PDSE.

/INSTALL_DIR/clients/zos

There you will find the WOLA modules:

```
USER:/INSTALL_DIR/clients/zos-> ls
bboalcng  bboalinv  bboalrcs  bboalsrq  bboalurg  bboacsrsv
bboalcnr  bboalrca  bboalreg  bboalsrv  bboaclnk  bboatrue
bboalget  bboalrcl  bboalsrp  bboalsrx  bboacntl
```

- From the Telnet session, issue the following command to copy the modules from the file system to the PDSE you just allocated:

```
cp -Xv ./* "'/'<data.set>' "
```

Where **<data.set>** is the target PDSE. For example:

```
cp -Xv ./* "'/'LIBERTY.V855FP02.LOADLIB' "
```

You should see something like this:

```
USER:/INSTALL_DIR/clients/zos-> cp -Xv ./* "'/'LIBERTY.V855FP02.LOADLIB' "
./bboalcng -> //'LIBERTY.V855FP02.LOADLIB(bboalcng)': executable
./bboalcnr -> //'LIBERTY.V855FP02.LOADLIB(bboalcnr)': executable
./bboalget -> //'LIBERTY.V855FP02.LOADLIB(bboalget)': executable
./bboalinv -> //'LIBERTY.V855FP02.LOADLIB(bboalinv)': executable
./bboalrca -> //'LIBERTY.V855FP02.LOADLIB(bboalrca)': executable
./bboalrcl -> //'LIBERTY.V855FP02.LOADLIB(bboalrcl)': executable
./bboalrcs -> //'LIBERTY.V855FP02.LOADLIB(bboalrcs)': executable
./bboalreg -> //'LIBERTY.V855FP02.LOADLIB(bboalreg)': executable
./bboalsrp -> //'LIBERTY.V855FP02.LOADLIB(bboalsrp)': executable
./bboalsrq -> //'LIBERTY.V855FP02.LOADLIB(bboalsrq)': executable
./bboalsrv -> //'LIBERTY.V855FP02.LOADLIB(bboalsrv)': executable
./bboalsrx -> //'LIBERTY.V855FP02.LOADLIB(bboalsrx)': executable
./bboalurg -> //'LIBERTY.V855FP02.LOADLIB(bboalurg)': executable
./bboaclnk -> //'LIBERTY.V855FP02.LOADLIB(bboaclnk)': executable
./bboacntl -> //'LIBERTY.V855FP02.LOADLIB(bboacntl)': executable
./bboacsrsv -> //'LIBERTY.V855FP02.LOADLIB(bboacsrsv)': executable
./bboatrue -> //'LIBERTY.V855FP02.LOADLIB(bboatrue)': executable
```

And if you look in the PDSE, you should see:

LIBERTY.V855FP02.LOADLIB									
Row 00001 of 00017									
Scroll ==> CSR									
Command	Name	Prompt	Alias-of	Size	TTR	AC	AM	RM	
	BBOACLNK			000122A4	000011	00	31	ANY	
	BBOACNTL			0000EF4C	000012	00	31	ANY	
	BBOACSRV			0000A29C	000013	00	31	ANY	
	BBOATRUE			00005078	000014	00	31	ANY	
	BBOAICNG			00000080	000004	01	31	ANY	
	BBOAICNR			00000080	000005	01	31	ANY	
	BBOAIGET			00000080	000006	01	31	ANY	
	BBOAINV			00000080	000007	01	31	ANY	
	BBOAIRCA			00000080	000008	01	31	ANY	
	BBOAIRCL			00000080	000009	01	31	ANY	
	BBOAIRCS			00000080	00000A	01	31	ANY	
	BBOAIREG			00000080	00000B	01	31	ANY	
	BBOAISRP			00000080	00000C	01	31	ANY	
	BBOAISRQ			00000080	00000D	01	31	ANY	
	BBOAISRV			00000080	00000E	01	31	ANY	
	BBOAISRX			00000080	00000F	01	31	ANY	
	BBOAURG			00000080	000010	01	31	ANY	

Compile sample COBOL batch

Do the following:

- ☐ From the same ZIP file you downloaded from the WP102439 Techdoc, extract the `ZOSCONN.txt` file. That contains a COBOL source deck³⁵ for a WOLA program that “hosts a service” – that is, will accept a call from the Java program in Liberty³⁶.
- ☐ Upload that `ZOSCONN.txt` file to your system. Whether you put that in a PDS or a sequential data set is a matter of your standard COBOL compilation procedures.
- ☐ Edit the uploaded `ZOSCONN` file and make the following changes:

```

MAINLINE SECTION.
  MOVE 'ZOSCONN'                TO register-name.
  MOVE 'GROUP'                  TO daemongroup.
  MOVE 'NAME2'                  TO node-name.
  MOVE 'NAME3'                  TO server-name.
  MOVE 'ServiceName'           TO SRV-service-name.
  MOVE '{"BestPlatformEver":"z/OS"}' TO reply-message.

```

That must match³⁷ the updates you made to the `server.xml` back on page 16. Your Liberty Profile server is known to WOLA by its three-part name; this COBOL program will now register into the server using the same three-part name.

- ☐ Compile that COBOL source.

Create SAF CBIND profile

Another SAF profile is needed, this one to control what external IDs may use WOLA to register into the Liberty Profile server.

Do the following:

- ☐ Work with your security administrator to create the following profile:

```
RDEF CBIND BBG.WOLA.group.name2.name3 UACC(NONE) OWNER(SYS1)
```

where `group`, `name2` and `name3` match what was used in the `server.xml` (page 16) to create the three-part name for WOLA.

Notes: You can wildcard the profile to make it more generic. For example, the `CBIND` profile `BBG.WOLA.*` would apply to any three-part name you use in Liberty.

`CBIND` profiles are stored in *upper-case*. If your three-part name contains lowercase, there will be a mismatch when checking `CBIND` authority. You can avoid problems by using upper-case values in your `server.xml`. When the three part name is all upper-case you can be sure it'll match an uppercase `CBIND`. Otherwise, you can work around lower-case values by using wildcard values in the `CBIND` profile.

- ☐ Then grant the external address space ID (the batch program in this example) `READ` access to the `CBIND` profile:

```
PERMIT BBG.WOLA.group.name2.name3 CLASS(CBIND) ACCESS(READ) ID(accessing_id)
```

- ☐ And then refresh:

```
SETROPTS RACLIST(CBIND) REFRESH
```

³⁵ That sample COBOL is based on a program from WP101490 Techdoc at ibm.com/support/techdocs. A “COBOL Primer” document shows how to use the WOLA APIs, and a set of sample COBOL files is provided. `ZOSCONN` is based on `EXER3B` from that Techdoc.

³⁶ A little more detail: `ZOSCONN` will first register in with `BBOA1REG`. Then it calls `BBOA1SRV`, which is what “hosts a service.” When a message is sent from Java to COBOL over WOLA, `BBOA1SRV` “wakes up” and returns program control to COBOL. The program then sends back a hard-coded response, then loops and calls `BBOA1SRV` again. It'll do that forever until it sees uppercase `STOP` sent from the Java program. Then it branches and calls `BBOA1URG` to unregister, and the program ends.

³⁷ Including case. We recommend your WOLA three-part name be in uppercase in the `server.xml`, and matched with uppercase in the calling program. This will also help match the `CBIND` profile.

Run COBOL batch and validating using sample Java program

Earlier you compiled the `ZOSCONN` program. Now it is time to run that program so it will register into the Liberty Profile z/OS server.

Do the following:

- Using whatever JCL you normally use and run the batch job. When operating properly it should register into the Liberty Profile server and then wait. The batch job should stay up and active. If it ends relatively quickly then there's a problem. Look in the output of the batch job:

Good sign:

The batch program output shows:

Successfully registered into GROUP

and then it sits and waits for the Java program to call.

Bad signs:

Batch program output	Cause
SEC3 abend with reason code=20000100	A Liberty Profile Angel process is in effect operating with code lower than 8.5.5.2 level. See note below.
BBOA1REG problem, rc/rsn: 00000012/00000016 EXITING program due to non-RC=0. <i>The very first API call – BBOA1REG – failed.</i>	The Liberty Profile server is not up and running, or there's a mismatch in the three-part name used by the calling program compared to what the Liberty Profile server is hosting.
Successfully registered into GROUP <i>The very first API call – BBOA1REG – worked, but ...</i> BBOA1SRV problem, rc/rsn: 00000012/00000014 <i>The BBOA1SRV call had a problem.</i> EXITING program due to non-RC=0.	This is a problem with the CBIND profile and the authority of the calling ID to access the profile. See page 21.

Note: If you experience the SEC3 abend with RC=20000100, the likely cause is you have z/OS 2.1 with zOSMF running. That is based on Liberty Profile and has its own Angel process. To get around this, stop zOSMF and its Angel. Start your copy of the Angel at 8.5.5.2 or higher. Restart zOSMF, which will use your Angel. Rerun the batch job to validate WOLA.

Important! zOSMF is now dependent on your Angel process. Do not cancel or stop the Angel.

- Earlier you deployed the `OLASampleLiberty.ear` application into the `/dropins` directory of your Liberty Profile server (page 19). From a browser, enter the following URL to get the initial page of the sample application:

`http://<host>:9080/OLASampleLibertyWeb/`

You should see a rather long and complex-looking page like this³⁸:

³⁸ We only care about a few of the fields. And this is being used for validation of WOLA only. We won't use this with z/OS Connect.

- Enter the Registration name and service name into those fields (about 1/3rd the way down the web page):

Case matters for both. Enter as shown here.

- Scroll to the bottom and click the “Run” button:

- If it works, you should get the following on your browser screen:

That output string is JSON that is hard-coded in the sample COBOL³⁹.

³⁹ That's useful for preliminary testing because it lets us avoid doing data conversion initially. We'll use an HTTP client to send a PUT with no JSON to z/OS Connect. It'll pass that through to the COBOL batch. But when no data conversion is in place then z/OS Connect

- ☐ Use the browser back button to return to the input screen.
- ☐ In the very top input field, enter the string **STOP** in *uppercase* as shown⁴⁰:

Data to send to external address space (Part 1): ☐ Convert to ebcdic
 STOP

- ☐ Click the “Run” button again. You should see:

Output: 000002 Okay I will stop!
 Test Executed

- ☐ The batch job should end, and if you look in the output of that now-ended job you should see the following:

```
Successfully registered into GROUP
Your message in original ASCII:
Your message converted to EBCDIC:
My reply in EBCDIC: {"BestPlatformEver":"z/OS"}
My reply converted to ASCII: #      &%/  ?  _      : | '
Your message in original ASCII:      / /  _?      >/% /      /
Your message converted to EBCDIC: Test data to external a/s part 1
My reply in EBCDIC: {"BestPlatformEver":"z/OS"}
My reply converted to ASCII: #      &%/  ?  _      : | '
Your message in original ASCII:  |&
Your message converted to EBCDIC: STOP
My reply in EBCDIC: Okay I will stop!
My reply converted to ASCII: |,/`  %%  ?
Successfully unregistered from GROUP
```

- ☐ Stop the Liberty Profile server. Leave the Angel process running.

Enable WOLA support in a CICS region

The WOLA support with Liberty Profile works with CICS regions as well. The mechanism z/OS Connect uses to communicate with CICS is WOLA, so showing how that works is important.

Note: The instructions given here are the *basics* to enable WOLA support in CICS and establish communications between the Liberty Profile server and the CICS region.

Beyond the basics there's a range of CICS system programmer activities for things such as enabling the WOLA Task Related User Exit (TRUE) to start at region startup, and configuring the Link Server Task to start using either INITPARM or sequential terminal. Those tasks are really CICS system programmer tasks, and the steps for Liberty Profile z/OS WOLA are pretty much the same as for full-function WAS z/OS WOLA.

For more on how to perform those CICS system programmer tasks, see the [WP101490](#) Techdoc at ibm.com/support/techdocs. The “WOLA Quick Start Guide” provides those step-by-step instructions.

Do the following:

- ☐ Identify the CICS region you wish to use for this exercise. Capture the following:

Region Name:	
ID of started task ⁴¹	

expects JSON to come back from backend program. So we hard-coded some JSON to satisfy that requirement. Once we get the basics of z/OS Connect validated, then we'll move on to data conversion.

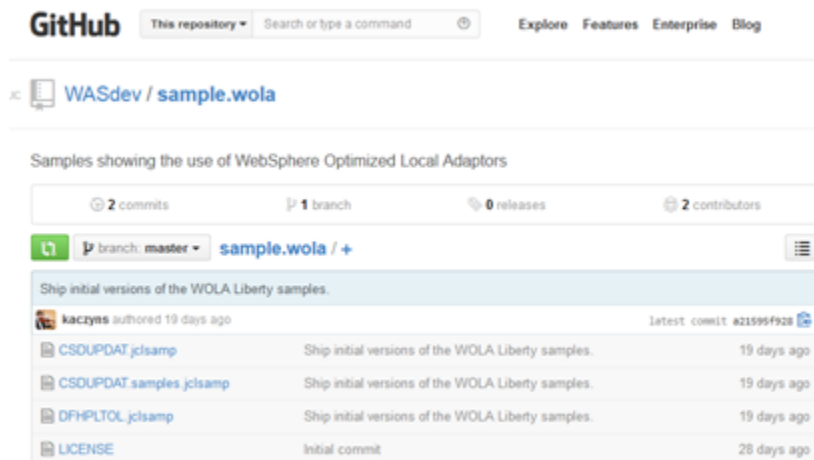
40 The COBOL batch program will keep looping back and calling BBOA1SRV until it receives the string **STOP**. Then it will branch out of the loop and call BBOA1URG and exit.

41 This ID will be granted **READ** to the WOLA **CBIND** profile, which will allow the region to register into the Liberty Profile server using WOLA.

- ❑ Go to the following website

<https://github.com/WASdev/sample.wola>

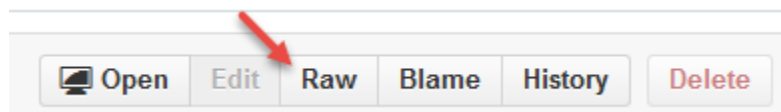
That is where the sample JCL jobs are provided that enable the WOLA support into the CICS region, along with sample WOLA programs. You should see something like this:



- ❑ Locate and click on the first file in the list – CSDUPDAT.jclsamp. That will open up the file and it will look like this:



- ❑ Click on the “Raw” button at the top:



That removes the line numbers on the left and makes a “select all / copy” much easier.

- ❑ Enter **Ctrl-A** to select all the contents, then **Ctrl-C** to copy to the clipboard (or whatever mechanism on your workstation performs that function).
- ❑ Create a file on your workstation with name **CSDUPDAT.jclsamp** and paste the contents. Save the file.
- ❑ Do the same for the following three files:
 - CSDUPDAT.jclsamp JCL to update the CICS CSD with the WOLA infrastructure components such as the Task Related User Exit and the Link Server Task.
 - CSDUPDAT.samples.jclsamp JCL to update the CICS CSD with the WOLA samples. The only sample we'll be using is OLACB01.⁴²
 - OLACB01.jclsamp JCL and source for the OLACB01 source program, which is a simple COMMAREA echo program with a single field.

⁴² The WOLA support for full-function WAS z/OS provides the CSD updates in one sample file. For Liberty Profile z/OS WOLA the CSD updates are broken out into two files: one for the key WOLA infrastructure elements; another for the sample programs.

- ☐ Upload those files *in ASCII* to a FB 80 PDS on your system⁴³.
- ☐ Modify the CSDUPDAT member so the STEPLIB and DFHCSD DD cards accurately reflect your system's CICS installation.
- ☐ Review and modify as necessary the ADD statement at the bottom of the file so it conforms with your CICS installation standards.
- ☐ Run that job and look for RC=0.
- ☐ Modify the CSD update member for the samples and do the same⁴⁴.
- ☐ Modify the OLACB01.jclsamp member so the OLACB01 program can be compiled into a load library that will be recognized by the CICS region⁴⁵.
- ☐ Edit the JCL start procedure for the CICS region and add the WOLA module PDSE (the one you created on page 20) to the DFHRPL DD concatenation.
- ☐ Grant the CICS region ID READ access to the CBIND profile created earlier (page 21):

```
PERMIT BBG.WOLA.group.name2.name3 CLASS(CBIND) ACCESS(READ) ID(cics_id)
```
- ☐ Then refresh:

```
SETROPTS RACLIST(CBIND) REFRESH
```
- ☐ Start the CICS region. In the MSGUSR held output for the region you should see a series of messages indicating the WOLA support is added:

```
Resource definition for BBOACALL has been added.
:
TRANSACTION definition entry for BBOC has been added.
TDQUEUE entry for BBOQ has been added.
```
- ☐ Make sure the Liberty Profile server is started. The next step involves issuing a CICS command to start the link server task and register into the Liberty Profile server. Therefore, the server must be active for this to succeed.
- ☐ Open a terminal session with the CICS region.
- ☐ Issue the following command⁴⁶:

```
BBOC START_TRUE
```

You should see the following message indicating success:

```
WOLA TRACE 1: Exit enabled successfully.
```
- ☐ Issue the following command (*as one line*) to start the link server task and register into your server^{47 48}:

```
BBOC START_SRVR RGN=CICSREG DGN=GROUP NDN=NAME2
SVN=NAME3 SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y
```

Note: Where GROUP, NAME2 and NAME3 matches the WOLA three-part name you set for your Liberty Profile server back on page 16.

You should see the following message indicating success:

```
WOLA TRACE 0: Start server completed successfully.
```

⁴³ You'll need to make the PDS member name for the CSDUPDAT.samples.jclsamp something other than CSDUPDAT since the CSDUPDAT.jclsamp file will take that member name.

⁴⁴ We will only make use of the OLACB01 sample, so if you wish you may trim the file so only that update is made to the CSD. Otherwise, leave as is and have the CSD updates in place for the future if you choose to use the other samples.

⁴⁵ For example, the PDSE into which the WOLA modules were copied as shown on page 20.

⁴⁶ This can be automated using samples DFHPLTOL and BBOACPLT. See WP101490 Techdoc Quick Start Guide.

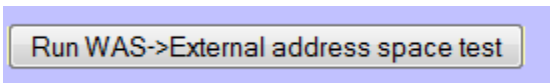
⁴⁷ RGN= may be any value you wish up to 12 characters. CICSREG is just an example of a registration name.

⁴⁸ This can be automated using INITPARM or sequential terminal. See WP101490 Techdoc Quick Start Guide.

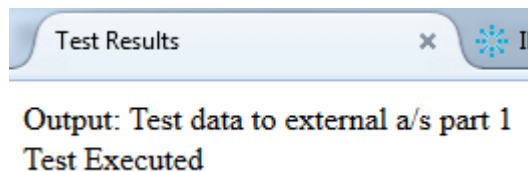
If you get a non-zero return code and reason code, then go to the Knowledge Center (see page 6) and search on the string `cdat_olaapis`. Open the link and go to the “Register” section (also known as the `BBOA1REG` section) and look for the RC and RSN code provided there.

- ☐ With a browser, go to the WOLA sample program you deployed in Liberty Profile earlier:
`http://<host>:9080/OLASampleLibertyWeb/`
- ☐ As before, enter the registration name and the service name:

- ☐ Scroll to the bottom and click the “Run” button:



- ☐ This will simply echo back the text string sent over:



That validates the ability to communicate over WOLA to CICS using the WOLA Link Server Task⁴⁹.

- ☐ From the CICS terminal, enter the command:

```
BBOC STOP_SRVR RGN=CICSREG
```

Where `CICSREG` matches the `RGN=` value supplied on the `BBOC` command you entered on page 26.

- ☐ Then issue the command:

```
BBOC STOP_TRUE
```

At this point the WOLA Link Server Task is stopped, the registration into the Liberty Profile server taken down, and the `TRUE` stopped. If you wished to use WOLA into this CICS region you would restart the `TRUE` and the Link Server Task⁵⁰.

Configuration checkpoint: server.xml at this point

See “Checkpoint 1: End of WOLA Enablement” on page 82 for a sample of the `server.xml` at this point in the flow of this document.

⁴⁹ If you look at the source for `OLACB01` you'll find it does not use any of the WOLA native APIs. `OLACB01` is completely WOLA-unaware. The Link Server Task shields it from having to know about WOLA.

⁵⁰ The `TRUE` is commonly started at CICS region initialization using the supplied PLT program. Starting the Link Server Task implies the creation of a registration, and that requires the Liberty Profile server be up ready to accept the registration. After the initial successful start of the Link Server with registration, the Link Server will retry registration if the Liberty Profile server goes down and comes back up.

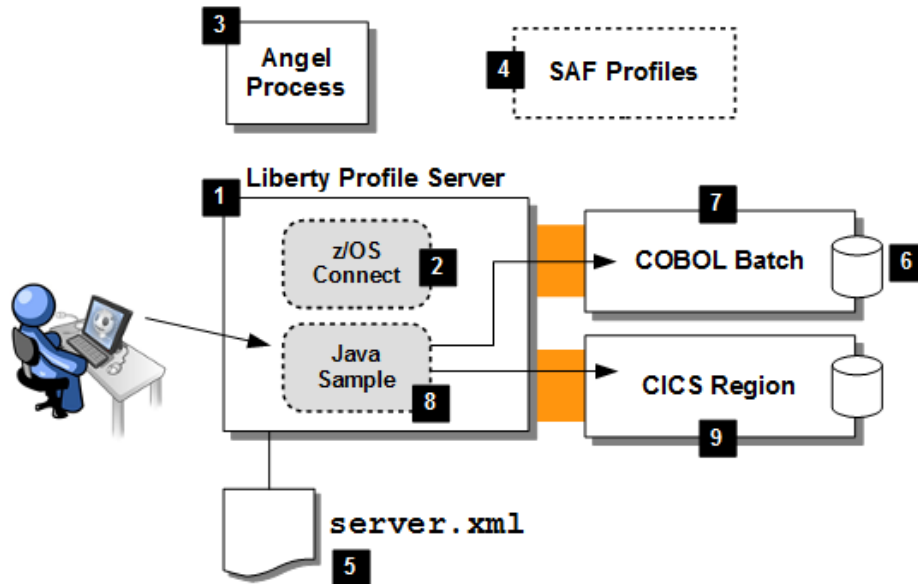
Recap of the WOLA validation

The purpose of this section was to get the WOLA infrastructure working and validated outside of any z/OS Connect considerations. WOLA is what z/OS Connect uses to talk to CICS and Batch programs⁵¹, so having this work properly a key first step.

With WOLA validated and a simple batch program known to work using WOLA, we may proceed to setting up and using z/OS Connect.

Status at this point

At this point you've accomplished a fair amount:



Numbers below correspond to the blocks in the diagram:

1. You have a Liberty Profile server running as a z/OS started task
2. The z/OS Connect function has been enabled inside the Liberty Profile server, but up to this point you have not done anything with it. It's there, just unused at the moment.
3. The Angel process is running as a z/OS started task. The Angel is needed for the WOLA support, and WOLA is an important part of z/OS Connect usage.
4. You created various SAF profiles such as `STARTED`, `SERVER` and `CBIND`.
5. You made various updates to `server.xml`, particularly in support of WOLA.
6. You copied the WOLA modules from the product file system into a pre-allocated PDSE so programs such as the sample batch can access the modules.
7. You compiled and ran a sample COBOL batch program that registers into the Liberty Profile server and "hosts a service".
8. You used a sample WOLA application in Liberty to validate WOLA and send messages over WOLA to the COBOL batch hosting a service.
9. You enabled WOLA in a CICS region and validated the connection by using the same sample Java application to send a request to the sample `OLACB01` program. The CICS region has the WOLA module PDSE concatenated to the `DFHRPL DD` card.

⁵¹ z/OS Connect and IMS uses a JCA resource adapter supplied by IMS. That is outside the scope of this document.

Section 3: Exploring z/OS Connect Functionality

The time has come to begin exploring the z/OS Connect functionality.

Level-set before we begin

Up to this point most of the work has been setup in nature – using Installation Manager to install Liberty Profile z/OS and the required iFixes, running the `featureManager` shell script to install the z/OS Connect and the Local Adapters feature, and enabling WOLA to talk to a backend program. You used a browser to validate some things, but you did not exercise the REST/JSON capabilities of z/OS Connect. Now you will.

Set up basic registry and validate access to z/OS Connect

z/OS Connect runs inside Liberty Profile as a servlet, and that servlet imposes a requirement for SSL as well as authentication. In this section we'll add XML to `server.xml` to provide basic⁵² setup for both of those requirements.

Important! To this point we have *not* instructed you to create the SAF `SERVER` class profile for `SAFCRED`. But you *may* have done it ahead of time based on what you saw under “SAF definitions” on page 76. *If* you had done that, *and* your server ID has `READ` to it, *then* you will need *two more* things in SAF to do the following exercise. See “Workaround to do basic authentication using local repository” on page 80.

If you have not yet created the SAF `SERVER` class for `SAFCRED`, or your server ID does not have `READ` to it, then proceed.

Do the following:

- ☐ Edit your `server.xml` file and **add** the lines **highlighted** here:

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
  </featureManager>

  <keyStore id="defaultKeyStore" password="Liberty"/>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <basicRegistry id="basic1" realm="zosConnect">
    <user name="Fred" password="fredpwd" />
  </basicRegistry>

  <authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
      <user name="Fred"/>
    </security-role>
  </authorization-roles>

  <zosLocalAdapters
    wolaGroup="GROUP"
    wolaName2="NAME2"
    wolaName3="NAME3" />

  <connectionFactory id="wolaCF" jndiName="eis/ola">
```

⁵² That is, no SAF or LDAP involved at this point.

```

<properties.ola />
</connectionFactory>

<!-- To access this server from a remote client add a -->
<httpEndpoint id="defaultHttpEndpoint"
  host="*"
  httpPort="9080"
  httpsPort="9443" />

</server>

```

Notes: The appSecurity-2.0 feature enables application-level security; the other XML establishes basic authentication using a dummy name ("Fred") and password, as well as a role that allows Fred to access the z/OS Connect servlet. All this is internal to Liberty. There's no underlying SAF or LDAP support here at all yet.

- ❑ Start your server with the **S BBGZSRV,PARMS='server1'** command.
- ❑ Look in the `messages.log` file. You should see the following messages:

```

CWWKS0007I: The security service is starting...
:
CWWKS9120I: Authorization roles with id="zos.connect.access.roles" ...
:
CWWKO0219I: TCP Channel defaultHttpEndpoint-ssl has been started...
:
CWWKS0008I: The security service is ready.

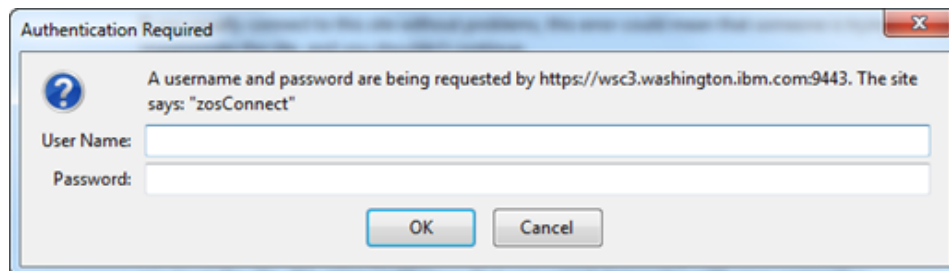
```

- ❑ From a browser, enter the following URL:

https://<host>:9443/zosConnect/services

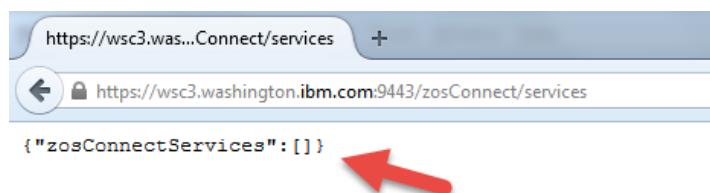
Note: You will be challenged by your browser because the server certificate will be unrecognized. That's because the server certificate created by this process is not signed by a well known certificate authority. If your corporate security policy allows it, accept the certificate as is⁵³.

- ❑ You will be prompted for a userid and password⁵⁴:



Enter the ID of **Fred** and the password of **fredpwd**. Both are case sensitive.

- ❑ You should receive this:



That is z/OS Connect telling you no services are configured. That's correct – you have not yet configured any.

⁵³ This will also result in an FFDC capture by the Liberty Profile server. Ignore that.

⁵⁴ An alternative to basic authentication is to use client certificates. See "Use SAF and client certificates" on page 68 for more on this.

Slight detour: install HTTP client that can send REST with JSON

Up to this point your browser activity has been performed using HTTP `GET` operations. To test z/OS Connect more thoroughly you will need to issue `PUT` or `POST` requests. There are several ways this can be done⁵⁵, and in this document we'll illustrate the use of a browser add-on HTTP REST client – specifically, the Advanced REST Client for the Google® Chrome® browser.

Note: This is *not* necessarily an endorsement of Google, Google Chrome, or the Advanced REST client application. It is merely the illustration of a REST client that was used to drive z/OS Connect. Any REST client that allows sufficient control to form the request properly will work.

Do the following:

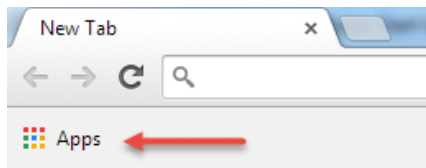
- ☐ Install the Chrome browser if you have not already done so.
- ☐ You will need a Google ID to acquire the no-charge application, so use an existing ID or sign up to create a new ID.
- ☐ Open the Google search engine and search for 'Chrome Advanced REST client' ... you should get a hit that looks like this:

Chrome Web Store - Advanced REST client

<https://chrome.google.com/.../advanced-rest-client/hgmlo...> Google Chrome ▾
 ★★★★★ Rating: 4.5 - 4,864 votes - Free - Developer
 The web developers helper program to create and test custom HTTP requests.

Click on that link and get/install the application. Then close the browser.

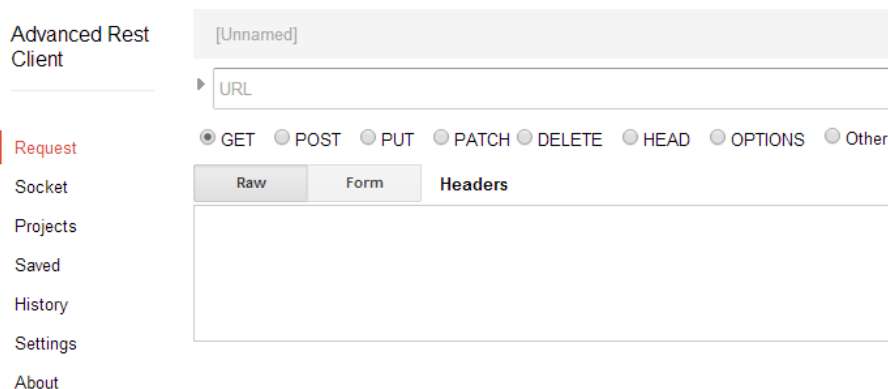
- ☐ Reopen the browser and click on the Apps icon:



- ☐ Click on the icon for the REST client:



- ☐ You should see something like this:



⁵⁵ Web page with a button that performs a POST operation; a program that issues HTTP POST requests; a browser add-on.

- ❑ Close the browser. We'll come back to that after the service has been configured into the Liberty Profile z/OS server and the batch program has been re-started and registered into the server.

Configure the COBOL batch as a z/OS Connect service

To start with we'll use a simple batch program as the backend service. Then we'll use the CICS backend to explore the data transformation function of z/OS Connect.

Update server.xml with service and service provider definitions

Do the following:

- ❑ Edit your `server.xml` file and **add** the lines **highlighted** here:

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
  </featureManager>

  <keyStore id="defaultKeyStore" password="Liberty"/>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <basicRegistry id="basic1" realm="zosConnect">
    <user name="Fred" password="fredpwd" />
  </basicRegistry>

  <authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
      <user name="Fred"/>
    </security-role>
  </authorization-roles>

  <zosConnectService id="zosconn"
    serviceName="batch-backend"
    serviceRef="wolaService" />

  <localAdaptersConnectService id="wolaService"
    registerName="ZOSCONN"
    serviceName="ServiceName"
    connectionFactoryRef="wolaCF" />

  <zosLocalAdapters
    wolaGroup="GROUP"
    wolaName2="NAME2"
    wolaName3="NAME3" />

  <connectionFactory id="wolaCF" jndiName="eis/ola">
    <properties.ola />
  </connectionFactory>

  <!-- To access this server from a remote client add a -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

</server>
```

Notes: The `zosConnectService` element defines the service that will be exposed. The `serviceName` element will be used on the URI to invoke this service. The `serviceRef` element points to the mechanism used to reach the backend process, which in this case is WOLA and the backend process will be a batch program. The `localAdaptersConnectService` provides the definitions used to carry the request over WOLA to the batch program.

- ☐ Save the file. The updates will be dynamically added to your running server.

Discover your new service

Earlier we had you invoke a URI against z/OS Connect to show the configured services, and it returned a null set – that is, `[]`. You'll try again, and this time it will show your new service.

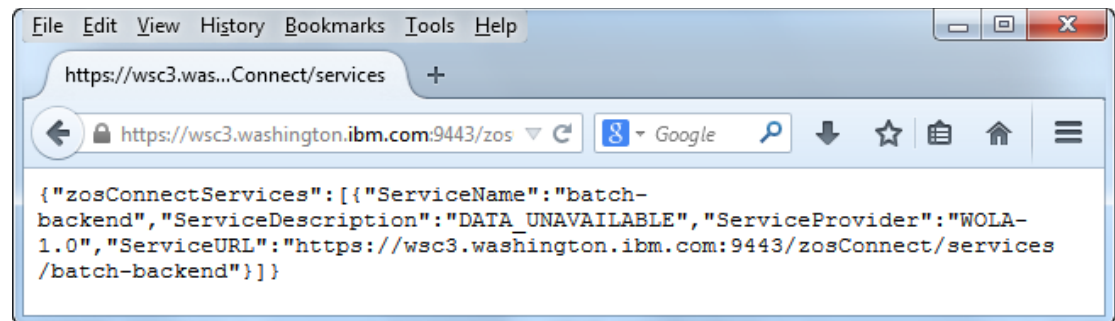
Do the following:

- ☐ From a browser, enter the following URL:

https://<host>:9443/zosConnect/services

Note: Depending on the state of your browser since the last time you issued this URL, you either will or will not be challenged because of the self-signed certificate. If you're challenged because of the certificate, accept it and provide the userid and password at the basic-auth prompt (**Fred** and **fredpwd**).

You should now see something⁵⁶ like this:



That's showing the service you just configured⁵⁷.

Run the batch job and have it register into the Liberty Profile z/OS server

Earlier we had you issue the string `STOP` from the web application, and that caused the batch program to branch-to-exit and stop. It's time to start it again so the Chrome HTTP client can be used to PUT a REST request into z/OS Connect.

- ☐ Restart the `ZOSCONN` batch job.
- ☐ Verify it has registered into the Liberty Profile z/OS server by looking at the held output for the batch job. You should see:

`Successfully registered into GROUP`

⁵⁶ Your host name will be different.

⁵⁷ That's a JSON object, but the browser does not format it well. To see what nicely formatted JSON looks like, see page 36.

Invoke service using REST client

- ❑ Open your Chrome browser and go the URL:
`https://<host>:9443/zosConnect/services`

Important: Accept the browser challenge regarding the untrusted self-signed certificate. It is important you manually do this because the REST client will not do it automatically.
It will also prompt you for ID and password. Enter `Fred` and `fredpwd`.

The result should be the same service listing you saw just a moment ago.

- ❑ Leave that tab as it is and open a *new* tab
- ❑ Open the Advanced REST client application⁵⁸.
- ❑ Do the following:

The screenshot shows the Advanced REST Client application. On the left is a sidebar with navigation links: Request, Socket, Projects, Saved, History, Settings, About, Rate this application, and Donate. The main area is titled '[Unnamed]' and has 'Save' and 'Open' buttons. Below the title is a 'URL' field with a callout '1' pointing to it. Below the URL field is a row of radio buttons for HTTP methods: GET, POST, PUT (selected), PATCH, DELETE, HEAD, OPTIONS, and Other. A callout '2' points to the PUT radio button. Below the radio buttons are tabs for 'Raw', 'Form', and 'Headers'. The 'Raw' tab is selected, and below it is a large text area for the payload with a callout '3' pointing to it. Below the text area is a dropdown menu showing 'application/json' with a callout '4' pointing to it. To the right of the dropdown is the text 'Set "Content-Type" header to overwrite this value.' Below the dropdown and text are 'Clear' and 'Send' buttons, with a callout '5' pointing to the 'Send' button.

Notes:

1. In the URL field, enter the following URL^{59 60}:
`https://<host>:9443/zosConnect/services/batch-backend?action=invoke`
 2. Click the **PUT** radio button
 3. Enter the following in the “raw” payload field:
`{"name": "value"}`
 4. Make sure the drop-down selector shows `application/json`.
 5. Click the “Send” button.
- ❑ You should get back a number of things:
An HTTP “200 OK” status:

Status **200 OK** ⓘ

⁵⁸ See page 31 for a reminder of how that is done.

⁵⁹ While issuing PUT to the non-SSL port may work, the HTTP specification says that GET and HEAD are idempotent but that POST/PUT/DELETE are not. A more proper approach to PUT or POST is to issue directly to the https port as shown here.

⁶⁰ As you'll see, an alternative to this URI pattern is to configure `invokeURI=""` for the service. That allows a URI of your design to be used. See “Configure service with invokeURI” on page 35 for an example of how this is done.

Returned JSON⁶¹:



- ❑ Check the held output of the batch job. You should see:

Your message converted to EBCDIC: {"name":"value"}

That shows the JSON you entered in the client was passed through z/OS Connect and given to the backend batch program.

Configure service with invokeURI

The URL you sent in had `/zosConnect/services` as the first part of the URI string. What if you don't want to expose services with that string? You can use an “invoke URI” element in the `server.xml` to give your service a different value.

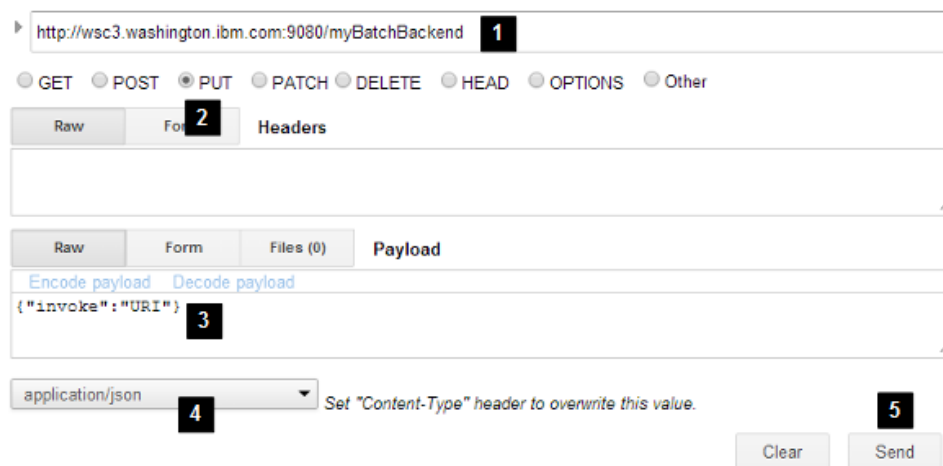
Do the following:

- ❑ Edit the `server.xml` file and add the *following line* to the `zosConnectService` element you added earlier:

```
<zosConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />
```

Note: The use of `invokeURI=` is a general best practice. It shields server configuration internals (service names) from the outside world.

- ❑ Save the file.
- ❑ Go back to the REST client and do the following:



⁶¹ This is JSON that is hard-coded in the ZOSCONN sample batch program. You don't yet have any data conversion routines configured for this instance of z/OS Connect. z/OS Connect will happily pass through to the backend any JSON it receives. However, if it see anything other than JSON coming back it will fail unless data conversion is configured. We'll configure data conversion in a moment, but for now we wanted to show z/OS Connect working. So we hard-coded JSON in the batch program and that's what was returned. z/OS Connect was happy – it saw JSON coming back so it did no data conversion.

Notes:

1. Change the URL to:
`https://<host>:9443/myBatchBackend`
That matches the string you put on the `invokeURI` element.
2. Leave the PUT button selected
3. Code something different in the payload section.
4. Make sure the drop-down still shows `application/json`.
5. Click the “Send” button.

You should see the same result as before – but this time the URI you used was your own custom URI.

- ☐ Check the held output for the batch job. You should see your JSON string, which verifies z/OS Connect received it and passed it through to your backend batch program.
- ☐ Go back to the REST client and change the radio button selection to **GET**. You will find screen changes – the payload section goes away. Click the “Send” button.
- ☐ You should get the same “200 OK” response, as well as the same JSON response. But if you look in the held output for the batch job you'll find:

Your message converted to EBCDIC: `My reply in EBCDIC: {"BestPlatformEver":"z/OS"}`

That is showing *no data* passed into the backend batch program.

Note: HTTP `GET` does not pass data in. The response may contain data, but HTTP `GET` does not pass data in. The `invokeURI` element processed the `GET` and invoked the service with no data. The sample backend batch program, by its design, doesn't care whether data is sent in or not, or what that data is. It just sends the same hard-coded JSON back.

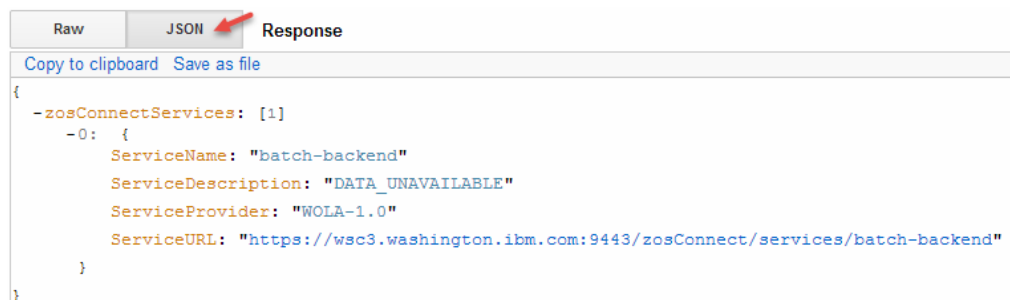
Some services may not require input data. You may want a service to simply respond back with some data based on a simple invoke. The `invokeURI` element handles simple `GET` operations for this purpose.

The `invokeURI` element also handles HTTP `PUT` with JSON, but with a URI pattern more customized to your wishes.

- ☐ While you're in the REST client, issue a `GET` operation with the URL

`https://<host>:9443/zosConnect/services`

The resulting JSON object will be formatted nicely⁶²:



```

{
  -zosConnectServices: [1]
    -0: {
      ServiceName: "batch-backend"
      ServiceDescription: "DATA_UNAVAILABLE"
      ServiceProvider: "WOLA-1.0"
      ServiceURL: "https://wsc3.washington.ibm.com:9443/zosConnect/services/batch-backend"
    }
}

```

- ☐ Cancel⁶³ the sample batch application.

⁶² As opposed to the way a browser simply strung the JSON elements into one long string, as we saw earlier.

⁶³ That sample batch isn't very smart. The code to evaluate the input looks at the first two characters, and if uppercase `ST` it branches to the end and stops. If you passed in JSON `{"STOP"}` the first two characters would be `{ " ...` all JSON starts that way, so it would *not* stop on that passed-in JSON string. We're almost done with this sample batch job, so we can live with this limitation.

Invoke CICS OLACB01 using REST client

Earlier (starting on page 24) you validated WOLA into CICS, and you exercised the OLACB01 sample using the web interface of the supplied sample “OLASampleLibertyWeb” application. But that didn't use z/OS Connect ... you were using a browser-based web application to invoke OLACB01 over WOLA.

The purpose of this section is to update your z/OS Connect `server.xml` to recognize OLACB01 in CICS as the backend service. With that validated we'll then move on to data transformation.

Do the following:

- Add the following to your `server.xml` by *replicating* the existing `zosConnectService` and `localAdaptersConnectService` elements and modifying a few fields.

In the following diagram, gray background is existing; yellow background is new, and fields with white background in yellow highlighting⁶⁴ are updated:

```
<zosConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />

<localAdaptersConnectService id="wolaService"
  registerName="ZOSCONN"
  serviceName="ServiceName"
  connectionFactoryRef="wolaCF" />

<zosConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  serviceRef="wolaCICS" />

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />
```

Notes:

- The initial `zosConnectService` and `localAdaptersConnectService` elements remain to provide access to the batch program if or when it's used again.
- A new set of `zosConnectService` and `localAdaptersConnectService` elements is added for the service to CICS and OLACB01.
- The `id=""` for both new elements is updated to make them unique.
- The `invokeURI=""` for the new `zosConnectService` is updated to something unique.
- The `serviceName=""` for the new `zosConnectService` is updated to something unique.
- The `serviceRef=""` for the new `zosConnectService` element is updated to point to the `id=""` for the new `localAdaptersConnectService` element.
- The `registerName=""` for the new `localAdaptersConnectService` element is changed to match the expected registration name used when the CICS `BBOC START_SRVR` command is entered from the CICS terminal session.
- The `serviceName=""` for the new `localAdaptersConnectService` element is changed to match the backend program to be called – OLACB01.

⁶⁴ If only we had a blinking attribute for PDF. ©

- Go to the Chrome Advanced REST client. If still up from before then skip to the next check-box to-do item. Otherwise, start it and issue the following URL:

https://<host>:9443/zosConnect/services

accept the security challenge caused by the self-signed certificate, then authenticate with the ID of Fred and the password of fredpwd.

- Open a new tab and go to the Advanced Rest Client application. Then Issue an HTTP **GET** operation with the same URL:

https://<host>:9443/zosConnect/services

You should see your new service reported back on the discovery call⁶⁵:



So far so good ... z/OS Connect recognizes your updates. Now let's use the REST client to drive a JSON string into OLACB01 and have it echo the JSON back.

- Then, from a CICS terminal session issue the following command⁶⁶:

BBOC START_TRUE

You should see the following message indicating success:

WOLA TRACE 1: Exit enabled successfully.

- Issue the following command (*as one line*) to start the link server task and register into your server^{67 68}:

BBOC START_SRVR RGN=CICSREG DGN=GROUP NDN=NAME2

SVN=NAME3 SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y

Note: Where GROUP, NAME2 and NAME3 matches the WOLA three-part name you set for your Liberty Profile server back on page 16.

You should see the following message indicating success:

WOLA TRACE 0: Start server completed successfully.

⁶⁵ For the WOLA and CICS part of this document the author switched to a different z/OS system. Access is with a specific IP address. Rather than show that IP address in a public document, it is blacked out in this bitmap capture.

⁶⁶ This can be automated using samples DFHPLTOL and BBOACPLT. See WP101490 Techdoc Quick Start Guide.

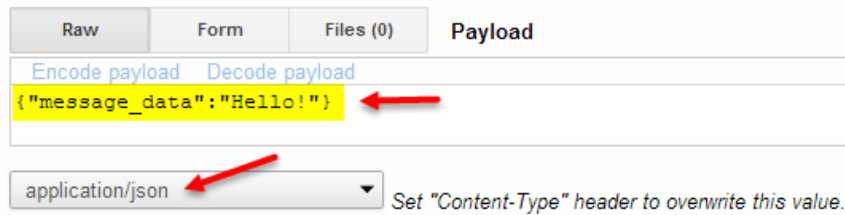
⁶⁷ RGN= may be any value you wish up to 12 characters. CICSREG is just an example of a registration name.

⁶⁸ This can be automated using INITPARM or sequential terminal. See WP101490 Techdoc Quick Start Guide.

- In the REST client, change the URL to:

https://<host>:9443/myCICSBackend

and set the operation to **PUT**. In the payload section, put some JSON such as what's shown here, and make sure the `application/json` setting is in place.



Raw Form Files (0) Payload

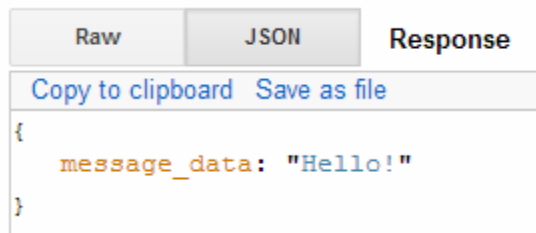
Encode payload Decode payload

`{"message_data": "Hello!"}`

application/json Set "Content-Type" header to overwrite this value.

Then click the "Send" button.

You should then see the following response:



Raw JSON Response

Copy to clipboard Save as file

```
{
  message_data: "Hello!"
}
```

Note: It's important to understand that the REST client sent JSON to z/OS Connect, and because no data transformation function has yet been enabled the JSON was passed back to OLACB01. OLACB01 saw that JSON as a string and simply echoed it back.

That was a good validation of the new service and of WOLA to get to the backend CICS, but it's not real-world. The real world has applications that require multiple fields with multiple data and sending in JSON won't work. The JSON has to be converted to what the backend understands. That's what we'll cover in the next section.

Data transformation and CICS as a backend

The supplied sample program OLACB01 is a simple echo program, and it has a simple DFHCOMMAREA definition⁶⁹:

```
01 DFHCOMMAREA.
05 MESSAGE-DATA PIC X(80).
```

You just validated your WOLA connection to CICS and you saw how OLACB01 received the JSON string and echoed it back. The next step in this journey is to configure the data transformation function ... that is, so the JSON sent in by the REST client is transformed into data in a format other than JSON.

Important: This topic can be somewhat complex, depending on how deeply one goes into it. This document is going to cover the topic at a *high level* and show how to get the function working for a simple example such as OLACB01.

Two URLs are offered for your further study:

http://www.ibm.com/support/knowledgecenter/SS7K4U_8.5.5/com.ibm.websphere.wlp.zseries.doc/ae/twlp_zconnect_msg_payload_conv.html

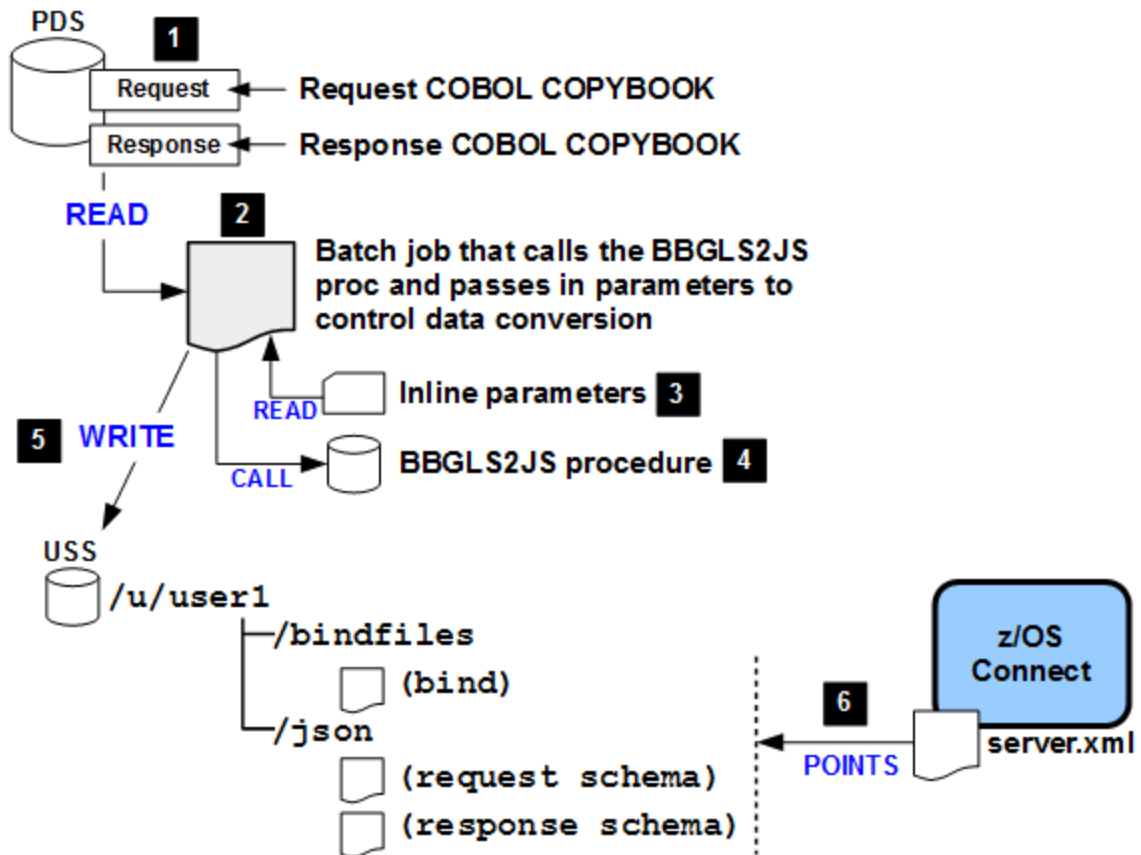
http://www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0_beta/com.ibm.cics.ts.webservices.doc/reference/dfhws_ls2js.html

The first is the data transformation parent topic in the z/OS Connect Knowledge Center, and the second is the CICS Knowledge Center article about the conversion utility on which the z/OS Connect function is based.

⁶⁹ This from the OLACB01 sample we had you pull from GitHub back on page 25.

Overview of the process

The following picture illustrates the process. Numbered blocks correspond to notes that follow:



Notes:

1. The starting point is the request and response data structure for the program. These must be provided in a PDS member. Just the data structure elements are provided here, not any of the other program source code.
2. A JCL job is used to call the `BBGLS2JS` JCL procedure. It's role is to read the inline parameters you supply (block 3) and call the `BBGLS2JS` procedure (block 4).
3. The inline parameters provide the utility knowledge of where the data structure PDS (1) is located and what member names to read in, and various other things to control the generation of the data conversion bind files.
4. The `BBGLS2JS` (LS2JS = "language structure to JSON structure") procedure is supplied with z/OS Connect. This is called by the job (2). No changes are needed to this procedure.
5. The utility writes bind files and JSON schema files out to the UNIX Systems Services location you specify in the inline parameters (3).
6. Updates to the `server.xml` of the Liberty Profile server provide z/OS Connect with knowledge of where the bind and schema files are located.

With the bind and schema files in place, z/OS Connect is able to transform JSON into what the backend program requires, and transform back to JSON the response received.

In the following section you will set this up to perform data transformation for the OLACB01 program in CICS.

BBGLS2JS not supplied with 8.5.5.2

You need the `BBGLS2JS` JCL procedure, but unfortunately it was not provided with the 8.5.5.2 distribution of Liberty Profile server. It is supplied as part of the ZIP provided with the WP102439 Techdoc at ibm.com/support/techdocs.

Do the following:

- ☐ Go to the WP102439 Techdoc⁷⁰.
- ☐ Scroll to locate the “Quick Start Guide” section. That’s where the PDF for this document may be found.
- ☐ Download the ZIP file that accompanies the Quick Start Guide PDF. That ZIP includes the sample Java EAR file as well as other sample files referenced in this document.
- ☐ Extract the `BBGLS2JS` and `BBGJS2LS` files from the ZIP.

Set up data conversion utility structure and generate bind files

Do the following:

- ☐ Determine what FB 80 PDS you will use for the JCL. You may use an existing or allocate a new one.
- ☐ Upload the `BBGLS2JS` and `BBGJS2LS` files into the PDS.
- ☐ Create a member call `CB01COPY` and place in it the data structure from the `OLACB01` sample:

```
01 DFHCOMMAREA.
05 MESSAGE-DATA PIC X(80) .
```

- ☐ Create a member in the PDS with name `LS2JS` and populate it with the following JCL:

```
//LS2JS JOB NOACCT, 'LS2JS Job', CLASS=A, MSGCLASS=A,
// REGION=300M, NOTIFY=&SYSUID
//JCLLIB JCLLIB ORDER=<location_of_BBGL2JS>
//JAVAPROG EXEC BBGLS2JS,
// WLPDIR='INSTALL_DIR'
//INPUT.SYSUT1 DD *
PDSLIB=<PDS_containing_COPYBOOK>
REQMEM=CB01COPY
RESPMEM=CB01COPY
MAPPING-LEVEL=3.0
MINIMUM-RUNTIME-LEVEL=3.0
CHAR-VARYING=COLLAPSE
JSON-SCHEMA-REQUEST=/u/user1/json/CICS-backend_request.json
JSON-SCHEMA-RESPONSE=/u/user1/json/CICS-backend_response.json
LANG=COBOL
LOGFILE=/u/user1/bindfiles/CICS-backend.log
PGMINT=COMMAREA
PGMNAME=OLACB01
WSBIND=/u/user1/bindfiles/CICS-backend.bnd
/*
```

Where:

- `//JCLLIB` points to the PDS where the `BBGL2SJS` procedure is located
- `INSTALL_DIR` is the location where Liberty Profile z/OS is installed. See page 7 for the discussion on installing Liberty Profile 8.5.5.2 and the required iFixes.

⁷⁰ <http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102439>

- `PDSLIB=` points to the PDS where the `OLACB01` data structure member `CB01COPY` resides.
- `JSON-SCHEMA-REQUEST` and `JSON-SCHEMA-RESPONSE` point to a directory and file name for the generated schema files⁷¹. To keep things simple both request and response should point to the same directory.

Note: The name of the JSON files should match the service name. In this example the service name is `CICS-backend`, so the schema names are `CICS-backend_request` and `CICS-backend_response`. This is necessary to allow the `getRequestSchema` and `getResponseSchema` APIs (page 46) to return the service schema.

- `LOGFILE=` and `WSBIND=` both point to a directory and file name for the log file and the bind file.

Note: The bind file name must be equal to the `serviceName=` element in the `server.xml` for this defined service. That's how z/OS Connect knows which bind file to read. Back on page 37 we had you update the `server.xml` with `serviceName="CICS-backend"`. Therefore, here you should name the `WSBIND=` file name `CICS-backend.bnd`.

The example shows this is in a separate directory from the schema. That's not strictly required, but it provides a nice separation of bind file from schema file.

- ❑ This job assumes the ID under which it runs has `JAVA_HOME` and `PATH` properly pointing to a valid Java installation. That is set in the `.profile` for the ID. `RC=127` is a symptom of Java not being found.
- ❑ Submit the job `LS2JS`. Correct any JCL errors or other errors that create a return code greater than 4. A return code of 4 is acceptable.
- ❑ Inspect the directory you specified on `JSON-SCHEMA-REQUEST=`. You should see two files in that directory:

```
CICS-backend_request.json
CICS-backend_response.json
```

- ❑ Inspect the directory you specified on `WSBIND=`. You should see two files in that directory:

```
CICS-backend.bnd
CICS-backend.log
```

The pieces are in place. Now a quick update to `server.xml` and you're ready to test.

Update `server.xml` for and provide data conversion for the CICS service

Do the following:

- ❑ Edit the `server.xml` file and add the following line to the `zosConnectService` element for the CICS backend service⁷²:

```
<zosConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  dataXformRef="xformJSON2Byte"
  serviceRef="wolaCICS" />
```

- ❑ Now add the following block of XML immediately below the CICS backend service definition:

```
<zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/u/user1/bindfiles"
```

⁷¹ The ID under which the job runs must have write access. The Liberty Profile server ID needs to have at least READ to this directory and the files created.

⁷² But not the batch backend service definition. We're not showing data conversion to that service.

```

bindFileSuffix=".bnd"
requestSchemaLoc="/u/user1/json"
responseSchemaLoc="/u/user1/json"
requestSchemaSuffix=".json"
responseSchemaSuffix=".json">
</zosConnectDataXform>

```

Note how the `dataXformRef=` in the service definition points to the `id=` value in the data transformation element.

Make sure the directories for the bind file and JSON schema correctly point to where the files were actually generated. See page 41 for the illustration of the `LS2JS JCL` job where the directories were specified. This is how z/OS Connect will be able to read the bind file and schema and do the data conversion.

Validate using REST client

The time has come to test the service using the REST client.

Do the following:

- Take a look at the JSON *request* schema in the directory where it was generated. It should look like this:

```

{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "Request schema for the OLACB01 JSON interface",
  "type": "object",
  "properties": {
    "OLACB01Operation": {
      "type": "object",
      "properties": {
        "dfhcommarea": {
          "type": "object",
          "properties": {
            "message_data": {
              "type": "string",
              "maxLength": 80
            }
          }
        },
        "required": "Y",
        "message_data": {
          "type": "string",
          "maxLength": 80
        }
      }
    },
    "required": "Y",
    "dfhcommarea": {
      "type": "object",
      "properties": {
        "message_data": {
          "type": "string",
          "maxLength": 80
        }
      }
    },
    "required": "Y",
    "OLACB01Operation": {
      "type": "object",
      "properties": {
        "dfhcommarea": {
          "type": "object",
          "properties": {
            "message_data": {
              "type": "string",
              "maxLength": 80
            }
          }
        },
        "required": "Y",
        "message_data": {
          "type": "string",
          "maxLength": 80
        }
      }
    }
  }
}

```

That determines the structure of the JSON you need to send in to z/OS Connect with the REST client. If the request JSON isn't correctly formed, z/OS Connect will throw an error. The format of the request JSON that matches that schema is⁷³:

```

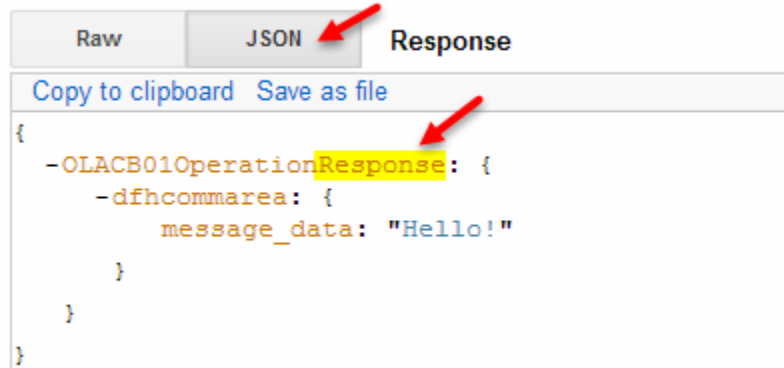
{
  "OLACB01Operation": {
    "dfhcommarea": {
      "message_data": "Hello!"
    }
  }
}

```

⁷³ If you're struggling with the format of JSON, website <http://jsonlint.com/> provides a means of validating JSON.

```
}
}
```

- ❑ Go to your Chrome Advanced REST client⁷⁴ and do the following:
 - PUT with URL of `https://<host>:9443/myCICSBackend`
 - Copy the request JSON from above and paste into the payload section.
 - Make sure pull down shows `application/json`, then click on “Send”
- ❑ You should get back the following:



That is showing the response coming back as JSON.

- ❑ The true test is seeing what is passed into the CICS program over WOLA by z/OS Connect. It should *not* be JSON⁷⁵. It should be just `Hello!`.

Look in the `CEEMSG` held output for the CICS region. You should see:

```
BBO# 20140711154116 Hello!
```

As compared to higher up in `CEEMSG` where you'll see:

```
BBO# 20140711113641 # _ /  / /  %%? ' /
```

The presence of just `Hello!` means z/OS Connect performed the data conversion and formed the data to pass over to CICS by parsing out the content from the JSON.

This also shows that z/OS Connect performed code page conversion. The JSON sent by the REST client was in ASCII. The fact that `Hello!` shows up in readable form in `CEEMSG` indicates its in EBCDIC.

The line with the odd characters is the JSON string sent in unchanged in the earlier test. That is in ASCII, and it's whatever JSON was sent over. No conversion was performed for the earlier test, so z/OS Connect simply passed through the JSON in raw format.

Exercise some of the z/OS Connect REST APIs

Earlier we exercised the generic discovery function to list the configured services. That was simply this URL:

```
http://<host>:9080/zosConnect/services
```

With the data conversion function configured for the CICS-backend service, let's now explore some of the other API functions⁷⁶.

Do the following:

⁷⁴ If it's not up, remember to drive the `/zosConnect/services` URL with a GET and accept the security challenge, then authenticate with the Fred ID and password. Then open a new tab and start the REST client application.

⁷⁵ As it was before we configured data conversion. There it passed JSON over as a string, and OLACB01 echoed it back.

⁷⁶ See “Knowledge Center Links” on page 88 for the search string to find the Knowledge Center article on these APIs

- From the REST client⁷⁷, issue the following URL with a **GET** operation:

https://<host>:9443/zosConnect/services/CICS-backend

You should get back some details on that service:



The screenshot shows a REST client interface with tabs for 'Raw', 'JSON', and 'Response'. The 'Response' tab is selected, displaying a JSON object. The JSON object contains details for the 'zosConnect' service and the 'CICS-backend' service. The 'zosConnect' object includes 'serviceName', 'serviceDescription', 'serviceProvider', 'serviceURL', 'serviceInvokeURL', and 'dataXformProvider'. The 'CICS-backend' object includes 'useCICSContainer', 'linkTaskChanType', 'linkTaskReqContType', 'linkTaskRespContType', 'registerName', and 'serviceName'.

```

{
  -zosConnect: {
    serviceName: "CICS-backend"
    serviceDescription: "DATA_UNAVAILABLE"
    serviceProvider: "WOLA-1.0"
    serviceURL: "https://[redacted]:9443/zosConnect/services/CICS-backend"
    serviceInvokeURL: "https://[redacted]:9443/zosConnect/services/CICS-backend?action=invoke"
    dataXformProvider: "zosConnectXform-1.0"
  }
  -CICS-backend: {
    useCICSContainer: false
    linkTaskChanType: 0
    linkTaskReqContType: 0
    linkTaskRespContType: 0
    registerName: "CICSREG"
    serviceName: "OLACB01"
  }
}

```

- Now query for the status with an HTTP **GET** operation:

https://<host>:9443/zosConnect/services/CICS-backend?action=status

You should see a status of "Started"⁷⁸:



The screenshot shows a REST client interface with tabs for 'Raw', 'JSON', and 'Response'. The 'Response' tab is selected, displaying a JSON object. The 'zosConnect' object includes 'serviceName', 'serviceDescription', 'serviceProvider', 'serviceURL', 'serviceInvokeURL', 'dataXformProvider', and 'serviceStatus'. The 'serviceStatus' is highlighted in yellow and labeled 'Started' with a red arrow pointing to it.

```

{
  -zosConnect: {
    serviceName: "CICS-backend"
    serviceDescription: "DATA_UNAVAILABLE"
    serviceProvider: "WOLA-1.0"
    serviceURL: "https://[redacted]:9443/zosConnect/services/CICS-backend"
    serviceInvokeURL: "https://[redacted]:9443/zosConnect/services/CICS-backend?action=invoke"
    dataXformProvider: "zosConnectXform-1.0"
    serviceStatus: "Started"
  }
}

```

- Using a **PUT** operation, stop the service (with no JSON payload):

https://<host>:9443/zosConnect/services/CICS-backend?action=stop

You should get back an indication the service is stopped:

```

dataXformProvider: "zosConnectXform-1.0"
serviceStatus: "Stopped"
}

```

The 'serviceStatus' is highlighted in yellow and labeled 'Stopped' with a red arrow pointing to it.

- Prove the service is indeed stopped by issuing a **PUT** operation using the same URL and JSON payload you used earlier (page 44) to validate. You should see back:

CWWKB0451I: The CICS-backend service is stopped. The request is rejected.

⁷⁷ Going forward we will assume you have accepted the security challenge regarding the certificate, and you have authenticated.

⁷⁸ A note of caution: that does not necessarily mean the backend WOLA connection is up and ready to go.

- Start the service with the following URL and a **PUT** (no JSON payload) operation:

`https://<host>:9443/zosConnect/services/CICS-backend?action=start`

Expected response:

```
dataXformProvider: "zosConnectXform-1.0"
serviceStatus: "Started"
}
```

- Prove the service is indeed re-started by issuing a **PUT** operation using the same URL and JSON payload you used earlier (page 44) to validate. You should see back:

```
{
  -OLACB01OperationResponse: {
    -dfhcommarea: {
      message_data: "Hello!"
    }
  }
}
```

- Retrieve the request JSON configured for this service by issuing a **GET** operation with the following URL:

`https://<host>:9443/zosConnect/services/CICS-backend?action=getRequestSchema`

You should see:



```
Raw  JSON  Response
Copy to clipboard  Save as file
{
  $schema: "http://json-schema.org/draft-04/schema#"
  description: "Request schema for the OLACB01 JSON interface"
  -properties: {
    -OLACB01Operation: {
      -properties: {
        -dfhcommarea: {
          -properties: {
            -message_data: {
              maxLength: 80
              type: "string"
            }
          }
          -required: [1]
          0: "message_data"
          type: "object"
        }
      }
      -required: [1]
      0: "dfhcommarea"
      type: "object"
    }
  }
  -required: [1]
  0: "OLACB01Operation"
  type: "object"
}
```

And that should match the JSON file that was generated by the conversion utility you ran back on page 41.

- Get the *response* schema by changing the URL slightly:

`https://<host>:9443/zosConnect/services/CICS-backend?action=getResponseSchema`

You should get back the JSON response schema.

Enable the audit interceptor

The supplied audit interceptor⁷⁹ will write SMF 120.11 records providing information about usage of z/OS Connect⁸⁰. If you go to the Knowledge Center⁸¹ you will find an article that details the fields in the SMF 120.11 record.

In this section we will cover how to configure the audit interceptor to cut SMF 120.11 records, and how to validate the function is working properly. But we won't go into too much detail about SMF in general. In other words, this section is intended for those who have a working knowledge of SMF to begin with.

Verify system requirements to write SMF 120.11 records

Depending on the system you're using for this test of Liberty Profile and z/OS Connect, the SMF 120.11 records may or may not be written. Further, the Liberty Profile server itself may not be authorized to write records. Here we'll verify things will work as we need them to.

Do the following:

- ❑ Work with your system administrator and determine which PARMLIB SMFPRMxx member is in use. Then check to make sure 120 records are indicated to be captured.

Note: In particular, make sure the capture is not for just 120.9, which is the SMF record format for the full function WAS z/OS profile. `TYPE(120(9))` would indicate *just* the 120.9, while `TYPE(120)` would include all subtypes, including 120.11.

- ❑ Work with your security administrator and determine if the Liberty Profile server ID (see page 8) has at least `READ` access to the SAF FACILITY class `BPX.SMF`. If not, then grant that access.

Update server.xml with interceptor information

The updates to `server.xml` are relatively simple. Do the following:

- ❑ Update your `server.xml` with the lines highlighted below (we're showing the full `server.xml` at this point so you can see it all in context):

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

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>appSecurity-2.0</feature>
        <feature>zosConnect-1.0</feature>
        <feature>zosLocalAdapters-1.0</feature>
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty"/>

    <webAppSecurity allowFailOverToBasicAuth="true" />

    <basicRegistry id="basic1" realm="zosConnect">
        <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="zos.connect.access.roles">
        <security-role name="zosConnectAccess">
            <user name="Fred"/>
        </security-role>
    </authorization-roles>
</server>
```

⁷⁹ z/OS Connect comes with a service provider interface (SPI) that allows you to write your own interceptor if you wish. That topic is outside the scope of this document.

⁸⁰ There is a REST API that provides usage statistics as well. That is something different from the audit interceptor. The SMF records provide a more detailed view of usage. The REST API is a bit more high-level.

⁸¹ See "Knowledge Center Links" on page 88.

```

    </security-role>
</authorization-roles>

<zosConnectManager
  globalInterceptorsRef="interceptorList_g" />

<auditInterceptor id="audit" />

<zosConnectInterceptors
  id="interceptorList_g"
  interceptorRef="audit"/>

<zosConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />

<localAdaptersConnectService id="wolaService"
  registerName="ZOSCONN"
  serviceName="ServiceName"
  connectionFactoryRef="wolaCF" />

<zosConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  dataXformRef="xformJSON2Byte"
  serviceRef="wolaCICS" />

<zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/u/user1/bindfiles"
  bindFileSuffix=".bnd"
  requestSchemaLoc="/u/user1/json"
  responseSchemaLoc="/u/user1/json"
  requestSchemaSuffix=".json"
  responseSchemaSuffix=".json">
</zosConnectDataXform>

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />

<zosLocalAdapters
  wolaGroup="GROUP"
  wolaName2="NAME2"
  wolaName3="NAME3" />

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<!-- To access this server from a remote client add a host -->
<httpEndpoint id="defaultHttpEndpoint"
  host="*"
  httpPort="9080"
  httpsPort="9443" />
</server>

```

Notes:

- The `<auditInterceptor>` element is used to define the IBM-supplied SMF 120.11 audit interceptor.

- The `<zoscConnectInterceptors>` element is used to create an interceptor *list*. The list has a name (`id="interceptorList_g"`) and points to the interceptor to use with `interceptorRef="audit"`.
- The `<zoscConnectManager>` element is used to define global interceptors, and in this case it provides a reference to the list to use, which is the `"interceptorList_g"` list.
- With that global definition in place, the audit interceptor will apply to all the services defined.

- ☐ Save the file. Liberty Profile will dynamically update with the changes.

Drive the REST service a few times

The SMF 120.11 records are cut for each request that comes into the Liberty Profile server and invokes z/OS Connect. Do the following:

- ☐ From the z/OS console, issue `/I SMF` to switch the SMF data sets⁸².
- ☐ Go back to the Chrome Advanced REST client application.
- ☐ As you did earlier (see page 44), drive the CICS-backend service with a `PUT` operation and with the JSON payload you used earlier.
- ☐ Send a few more of the same requests.
- ☐ Switch the SMF data set again: `/I SMF`
- ☐ Dump that data set using your normal SMF dump utility. Read the output and make sure 120 records were processed as part of that dump operation.

Use SMF browser to see the records

The SMF browser is a Java program that reads the SMF dump data set and formats the results in a more human-readable format.

The SMF browser does not replace the more powerful SMF analysis tools that exist. You would use those tools to analyze the data captured in the SMF 120.11 records. The browser is handy for this exercise because it's free and relatively simple to setup and use.

Do the following:

- ☐ Go to this website:
https://www.ibm.com/services/forms/preLogin.do?source=zosos390&S_PKG=smf5
- ☐ It will prompt you for a userid and password. This is the “universal IBM user ID,” and if you don't have one you can acquire one using the link on that same page:

Not registered?

If you do not have a universal IBM user ID, please [register here](#), then return to sign in for this offering.

Then go back to the URL above and sign in.

- ☐ Agree to terms and confirm:

I agree*

☒ I agree

By clicking the "I confirm" button below, I confirm my Pr license. By clicking the "I cancel" button, I cancel my P license.

I confirm

I cancel

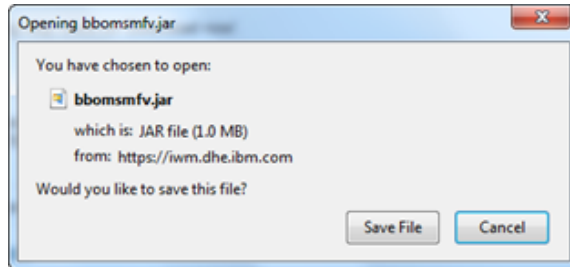
⁸² Or whatever your system command is to switch the SMF data set. We're doing this simply to start with a clean data set so the few requests you send here are not part of a massive capture of other SMF records.

- ☐ Download the file:

SMF Browser for WebSphere
bbomsmfv.jar (795 KB)

[Download now](#)

It will be in the form of a single file (bbomsmfv.jar). Save the file:



What you will do next is create a shell script that will set up the `JAVA_HOME` and `CLASSPATH` variables, and then invoke the SMF browser utility to read in the dumped SMF data. Do the following:

- ☐ Capture a few values you will need:

<u>your_directory</u> This is where the shell script will live, and is where the <code>bbomsmfv.jar</code> file will be uploaded. Choose any directory you wish.	
<u>JAVA_HOME</u> You captured the path for Java back on page 8. Use the same value here.	
<u>SMF.DATASET</u> This is the MVS data set where the SMF records were dumped.	

- ☐ Create a shell script called `runSMF.sh` in the directory you chose and populate it with the following⁸³:

```
export JAVA_HOME=/ JAVA\_HOME
export CLASSPATH=/ your\_directory /bbomsmfv.jar
export CLASSPATH=$CLASSPATH:/$JAVA_HOME/lib/ext/ibmjzos.jar
java com.ibm.ws390.sm.smfview.SMF "INFILE(SMF.DATASET) "
```

Where:

- [JAVA_HOME](#) is the path to Java you captured
 - [<your_dir>](#) is the directory in which this shell script resides and, more importantly, where the `bbomsmfv.jar` file will be located.
 - [SMF.DATASET](#) is the SMF dump data set that has the SMF records to be processed by the browser.
- ☐ Upload the `bbomsmfv.jar` file to the directory you will use for the shell script. Make the permissions so the ID you will run under has `READ`.
- ☐ Run the shell script: `./runSMF.sh`

⁸³ The `bbomsmfv.jar` file contains the SMF browser class file that will be invoked. The `ibmjzos.jar` file is needed to read the records from the MVS data set. This puts both on the `CLASSPATH`, then the browser class is invoked and the input file is passed in.

- For *each* SMF 120.11 record⁸⁴ in the output, you should see something like this:

```
-----
Record#: 2;
Type: 120; Size: 2212; Date: Tue Jul 15 14:13:06 EDT 2014;
SystemID: WG31; SubsystemID: WAS; Flag: 94;
Subtype: 11 (Liberty Request Activity);

#Subtype Version: 1;
Index of this record: 0;
Total number of records: 1;
record continuation token      * 00000000 00000000 ----- *
#Triplets: 2;
Triplet #: 1; offsetDec: 72; offsetHex: 48; lengthDec: 80; lengthHex: 50; count: 1;
Triplet #: 2; offsetDec: 152; offsetHex: 98; lengthDec: 2060; lengthHex: 80c; count: 1;

Triplet #: 1; Type: LibertyServerInfoSection;
  Server Info Version      : 1;
  System Name (CVTSNAME)   : WG31;
  Sysplex Name             : WSLPLEX;
  JobID                    : STC00250;
  JobName                  : BBGZSRV;
  Stoken                   * 000000f8 00000014 ----- *

Triplet #: 2; Type: UserDataSection;
  Version      : 1;
  Data Type    : 102;
  Data Length  : 251;
  data version : 1;
  arrival Time      * cd7507b4 fa0dc3b1 ----- *
  completion Time   * cd7507b4 fd2e9129 ----- *
  target URI        : /zosConnect/services/CICS-backend;
  input Length      : 62;
  service name      : CICS-backend;
  method name       : PUT;
  response Length    : 70;
  userid            : Fred;
-----
```

The **yellow** highlighting is simply to draw your attention to a few things in the formatted record. For more information on the record format for SMF 120.11, see the Knowledge Center and search on string `twlp_zconnect_audit`.

At this point we are finished exploring the SMF 120.11 audit interceptor. You have configured the function and validated the records are being written.

For those who have a deeper background in SMF analysis, the next step would be to better understand the specifics of the record layout and determine if the more powerful SMF analysis tools used in the organization recognize the SMF 120.11 layout.

Configuration checkpoint: server.xml at this point

See “Checkpoint 2: After audit interceptor configuration” on page 82 for a sample of the `server.xml` at this point in the flow of this document.

Status at this point

At this point you've exercised the z/OS Connect function against both a batch backend service and a CICS backend service. You used the REST client to drive the requests into z/OS Connect. You configured the data conversion function to translate JSON to COPYBOOK and back. And you enabled the audit interceptor to write SMF 120.11 records.

⁸⁴ If you drove the service request several times, you'll see several of these blocks of output.

Section 4: More security - SAF, Authentication, and Client Certificates

Up to this point we've used the basic registry for identity and authorization because it was simple. It allowed us to keep the focus on getting the components of Liberty, WOLA and z/OS Connect up and running. But basic registry is not real-world. So we move now to using SAF.

Use SAF for SSL keystore and user registry

In this section we'll accomplish two things – (1) enable SAF⁸⁵ as the store for the Certificate Authority and server certificates, and (2) enable SAF for authentication and authorization.

Enable SAF as keystore for CA and server certificates⁸⁶

Earlier we saw SSL in action – the z/OS Connect application requires SSL, and your browser challenged you to inspect and allow the default server certificate included with Liberty Profile z/OS.

In this section we are going to create a CA certificate, then use that CA to sign a server certificate for the Liberty Profile server. We'll create a SAF keyring that is accessible by the ID under which Liberty runs.

Note: Your browser will *still* challenge you to inspect the certificate and accept it. The CA used to sign the server certificate is *not* a “well-known” CA⁸⁷. It will be a different certificate from earlier.

Illustrating how to use SAF for certificates brings you closer to understanding how a *well-known* CA would be used to sign your server certificate for use on z/OS.

Do the following:

- ☐ Stop your server (the Angel process may stay up).
- ☐ Work with your security administrator to issue the following commands:

Note: These commands are long and broken across lines in this document. On your system you enter them as one long line.

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty')
                                     OU('LIBERTY')) WITHLABEL('LibertyCA.LIBERTY')
                                     TRUST NOTAFTER (DATE (2018/12/31))

RACDCERT ADDRING(Keyring.LIBERTY) ID(liberty_id)

RACDCERT ID (liberty_id) GENCERT SUBJECTSDN(CN('yourhost.com')
                                     O('IBM') OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY')
                                     SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY'))
                                     NOTAFTER (DATE (2018/12/31))

RACDCERT ID(liberty_id) CONNECT (LABEL('DefaultCert.LIBERTY')
                                RING(Keyring.LIBERTY) DEFAULT)

RACDCERT ID(liberty_id) CONNECT (LABEL('LibertyCA.LIBERTY')
                                RING(Keyring.LIBERTY) CERTAUTH)

SETROPTS RACLIST(FACILITY) REFRESH
```

Where:

- *liberty_id* is the ID under which your Liberty Profile server runs. This you planned back on page 8.
- *yourhost.com* is the host name of your system.

⁸⁵ Applicable to any SAF product. Here we illustrate IBM's SAF product: RACF®.

⁸⁶ This is related to Liberty Profile z/OS itself rather than z/OS Connect specifically. The information here is from the Liberty Profile z/OS Techdoc at ibm.com/support/techdocs under number WP102110. The “Quick Start” guide there has this information as well.

⁸⁷ For example, VeriSign®, or some other well-known CA that browsers today recognize.

- Continue working with your security administrator and create the following `SERVER` class profile⁸⁸ and grant your server ID `READ` to it:

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SAFCRED UACC(NONE) OWNER(SYS1)
```

```
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED CLASS(SERVER)
ACCESS(READ) ID(liberty_id)
```

```
SETROPTS RACLIST(SERVER) REFRESH
```

- Update your `server.xml` with the lines highlighted here:

```
:
<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<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" />

  <!-- To access this server from a remote client add a host attribute
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

</server>
```

- Stay in your edit session and go to the top of `server.xml`. Add the following two lines:

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

  :
```

- Save the file.

Remove basicRegistry and use SAF for registry and zosConnectAccess role checking

Now we're going to make a transition away from the XML-based registry to having userids and passwords stored in SAF. In addition, we will move the role definition⁸⁹ from the `server.xml` to SAF as well. Every other aspect of z/OS Connect will remain the same for now. Do the following:

⁸⁸ See "SERVER profiles" on page 77 for a listing of all the SERVER profiles related to Liberty Profile z/OS and z/OS Connect.

⁸⁹ z/OS Connect protects who may access the application. A valid ID is not enough; you must also have access to the access role.

- For this test we will keep using the ID of “FRED,” but have it in SAF rather than coded in the `server.xml`. Therefore, work with your security administrator to create the following group and ID:

```
ADDGROUP FREDGRP OMVS (GID(gid) ) OWNER(SYS1)
```

```
ADDUSER FRED DFLTGRP (FREDGRP) OMVS (UID(uid) HOME(/path)  
PROGRAM(/bin/sh)) NAME('TEST ID OF FRED') NOPASSWORD
```

```
ALTUSER FRED PASSWORD(xxxxxxx) NOEXPIRED
```

Note: If you wish to use an existing ID that will work as well. Replace `FRED` in the steps below with the ID you choose for this validation exercise.

- Edit your `server.xml` and remove the lines indicated with the ~~strike-through~~ and add the lines **highlighted**, then save the file

```
:  
<keyStore id="defaultKeyStore" password="Liberty"/>  
  
<webAppSecurity allowFailOverToBasicAuth="true" />  
  
<basicRegistry id="basic1" realm="zosConnect">  
  <user name="Fred" password="fredpwd"/>  
</basicRegistry>  
  
<authorization roles id="zos.connect.access.roles">  
  <security role name="zosConnectAccess">  
    <user name="Fred"/>  
  </security role>  
</authorization roles>  
  
<safRegistry id="saf" />  
<safAuthorization id="saf" />  
<safCredentials unauthenticatedUser="WSGUEST"  
  profilePrefix="BBGZDFLT" />  
  
<zosConnectManager  
  globalInterceptorsRef="interceptorList_g" />  
:
```

Notice two things:

1. The `profilePrefix` value of `BBGZDFLT`. Several SAF profiles will be based on what's coded for this.
2. The `unauthenticatedUser` value of `WSGUEST`. That comes into play on the `APPL` class profile. `WSGUEST` is the default; we coded it here to draw your attention to how it would be coded if you wished to use something other than the default.

To do item:

○ If you wish to use something *other than* `WSGUEST` then code that ID value here⁹⁰.

- If you look in the XML updates above, we *removed* the section that named the authorization roles and granted Fred access to them. Now we're going to put the equivalent in SAF:

```
RDEFINE EJBROLE BBGZDFLT.zos.connect.access.roles.zosConnectAccess  
                                OWNER(SYS1) UACC(NONE)  
  
PERMIT BBGZDFLT.zos.connect.access.roles.zosConnectAccess  
                                CLASS(EJBROLE) ID(FRED) ACCESS(READ)  
  
SETR RACLIST(EJBROLE) REFRESH
```

⁹⁰ That ID should be a very limited authority ID. It should *not* connect to any group that has authority. Be careful what ID you put here.

Notes: If you're using another ID rather than `FRED`, put that ID on the `PERMIT`.

Rather than grant each ID access to the `EJBROLE`, you may grant a Group ID access. Then all IDs connected to that group will have access. If a new user comes along you simply connect that new user to this group and they will have access to the `EJBROLE`.

Notice how the `EJBROLE` starts with the `profilePrefix=` value we coded in the `server.xml`. See "The security profile prefix value" on page 79 for more on why you may wish to have unique prefix values for different Liberty Profile servers.

- ☐ Create another `EJBROLE`:

```
RDEF EJBROLE ** OWNER(SYS1) UACC(NONE)
SETR RACLIST(EJBROLE) REFRESH
```

Note: This one is optional. And it requires a bit of explaining.

This `EJBROLE` provides a way to insure an `ICH408I` message is issued in cases where no other role definition applies. The `ICH408I` message thrown will indicate what role was sought and what access to that role was sought. If the role requested doesn't match anything, then no `ICH408I`. This role will catch everything, but with a `UACC(NONE)` it doesn't permit anything. It is simply a way to make certain an `ICH408I` message is issued to assist with debugging if needed.

- ☐ There's another `SERVER` profile that will be checked, and it is also based on the "security prefix" value in use by your Liberty Profile server:

```
RDEFINE SERVER BBG.SECPFX.BBGZDFLT UACC(NONE)
PERMIT BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(liberty_id)
SETROPTS RACLIST(SERVER) REFRESH
```

Where `liberty_id` is the ID of the server, not the user (`FRED` or someone else) trying to access the application. This `SERVER` profile controls whether the server has authority to access the `EJBROLE` definitions in SAF.

- ☐ We have one more SAF profile to create. It's an `APPL` class profile, and it too is based on the security prefix value for the server. In the previous step we determined we would use the default security prefix of `BBGZDFLT`. The `APPL` profile is based on that. Users seeking to authenticate need `READ` to the profile.

Work with your security administrator to create the following:

```
RDEFINE APPL BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(unauth_ID)
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(*)
SETR RACLIST(APPL) REFRESH
```

Note: Match `unauth_ID` to what's coded in the `server.xml`

- ☐ We saw in the XML update that an unauthenticated user was specified. We mentioned that without one specified, Liberty Profile would assume a default value of `WSGUEST`.

Key: That ID must exist in SAF. If it does not exist, then the following message is thrown:

```
CWWKS2905E: SAF service IRRSIA00_CREATE did not succeed because user
WSGUEST was not found in the SAF registry.
```

The next step will have you check for the existence of that ID, and if it doesn't exist we will have you create it.

Do the following:

- Look at your `server.xml` and verify what value you have there
- Determine if that ID exists in SAF, and if so, whether or not that's ID you really wish to use as the unauthenticated userid. If not, then go back to the `server.xml` and change it to `WSGUEST`.
- If `WSGUEST` does not exist, then create a group and ID. Suggested commands are:

```
ADDGROUP WSGUESTG OMVS (GID(gid))
ADDUSER WSGUEST RESTRICTED DFLTGRP(WSGUESTG) OMVS (UID(uid)
      HOME(/path) PROGRAM(/bin/sh)) NAME('UNAUTHENTICATED USER')
      NOPASSWORD NOOIDCARD
```
- ❑ Restart your Liberty Profile server.
- ❑ When the server come, point your browser at the following URL:
`https://<host>:9443/zosConnect/services`
- ❑ You will be challenged because the certificate is not signed by a well-known CA. We knew that was going to happen. But if you choose to “view the certificate⁹¹,” you will see that it is based on the SAF commands you entered a few steps earlier:

Issued To

Common Name (CN) wg31.washington.ibm.com
 Organization (O) IBM
 Organizational Unit (OU) LIBERTY
 Serial Number 01

Issued By

Common Name (CN) CA for Liberty
 Organization (O) CA for Liberty
 Organizational Unit (OU) LIBERTY

Validity

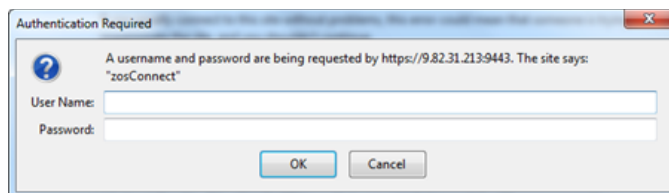
Issued On 7/16/2014
 Expires On 12/31/2018

Fingerprints

SHA1 Fingerprint ED:94:0F:27:04:D2:95:81:66:C6:6A:88:10:C0:EC:14:7D:73:2B:9E
 MD5 Fingerprint BA:AE:A7:74:06:32:56:83:77:9B:23:5B:E4:92:01:72

That verifies SAF is being used for the SSL certificate key and trust store.

- ❑ Accept the certificate challenge.
- ❑ When the same basic authentication prompt appears, enter the ID and password that is in SAF (either the `FRED` ID you created on page 54, or another ID you chose and granted `READ` to the `SERVER` profiles over the past few steps):



- ❑ You should see JSON indicating the services defined to z/OS Connect:

```
{ "zosConnectServices": [ { "ServiceName": "CICS-
backend", "ServiceDescription": "DATA_UNAVAILABLE", "ServiceProvider": "WOLA-
1.0", "ServiceURL": "https://9.82.31.213:9443/zosConnect/services/CICS-backend" },
{ "ServiceName": "batch-
backend", "ServiceDescription": "DATA_UNAVAILABLE", "ServiceProvider": "WOLA-
1.0", "ServiceURL": "https://9.82.31.213:9443/zosConnect/services/batch-backend" } ] }
```

⁹¹ How to do that depends on the browser you are using. All are fairly intuitive in how to view the certificate when challenged.

If you get “Error 403: AuthorizationFailed” then the ID you entered does not have the appropriate authority. Check the MVS console for ICH408I messages. For example, if you had entered the name ETHEL then you'd see:

```

ICH408I USER(ETHEL) GROUP(SYS1 ) NAME(ETHEL IN ACCOUNTING)
        BBGZDFLT.zos.connect.access.roles.zosConnectAccess
        CL(EJBROLE )
        INSUFFICIENT ACCESS AUTHORITY
        ACCESS INTENT(READ ) ACCESS ALLOWED(NONE )
  
```

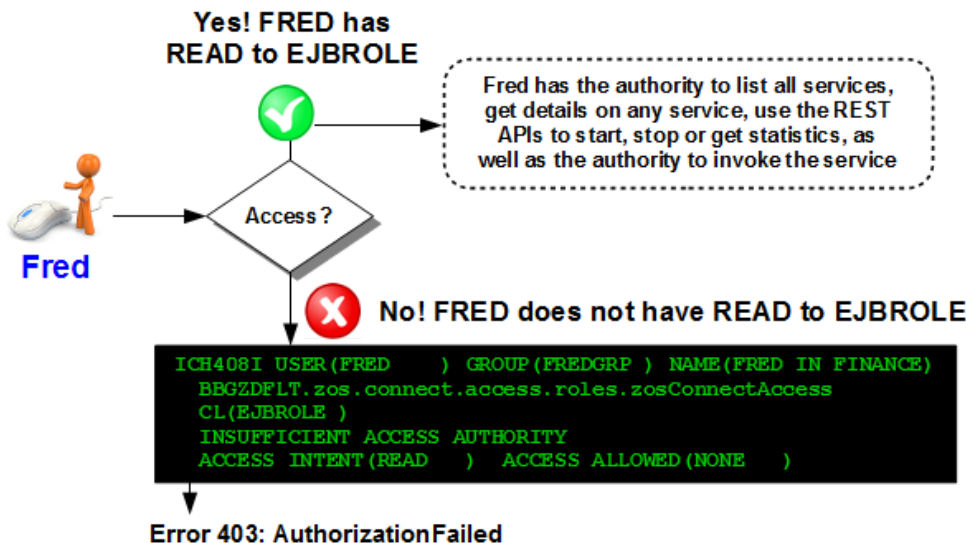
That's saying that the ID of “ETHEL” tried access and she did not have READ to the EJBROLE that permitted access to the z/OS Connect application. FRED had read, ETHEL did not.

Enable the authorization interceptor

In the previous section we did two things:

1. We moved the userid/password registry from `server.xml` to SAF
2. We moved the application role⁹² definition and checking from `server.xml` to SAF

In the absence of the authorization interceptor function we're about to explore, that “zosConnectAccess” role is fairly course-grained. That is, if an ID is able to pass the role check and “get in the front door,” then that ID is has the ability to perform any z/OS Connect function:



A more granular security model exists. That's covered next.

Review: a more granular security model

z/OS Connect offers a more fine-grained security access model. It defines three roles with different degrees of authority:

- *Administrator* – a user with administrator authority will have the authority to query services, perform operational tasks on them, and invoke them.
- *Operations* – a user with operations authority will be able to perform operational tasks on services (for example, start or stop). But they will *not* have authority to invoke the service.
- *Invoke* – a user with invoke authority will be able to invoke the service, but have no other authority.

⁹² The authorization role was `<security-role name="zosConnectAccess">`

The ID⁹³ *still* needs to “get past the front door” by having `READ` to the `zosConnectAccess` `EJBROLE` we had you define on page 54. But once it passes that check, then the next level checking takes place – whether *Administrator*, *Operations* or *Invoke* applies.

At a very high level, the z/OS Connect authorization interceptor function checks to see if the ID is a member of the group named as either an admin group, an operations group or an invoke group. If the ID is a member, then that ID gets the authority.

There are two levels – global and service-level. In the steps that follow you will first exercise the global authorization interceptor, then you will introduce a service-level interceptor.

Userids and access to the `zosConnectAccess` role

You have the ID “Fred” to use during the exercises. We'll need two more.

Do the following:

- ☐ Determine what IDs you will use⁹⁴ in addition to the “Fred” ID:

ID in this document	The ID you will use
UserX	
UserY	

- ☐ Give your IDs `READ` to the `EJBROLE` profile that controls access to the z/OS Connect servlet (Fred should already have it):

```
PERMIT BBGZDFLT.zos.connect.access.roles.zosConnectAccess
                                CLASS (EJBROLE) ID (<your_UserX>) ACCESS (READ)

PERMIT BBGZDFLT.zos.connect.access.roles.zosConnectAccess
                                CLASS (EJBROLE) ID (<your_UserY>) ACCESS (READ)

SETR RACLIST (EJBROLE) REFRESH
```

Create Authorization Groups

Now create some groups that will serve as the authorization groups:

- ☐ Create the following groups⁹⁵:

```
ADDGROUP GADMIN OMVS (GID (<gid>))
ADDGROUP GOPERS OMVS (GID (<gid>))
ADDGROUP GINVOKE OMVS (GID (<gid>))
ADDGROUP SADMIN OMVS (GID (<gid>))
ADDGROUP SOPERS OMVS (GID (<gid>))
ADDGROUP SINVOKE OMVS (GID (<gid>))
```

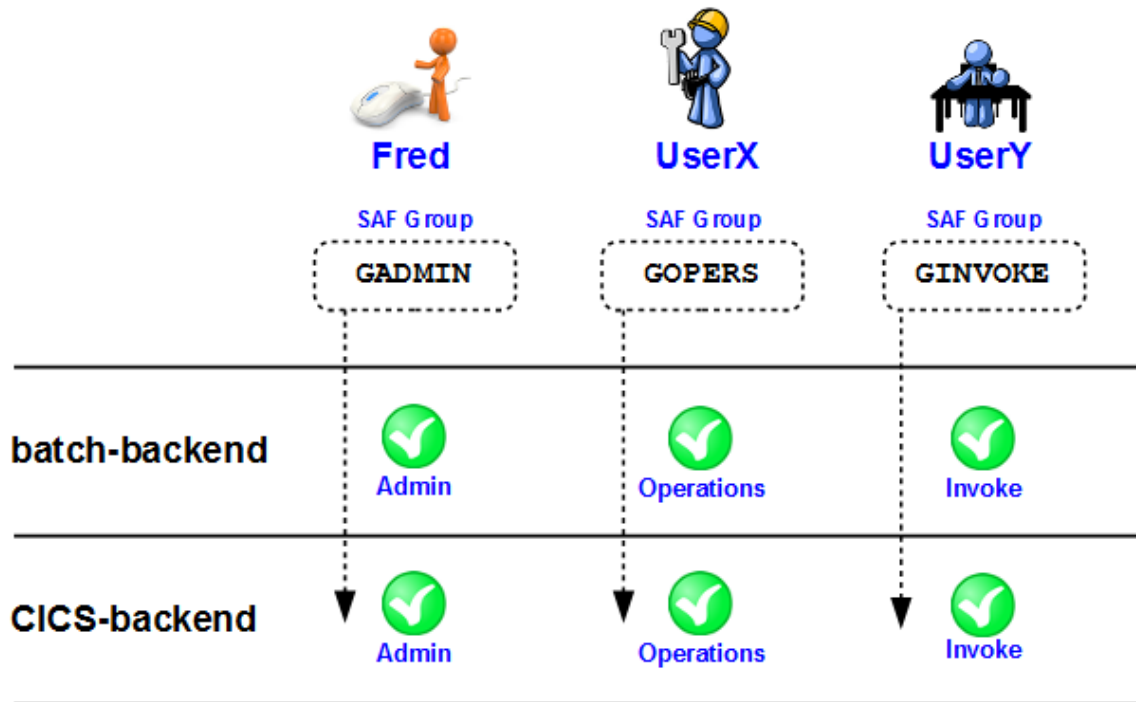
⁹³ Or the group to which the ID belongs. If you grant a group `READ` to the `EJBROLE`, then later you can simply connect IDs to the group and they will have access as well. That is a simpler way of controlling access to the `EJBROLE`.

⁹⁴ Any valid SAF ID will work. You will need to know the password for the ID because you will be prompted to authenticate with that PW.

⁹⁵ These group names are examples. You may use any group name meaningful to you.

Review: first set of authorization exercises

The following picture illustrates the testing we'll do in the next set of instructions:



You should see the following:

- Fred should have Admin over both services. That means Fred should be able to see both services, start and stop both, and invoke both.
- UserX should have Operations over both services. That means it should be able to perform operations actions (start, stop, status) but *not* invoke the services.
- UserY should have Invoke over both services. That means it should be able to invoke both services but not perform operations actions.

Connect users to the authorization groups

For this first exercise, connect your three users to the global groups.

- ☐ Issue the following commands:

```
CONNECT FRED GROUP (GADMIN)
```

```
CONNECT <your_UserX> GROUP (GOPERS)
```

```
CONNECT <your_UserY> GROUP (GINVOKE)
```

Changes to the server.xml in support of the global authorization scenario

To support the use of the global authorization groups, you will need to make some changes to your `server.xml`.

- ☐ Update your `server.xml` with the changes⁹⁶ highlighted here. See below for an brief explanation of the changes.

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
```

⁹⁶ Some of the changes are new lines, others are updates to existing lines. Compare this with the `server.xml` you have right now and the changes should become apparent.

```

    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
</featureManager>

<webAppSecurity allowFailOverToBasicAuth="true" />

<safRegistry id="saf" />
<safAuthorization id="saf" />
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

<zosConnectManager
  globalAdminGroup="GADMIN"
  globalOperationsGroup="GOPERS"
  globalInvokeGroup="GINVOKE"
  globalInterceptorsRef="interceptorList_g" />

<authorizationInterceptor id="auth" />
<auditInterceptor id="audit" />

<zosConnectInterceptors
  id="interceptorList_g"
  interceptorRef="auth,audit"/>

<zosConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />

<localAdaptersConnectService id="wolaService"
  registerName="ZOSCONN"
  serviceName="ServiceName"
  connectionFactoryRef="wolaCF" />

<zosConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  dataXformRef="xformJSON2Byte"
  serviceRef="wolaCICS" />

<zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/u/user1/bindfiles"
  bindFileSuffix=".bnd"
  requestSchemaLoc="/u/user1/json"
  responseSchemaLoc="/u/user1/json"
  requestSchemaSuffix=".json"
  responseSchemaSuffix=".json">
</zosConnectDataXform>

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />

<zosLocalAdapters
  wolaGroup="GROUP"
  wolaName2="NAME2"
  wolaName3="NAME3" />

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<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" />

<!-- To access this server from a remote client add a host -->
<httpEndpoint id="defaultHttpEndpoint"
  host="*"
  httpPort="9080"
  httpsPort="9443" />

</server>

```

Notes: This added three authorization groups to the `<zosConnectManager>` element, added the `<authorizationInterceptor>`, and added that interceptor to the list we used earlier.

Validation of global authorization interceptor

What we expect to see is the following:

Fred	Admin on both services.
Your "UserX"	Operations on both services.
Your "UserY"	Invoke on both services.

Ready your environment by starting both backend services

We need both the batch program (ZOSCONN) started and registered into the Liberty Profile Server as well as the CICS region link server task (BBOC START_SRVR) started and registered into the server.

Do the following:

- ☐ Check to see if the ZOSCONN program is started. If not, start it so it registers into the server. (See page 33 for where you did this earlier.)
- ☐ Make sure the CICS link server task is in a good state. If in doubt, enter the following commands in a CICS terminal session⁹⁷:
 - **BBOC STOP_SRVR RGN=CICSREG**
 If you see:
 WOLA TRACE 0: Stop server completed successfully.
 It means the server was already up and you just stopped it.
 However, if you see:
 CICS could not unregister with the optimized local adapters.
 The return code is 8 and reason code is 8.
 It means the server was not started. That's okay, you'll start it next.
 - **BBOC START_SRVR RGN=CICSREG DGN=GROUP NDN=NAME2 SVN=NAME3
 SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y**
 You should see:
 WOLA TRACE 0: Start server completed successfully

⁹⁷ This stops the Link Server and then restarts it. It's just a way to make sure you have a good registration into the server.

At this point you have two backend services up and ready to go.

Validate Fred's authority

Fred has administrator authority over both services. That means it should be able to exercise the REST APIs against both; and it should be able to invoke both.

Do the following:

Note: This is the same kind of activities you did back under “Invoke service using REST client” on page 34. We won't give you the full details again here; rather, we'll step through a more high-level set of instructions on what to test.

- ☐ Open the Chrome browser and go to:

`https://<host>:9443/zosConnect/services`

Accept the certificate challenge and log in with the Fred ID and password.

You should get back JSON showing *both* the batch-backend service and the CICS-backend service.

- ☐ Open another tab and start the Advanced REST client. Perform a **GET** operation with the following URL:

`https://<host>:9443/zosConnect/services/batch-backend?action=status`

That should succeed and you should see:

```
serviceStatus: "Started"
```

in the response field.

Validation: This indicates Fred has Operations over the batch-backend service.

- ☐ Perform another **GET** operation with `action=status`, but this time for the CICS-backend service:

`https://<host>:9443/zosConnect/services/CICS-backend?action=status`

That should succeed and you should see:

```
serviceStatus: "Started"
```

in the response field.

Validation: This indicates Fred has Operations over the CICS-backend service.

- ☐ Perform a **PUT** operation with the following URL and JSON payload:

`https://<host>:9443/zosConnect/services/batch-backend?action=invoke`

JSON payload:

```
{"name": "value"}
```

You should get back:

Raw	JSON	Response
Copy to clipboard Save as file		
{		
BestPlatformEver: "z/OS"		
}		

Validation: This indicates Fred has Invoke over the batch-backend service.

- ☐ Perform a **PUT** operation with the following URL and JSON payload:

`https://<host>:9443/zosConnect/services/CICS-backend?action=invoke`

JSON payload:

```
{
  "OLACB01Operation": {
    "dfhcommarea": {
      "message_data": "Hello!"
    }
  }
}
```

You should get back:



Validation: This indicates Fred has Invoke over the CICS-backend service.

- ☐ Close the browser (to clear the session Fred established).

Validate UserX's authority

The ID you chose to use as “UserX” has *operations* over both services. That means it should be able to check the status, and start or stop a service. But it should *not* be able to invoke either service.

Do the following:

- ☐ Open the Chrome browser and go to:

`https://<host>:9443/zosConnect/services`

Accept the certificate challenge and log in with your “UserX” ID and password.

- ☐ Open another tab and start the Advanced REST client. Perform a **GET** operation with the following URL:

`https://<host>:9443/zosConnect/services/batch-backend?action=status`

That should succeed and you should see:

```
serviceStatus: "Started"
```

in the response field.

Validation: This indicates the user has Operations over the batch-backend service.

- ☐ Perform another **GET** operation with the following URL:

`https://<host>:9443/zosConnect/services/CICS-backend?action=status`

That should succeed and you should see:

```
serviceStatus: "Started"
```

in the response field.

Validation: This indicates the user has Operations over the CICS-backend service.

- ☐ Perform a **PUT** operation with the following URL and JSON payload:

`https://<host>:9443/zosConnect/services/batch-backend?action=invoke`

JSON payload:

```
{ "name": "value" }
```

You should *fail* and you get back:

CWWKB0409W: User <user> is not authorized to perform the request

Validation: This is expected. The user does not have invoke authority.

- ❑ Perform a **PUT** operation with the following URL and JSON payload:

`https://<host>:9443/zosConnect/services/CICS-backend?action=invoke`

JSON payload:

```
{
  "OLACB01Operation": {
    "dfhcommarea": {
      "message_data": "Hello!"
    }
  }
}
```

That should *fail* and you should see:

CWWKB0409W: User <user> is not authorized to perform the request

Validation: This is expected. The user not have invoke authority.

- ❑ Close the browser (to clear the session this user established).

Validate UserY's authority

The ID you chose to use as “UserY” has global invoke. That means it may invoke both, but it may not perform operations actions against either.

Do the following:

- ❑ Open the Chrome browser and go to:

`https://<host>:9443/zosConnect/services`

Accept the certificate challenge and log in with the UserY ID and password.

You will get an error message with “CWWKB0409W: User <user> is not authorized to perform the request.” That’s expected: the authority for this user is global *invoke*, and the URL you issued is an operations message.

But it served our purpose to get you authenticated.

- ❑ Open another tab and start the Advanced REST client. Perform a **GET** operation with the following URL:

`https://<host>:9443/zosConnect/services/batch-backend?action=status`

That should *fail* and you should see:

CWWKB0409W: User <user> is not authorized to perform the request

Validation: This is expected. The function `action=status` is a *operations* function, and your user has *invoke*, not *operations*.

- ❑ Perform another **GET** operation with the following URL:

`https://<host>:9080/zosConnect/services/CICS-backend?action=status`

That should *fail* and you should see:

CWWKB0409W: User <user> is not authorized to perform the request

Validation: This is expected. The function `action=status` is a *operations* function, and your user has *invoke*, not *operations*.

- ☐ Perform a **PUT** operation with the following URL and JSON payload:

`https://<host>:9443/zosConnect/services/batch-backend?action=invoke`

JSON payload:

```
{"name": "value"}
```

That should *work* and you should see:

Raw	JSON	Response
Copy to clipboard Save as file		
		<pre>{ BestPlatformEver: "z/OS" }</pre>

Validation: This is expected. This user has global invoke privileges.

- ☐ Perform a **PUT** operation with the following URL and JSON payload:

`https://<host>:9443/zosConnect/services/CICS-backend?action=invoke`

JSON payload:

```
{
  "OLACB01Operation": {
    "dfhcommarea": {
      "message_data": "Hello!"
    }
  }
}
```

You should *work* and get back:

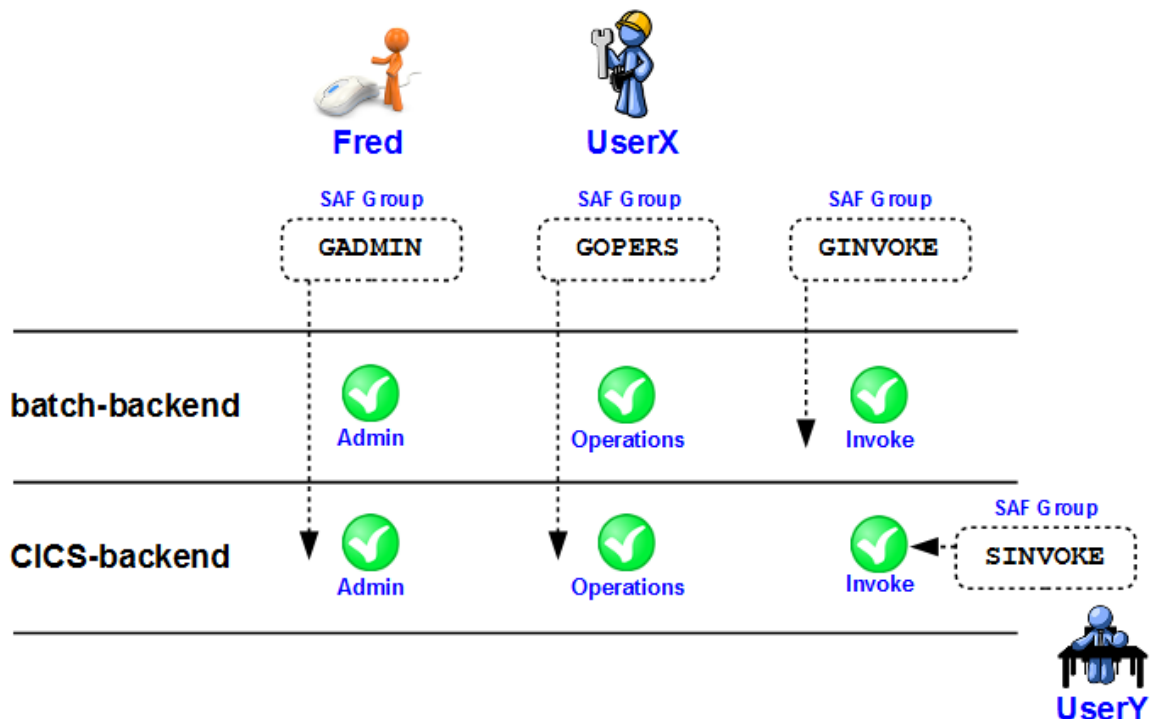
Raw	JSON	Response
Copy to clipboard Save as file		
		<pre>{ -OLACB01OperationResponse: { -dfhcommarea: { message_data: "Hello!" } } }</pre>

Validation: This is expected. This user has global invoke privileges.

- ☐ Close the browser (to clear the session this user established).

Service-level authorization

We will now introduce a service-level authorization interceptor. We'll move UserY down to that level so that ID has only *invoke* on the CICS-backend service:



What we expect from this is the following:

- Fred should still have Admin over both services. That means Fred should be able to see both services, start and stop both, and invoke both.
- UserX should have Operations over both services. That means it should be able to perform operations actions (start, stop, status) but *not* invoke the services.
- UserY should have Invoke over only the CICS-backend service. UserY should *not* be able to invoke the batch-backend service, nor perform any operations functions.

Note: What about a user connected to GINVOKE? It will have invoke authority over the batch-backend service, but not the CICS-backend service. The introduction of a service-level authorization for the CICS-backend service means the global authority is superseded by the service-level authority.

Do the following:

- ☐ Remove your UserY from the GINVOKE group and connect it to the SINVOKe group.
- ☐ Change the `server.xml` with the following updates (highlighted):

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <safRegistry id="saf" />
  <safAuthorization id="saf" />
  <safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />
</server>
```

```

<zosConnectManager
  globalAdminGroup="GADMIN"
  globalOperationsGroup="GOPERS"
  globalInvokeGroup="GINVOKE"
  globalInterceptorsRef="interceptorList_g" />

<authorizationInterceptor id="auth" />
<auditInterceptor id="audit" />

<zosConnectInterceptors
  id="interceptorList_g"
  interceptorRef="auth,audit"/>

<zosConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />

<localAdaptersConnectService id="wolaService"
  registerName="ZOSCONN"
  serviceName="ServiceName"
  connectionFactoryRef="wolaCF" />

<zosConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  dataXformRef="xformJSON2Byte"
  invokeGroup="SINVOKE"
  serviceRef="wolaCICS" />

<zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/u/user1/bindfiles"
  bindFileSuffix=".bnd"
  requestSchemaLoc="/u/user1/json"
  responseSchemaLoc="/u/user1/json"
  requestSchemaSuffix=".json"
  responseSchemaSuffix=".json">
</zosConnectDataXform>

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />

<zosLocalAdapters
  wolaGroup="GROUP"
  wolaName2="NAME2"
  wolaName3="NAME3" />

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<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" />

```

```
<!-- To access this server from a remote client add a host -->
<httpEndpoint id="defaultHttpEndpoint"
             host="*"
             httpPort="9080"
             httpsPort="9443" />
```

```
</server>
```

- ☐ Go through the same set of validations you went through for the global authorization interceptors:

☐	The ID "Fred" should have administrative authority over both services. Fred should be able to see both services, perform operations actions on both, and invoke both.
☐	Your UserX ID, with connection to the <code>GOPERS</code> group, should have operations authority over both services. But it should not be able to invoke either.
☐	Your UserY ID, now connected to the <code>SINVOKE</code> group at the service level, should have invoke authority over the CICS-backend service, but <i>not</i> the batch-backend service. That's the key distinction – we have narrowed UserY's authority to invoke over just that one service.

- ☐ (Optional) Exercise the audit validation steps again to verify that SMF 120.11 records are indeed still being cut for the CICS-backend service.

Configuration checkpoint: server.xml at this point

See "Checkpoint 3: after authorization interceptor configuration" on page 84 for a sample of the `server.xml` at this point in the flow of this document.

Use SAF and client certificates

Up to this point in the document we have operated with basic authentication and a prompt that appears on your browser. You've entered Fred for most of the steps. Fred was allowed access because Fred had `READ` to the `zosConnectAccess EJBROLE`.

The *default* method of authentication for z/OS Connect is *not* basic authentication. The default mechanism is to use *client certificates*. But rather than complicate things early with client certs, we overrode that default with the following bit of XML:

```
<webAppSecurity allowFailOverToBasicAuth="true" />
```

That told z/OS Connect to prompt the user with a basic authentication prompt. That's what you saw throughout the exercises each time the prompt appeared asking for userid and password.

In this section we will show how to use include a client certificate with your browser⁹⁸ so that the certificate is presented as authentication.

Create client certificate and export certificates from SAF

Here we will illustrate creating a client certificate for the user "Fred." Work with your security administrator and do the following:

- ☐ Issue the following command (one long line) to create a client certificate for Fred:

```
RACDCERT ID(FRED) GENCERT SUBJECTSDN(CN('Fred D. Client')
                                     O('IBM') OU('LIBERTY')) WITHLABEL('FRED')
                                     SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY')) SIZE(2048)
                                     NOTAFTER(DATE(2018/12/30))
```

- ☐ Export that client certificate to a data set:

```
RACDCERT ID(FRED) EXPORT(LABEL('FRED'))
```

⁹⁸ In this section we will illustrate how to do this with the Google Chrome browser, since that's the browser that has the Advanced REST client.

```
DSN (' yourid.FRED.P12' ) FORMAT (PKCS12DER)
PASSWORD (' password' )
```

Where:

- *yourid* is the ID used when entering this command. It's really just a way to qualify the data set that will contain the exported certificate.
- *password* is the password used to encrypt the client certificate. This password will be needed when importing the client certificate into the workstation. Do not send this password in the same e-mail as the certificate itself. Password is *case sensitive*.

- ☐ Export the CA certificate you created earlier:

```
RACDCERT CERTAUTH EXPORT (LABEL ('LibertyCA.LIBERTY'))
DSN (' yourid.CERTAUTH.CRT' ) FORMAT (CERTDER)
```

Where:

- *yourid* is the ID used when entering this command. It's really just a way to qualify the data set that will contain the exported certificate.

The client certificate for Fred is created and exported to a data set. The CA certificate has been exported as well. The next step is to download those certificates to your workstation and install them.

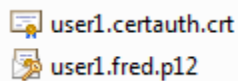
Download and import CA certificate and client certificate into workstation

Do the following:

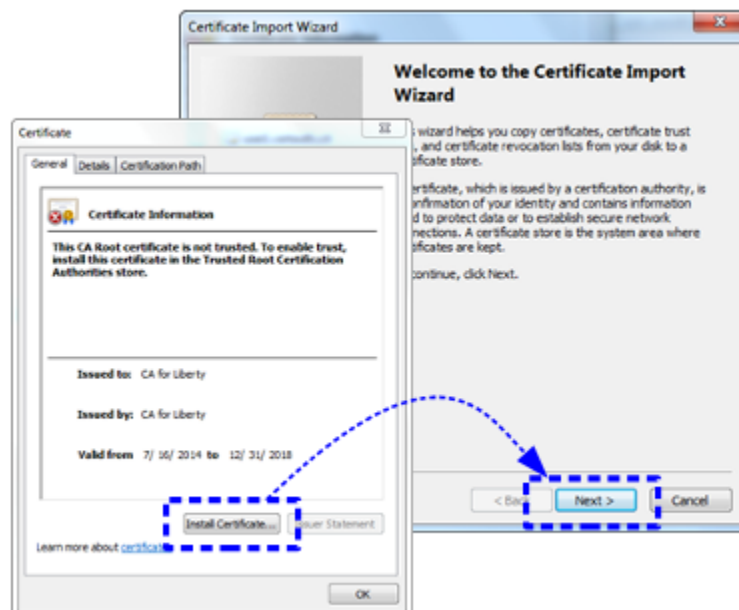
- ☐ FTP the client certificate (' *yourid*.FRED.P12') *in binary* to your workstation.
- ☐ FTP the CA certificate (' *yourid*.CERTAUTH.CRT') *in binary* to your workstation.

The instructions that follow assumes Windows 7 and the Chrome Browser. It also assumes you have the authority to install certificates to your workstation.

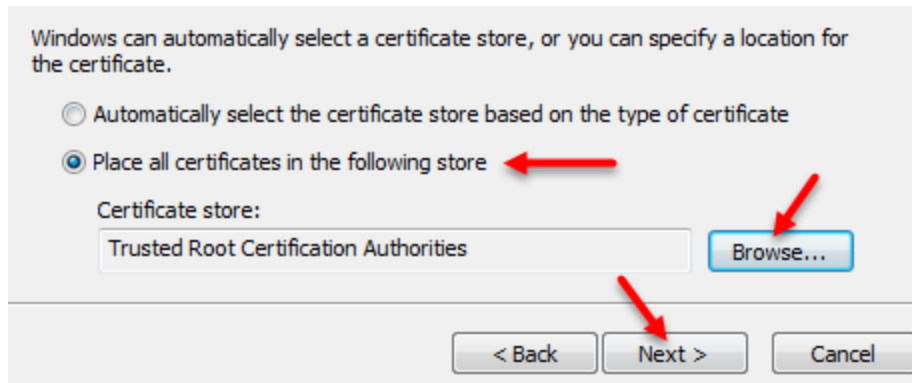
- ☐ Locate the folder on the workstation where the downloaded files reside:



- ☐ Double-click on the CA certificate (the one ending with 'crt'). That will bring up the Windows certificate. Click on "Install Certificate," then "Next":

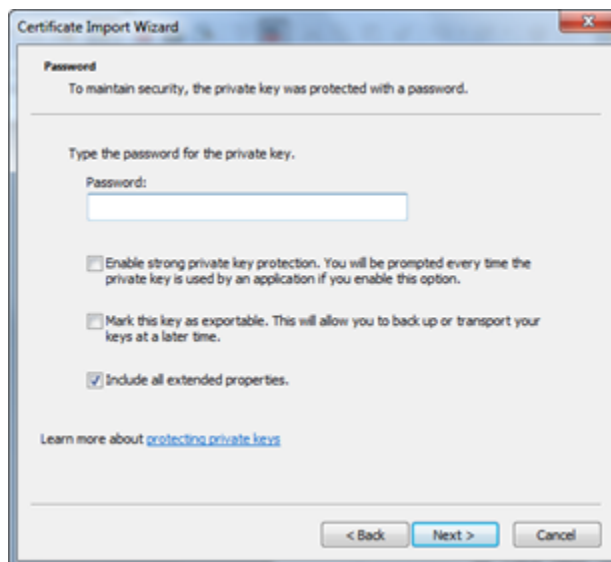


- To make sure this CA gets to the right place, do the following:



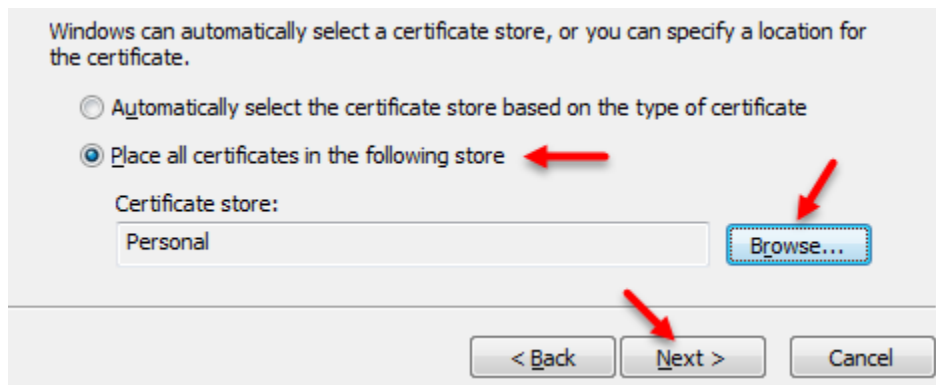
Click the “Place all certificates in the following store” radio button, then click the “Browse” button and select “Trusted Root Certification Authorities”. When that shows in the window, click “Next”.

- Click “Finish,” then read and click “Yes” to the security warning. Click “OK” on the confirmation panel and close out the wizard.
- Go back to the folder with the certificates and double-click on the client certificate (the one that ends in 'p12'). This opens the wizard for a second certificate install.
- Click “Next” on the welcome panel.
- Click “Next” on the “File to import” panel.
- The wizard will now present a prompt for you to enter the password used to encrypt the client certificate:



This is the same password used on the EXPORT command earlier for Fred's client certificate. The password is case sensitive, so enter it exactly as was used on the SAF export command. Then click “Next.”

- As with the CA certificate, tell Windows where you want this certificate placed:



Click the radio button to specify the location of the store, then click the browse button and select “Personal,” then click the next button.

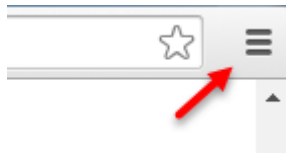
- ☐ Click “Finish” and then “OK” at the confirmation and close the wizard.

Validate browser recognizes the certificates

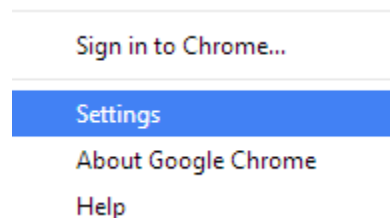
We’re going to take a quick tour of what Chrome sees as the state of the certificates. This will be a preliminary validation of the import of the certs.

Do the following:

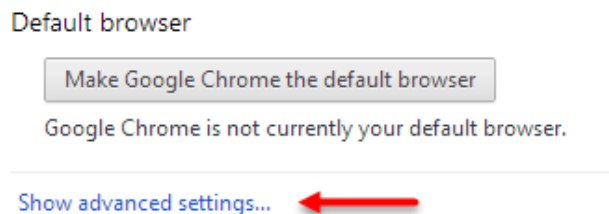
- ☐ Open the Chrome browser.
- ☐ In the upper-right, click on the three-bar icon:



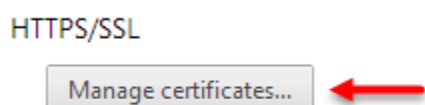
Then select “Settings”:



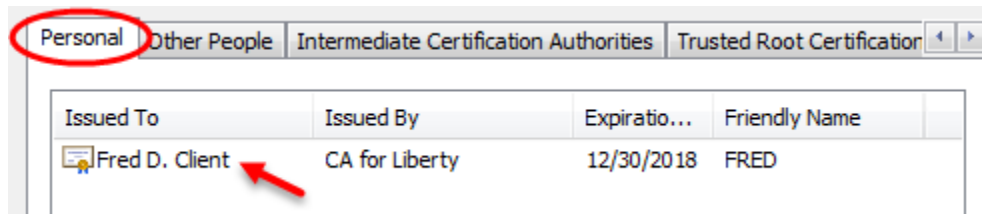
- ☐ Scroll down and click on the “Show advanced settings” link:



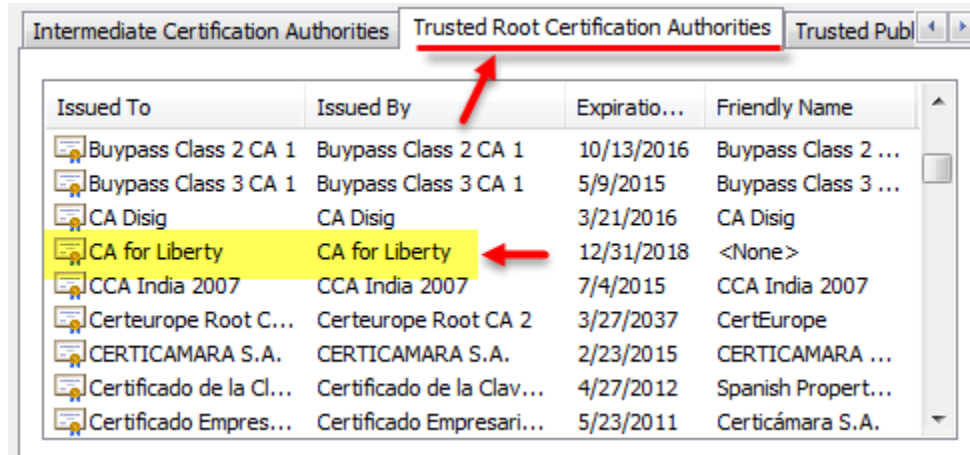
- ☐ Scroll down and click on the “Manage certificates” button:



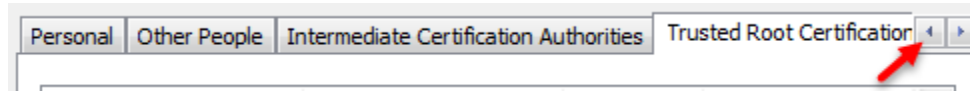
- ☐ You should see the Fred certificate in the “Personal” tab:



- Click on the “Trusted Root Certification Authorities” tab. Scroll down and look for the “CA for Liberty”:



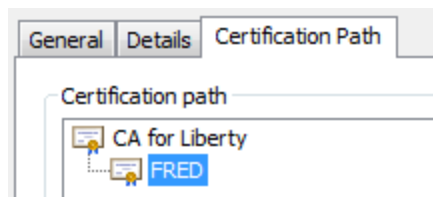
- Go back to the “Personal” tab by clicking on the little left arrow in the upper-right:



- Double-click on the Fred's certificate under the Personal tab, then click on the “Certificate Path” tab in the window that comes up:



- You should see Fred's certificate under the CA certificate:



That indicates the client certificate and the CA that signed the client certificate are both in the registry and recognized as related. This is the visual validation we were after before we moved on to testing the client certificate with z/OS Connect.

Update server.xml to prompt for certificate

A relatively simple change to the server.xml is needed to tell Liberty to prompt for client certificates first. Do the following:

- Edit your `server.xml` and locate the `<sslDefault>` element near the bottom:

```
<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" />

```

- Add the **highlighted** lines and remove the **strikethrough** characters:

```

<sslDefault sslRef="DefaultSSLSettings" />
  <ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultTrustStore" ➔
    clientAuthenticationSupported="true"
    clientAuthentication="false"/>
  <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" />

```

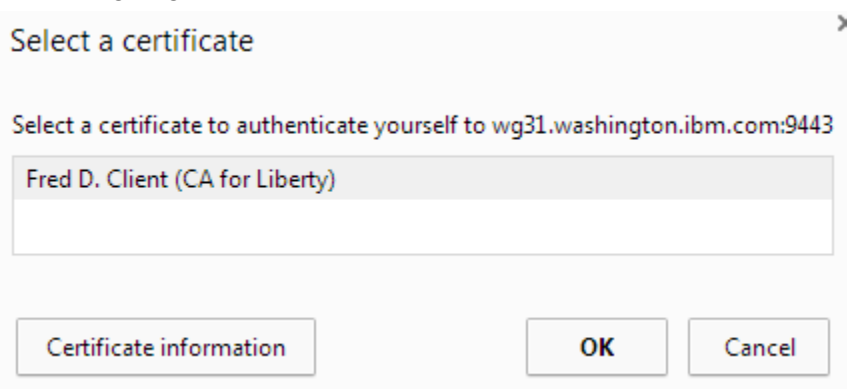
- Save the file. Liberty Profile will pick up the change dynamically.

Validate use of certificate to authenticate into z/OS Connect

- In your Chrome browser, enter this URL:

https://<host>:9443/zosConnect/services

Because of the `clientAuthenticationSupport="true"` line in the XML, the server is going to ask the browser to prompt for the certificate. You should see:



Select "Fred" and click "OK".

- You should then see the familiar JSON list of services.

That JSON indicates the client certificate satisfied the requirement for authentication. The certificate represented the ID of Fred, and since Fred has READ to the EJBROLE for zosConnectAccess, Fred was allowed in the front door. Because Fred has global admin from the previous exercise, he was allowed to see both services.

- Close your browser, reopen it, and then enter the same URL again. But this time, click "Cancel" on the prompt for Fred's client certificate. You should get the basic authentication prompt⁹⁹:

⁹⁹ If you don't, it may be due to the way Google Chrome works. Try Ctrl-Shift-N (for "New Incognito Window").

Authentication Required

The server <https://wg31.washington.ibm.com:9443> requires a username and password. The server says: zosConnect.

User Name:

Password:

Enter Fred's ID and password. You will see the same JSON as before. This is because the Liberty Profile server is set up to allow basic authentication for cases where client certificates don't exist.

- ❑ Let's change it to *require* client certificates. Edit your `server.xml` and make to changes:

Near the top, delete the "allowFailOverToBasicAuth" line:

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

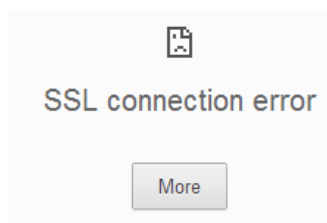
  <webAppSecurity allowFailOverToBasicAuth="true"/>

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

Then down in the SSL section you updated a few moments ago, change the `clientAuthentication=` line from `false` to **true**

```
<sslDefault sslRef="DefaultSSLSettings" />
  <ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultTrustStore"
    clientAuthenticationSupported="true"
    clientAuthentication="true"/>
  <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" />
```

- ❑ Save the file.
- ❑ Stop and restart your browser.
- ❑ Enter the same URL again. When prompted for the client certificate, click "Cancel". You should see:



Without a client certificate no SSL connection was allowed. No identity was passed.
No entry to z/OS Connect was granted.

Configuration checkpoint: server.xml at this point

See "Checkpoint 4: after client certificates for authentication" on page 85 for a sample of the `server.xml` at this point in the flow of this document.

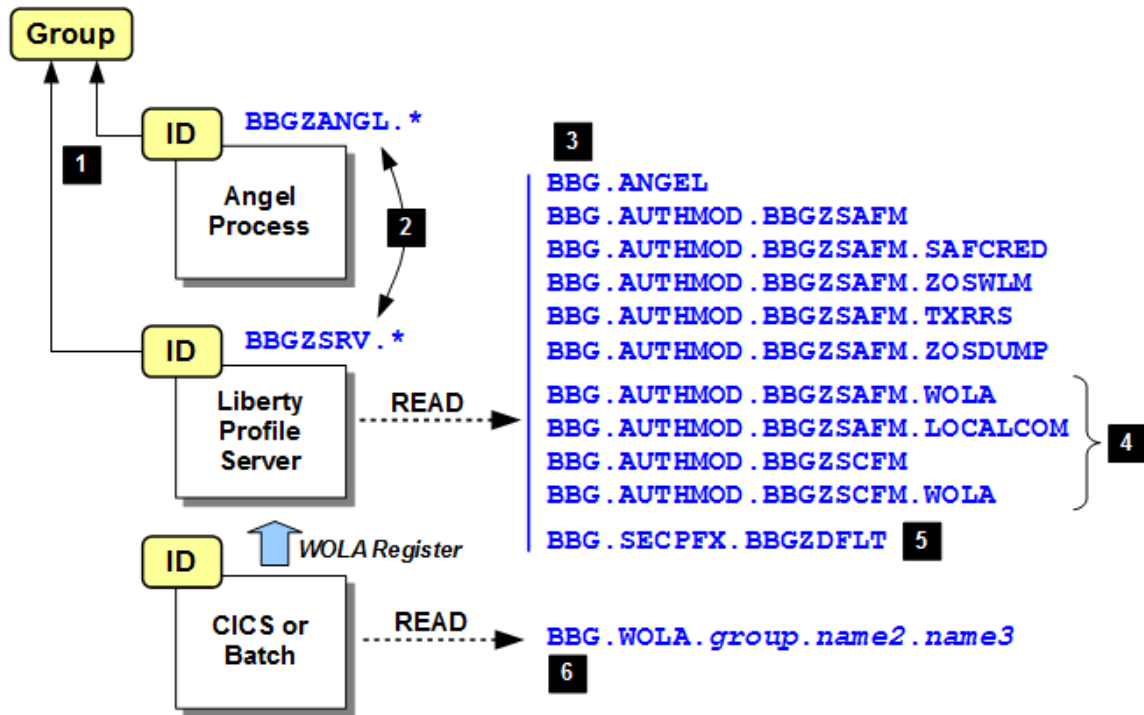
Miscellaneous Information

This section will serve as a gathering point for miscellaneous information we found useful, but chose not to include inline with the step-by-step discussion earlier.

SAF definitions

Overview of the SAF security model

What will follow is a long set of SAF commands to create users, groups, and profiles. To make sense of those details, here's an overview picture of the SAF security model¹⁰⁰:



The following numbers correspond to the blocks in the diagram:

1. The Angel process and the Liberty Profile server will each have its own ID, separate from one another¹⁰¹. Both may connect to a common group.
2. SAF *STARTED* profiles are used to assign the IDs to the respective started tasks: the Angel process ID to the Angel started task; and the Liberty Profile server ID to the server started task.
3. A collection of SAF *SERVER* profiles are used to control access to various z/OS related services. A brief description of what each profile is provided under “SERVER profiles” on page 77.
4. The four *SERVER* profiles highlighted here are new in 8.5.5.2¹⁰² and relate to the WebSphere Optimized Local Adapter support. z/OS Connect uses WOLA to access CICS and Batch.
5. A *SERVER* profile to control access to *EJBROLE* definitions based on the default security prefix of *BBGZDFLT*. This is used for the authentication interceptor function.
6. A *CBIND* profile is used to control who may “register in” to a Liberty Profile server using WOLA. Each Liberty Profile server will carry a unique *group.name2.name3* sequence. *READ* to the *CBIND* that specifies the *group.name2.name3* sequence provides access. This *CBIND* may be wild-carded so one *CBIND* permits access to many different server *group.name2.name3* sequences.

¹⁰⁰Not including the *EJBROLE* profiles. See “*EJBROLE* profiles to control access” on page 78 for that.

¹⁰¹Having the Angel process ID separate from the server ID is not a technical requirement. It is however a good practice.

¹⁰²The first six in the picture are original to Liberty Profile z/OS, which became available with 8.5.0.0.

ID and Group

Creates a Liberty Profile group ID

```
ADDGROUP lib_group OMVS (GID(gid)) OWNER(SYS1)
```

Creates the Angel ID and connects it to the Liberty Profile group

```
ADDUSER angel_id DFLTGRP(lib_group) OMVS(UID(uid))
      HOME(angel_home) PROGRAM(/bin/sh)) NAME('Liberty Angel')
      OWNER(lib_group) NOPASSWORD NOOIDCARD
```

Creates the Liberty Profile server ID and connects it to the Liberty Profile group

```
ADDUSER liberty_id DFLTGRP(lib_group) OMVS(UID(uid))
      HOME(server_home) PROGRAM(/bin/sh)) NAME('Liberty Server')
      OWNER(lib_group) NOPASSWORD NOOIDCARD
```

STARTED for Liberty Profile Angel process and server

Creates the STARTED profile for the Angel Process

```
RDEF STARTED BBGZANGL.* UACC(NONE) STDATA(USER(angel_id)
      GROUP(lib_group) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

Creates the STARTED profile for the Liberty Profile server

```
RDEF STARTED BBGZSRV.* UACC(NONE) STDATA(USER(liberty_id)
      GROUP(lib_group) PRIVILEGED(NO) TRUSTED(NO) TRACE(YES))
```

Refreshes the STARTED class profiles

```
SETROPTS RACLIST(STARTED) REFRESH
```

SERVER profiles

Grants an ID general access to the Angel process for authorized services

```
RDEF SERVER BBG.ANGEL UACC(NONE) OWNER(SYS1)
PERMIT BBG.ANGEL CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use the BBGZSAFM authorized module in the Angel process

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSAFM for SAF services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.SAFCREDS UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCREDS CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSAFM for WLM services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSWLM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSWLM CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSAFM for RRS services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.TXRRS UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.TXRRS CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSAFM for z/OS Dump services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.ZOSDUMP UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.ZOSDUMP CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSAFM for WOLA services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.WOLA CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSAFM for LOCALCOM services

```
RDEF SERVER BBG.AUTHMOD.BBGZSAFM.LOCALCOM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.LOCALCOM CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use the BBGZSCFM authorized module in the Angel process

```
RDEF SERVER BBG.AUTHMOD.BBGZSCFM UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSCFM CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls which server processes can use BBGZSCFM for WOLA

```
RDEF SERVER BBG.AUTHMOD.BBGZSCFM.WOLA UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSCFM.WOLA CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Controls access to EJBROLE definitions based on the prefix in use for a server

```
RDEF SERVER BBG.SECPFX.BBGZDFLT UACC(NONE) OWNER(SYS1)
PERMIT SERVER BBG.SECPFX.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```

Refreshes the SERVER class profiles

```
SETROPTS RACLIST(SERVER) REFRESH
```

CBIND for registration into server

Creates the CBIND profile to control who may register into the Liberty Profile server using WOLA

```
RDEF CBIND BBG.WOLA.group.name2.name3 UACC(NONE) OWNER(SYS1)
PERMIT BBG.WOLA.group.name2.name3 CLASS(CBIND) ACCESS(READ) ID(accessing_id)
```

Refreshes the CBIND class profiles

```
SETROPTS RACLIST(CBIND) REFRESH
```

SAF CA Certificate and Server Certificate signed by CA

Creates the CA certificate that will be used to sign the server certificate

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty')
                                         OU('LIBERTY')) WITHLABEL('LibertyCA.LIBERTY')
                                         TRUST NOTAFTER(DATE(2018/12/31))
```

Creates a keyring that will be associated with the Liberty Profile server ID

```
RACDCERT ADDRING(Keyring.LIBERTY) ID(liberty_id)
```

Creates the server certificate and signs it with the CA certificate created earlier

```
RACDCERT ID(liberty_id) GENCERT SUBJECTSDN(CN('yourhost.com')
                                             O('IBM') OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY')
                                             SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY'))
                                             NOTAFTER(DATE(2018/12/31))
```

Connects the server certificate to the keyring associated with the Liberty Profile server ID

```
RACDCERT ID(liberty_id) CONNECT (LABEL('DefaultCert.LIBERTY')
                                  RING(Keyring.LIBERTY) DEFAULT)
```

Connects the CA certificate to the keyring associated with the Liberty Profile server ID

```
RACDCERT ID(liberty_id) CONNECT (LABEL('LibertyCA.LIBERTY')
                                  RING(Keyring.LIBERTY) CERTAUTH)
```

Refreshes the FACILITY class profiles

```
SETROPTS RACLIST(FACILITY) REFRESH
```

APPL class profiles and the unauthenticated userid and group

Created APPL class profile based on security profile prefix of Liberty Profile server

```
RDEFINE APPL BBGZDFLT UACC(NONE) OWNER(SYS1)
```

Creates a group ID for the unauthenticated user

```
ADDGROUP unauthgrp OMVS(GID(gid)) OWNER(SYS1)
```

Creates the unauthenticated user ID and connects it to the unauthenticated user group

```
ADDUSER unauth_ID DFLTGRP(unauthgrp) OMVS(UID(uid)
                                             HOME(unauthID_home) PROGRAM(/bin/sh)) NAME('Unauthenticated User')
                                             OWNER(lib_group) NOPASSWORD NOOIDCARD RESTRICTED
```

Grants READ to the unauthenticated userid

```
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(unauth_ID)
```

Grants READ to any other ID that has authenticated

```
PERMIT BBGZDFLT CLASS(APPL) ACCESS(READ) ID(*)
```

Refreshes the FACILITY class profiles

```
SETR RACLIST(APPL) REFRESH
```

EJBROLE profiles to control access

Creates the EJBROLE used to control access to the z/OS Connect application itself

```
RDEFINE EJBROLE BBGZDFLT.zos.connect.access.roles.zosConnectAccess
                                             OWNER(SYS1) UACC(NONE)
```

Refreshes the FACILITY class profiles

```
PERMIT BBGZDFLT.zos.connect.access.roles.zosConnectAccess
                                CLASS(EJBROLE) ID(userid_or_group) ACCESS(READ)
```

Refreshes the EJBROLE class profiles

```
SETR RACLIST(EJBROLE) REFRESH
```

CA certificate and the Server certificate*Creates a Certificate Authority (CA) certificate used to sign the server certificate*

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty')
                                      OU('LIBERTY')) WITHLABEL('LibertyCA.LIBERTY')
                                      TRUST NOTAFTER(DATE(2018/12/31))
```

Creates a SAF keyring that is assigned to the ID for the Liberty Profile server

```
RACDCERT ADDRING(Keyring.LIBERTY) ID(liberty_id)
```

Creates a server certificate and signs it with the CA certificate

```
RACDCERT ID(liberty_id) GENCERT SUBJECTSDN(CN('yourhost.com')
                                      O('IBM') OU('LIBERTY')) WITHLABEL('DefaultCert.LIBERTY')
                                      SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY'))
                                      NOTAFTER(DATE(2018/12/31))
```

Connects the server certificate to the keyring associated with the Liberty Profile server ID

```
RACDCERT ID(liberty_id) CONNECT (LABEL('DefaultCert.LIBERTY')
                                RING(Keyring.LIBERTY) DEFAULT)
```

Connects the CA certificate to the keyring associated with the Liberty Profile server ID

```
RACDCERT ID(liberty_id) CONNECT (RING(Keyring.LIBERTY)
                                LABEL('LibertyCA.LIBERTY') CERTAUTH)
```

Refreshes the FACILITY class profiles

```
SETOPTS RACLIST(FACILITY) REFRESH
```

Client certificate and exporting certificates for download to workstation*Creates a client certificate for the ID of FRED*

```
RACDCERT ID(FRED) GENCERT SUBJECTSDN(CN('Fred D. Client')
                                      O('IBM') OU('LIBERTY')) WITHLABEL('FRED')
                                      SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY')) SIZE(2048)
                                      NOTAFTER(DATE(2018/12/30))
```

Exports the client certificate to a data set

```
RACDCERT ID(FRED) EXPORT(LABEL('FRED'))
                                DSN('yourid.FRED.P12') FORMAT(PKCS12DER) PASSWORD('password')
```

Exports the CA certificate to a data set

```
RACDCERT CERTAUTH EXPORT(LABEL('LibertyCA.LIBERTY'))
                                DSN('yourid.CERTAUTH.CRT') FORMAT(CERTDER)
```

The security profile prefix value

The security profile prefix for Liberty Profile defaults to BBGZDFLT. However, when put in place the authorization interceptors, we showed how to specify in the `server.xml` the security profile prefix this instance of Liberty Profile would use:

```
<safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />
```

This default value showed up in a few SAF profiles we created. For example:

```
RDEFINE SERVER BBG.SECPFX.BBGZDFLT ...
```

and:

```
RDEFINE APPL BBGZDFLT UACC(NONE) ...
```

and:

```
RDEFINE EJBROLE BBGZDFLT.zos.connect.access.roles.zosConnectAccess ...
```

By showing you how to code the `profilePrefix=` value we opened the door to coding a value different from the default of BBGZDFLT. The value in that is it allows different

instances of Liberty Profile z/OS (and z/OS Connect running inside) to have different SAF profiles controlling access.

For example, the `zosConnectAccess` `EJBROLE` has the profile prefix at the beginning. If you had three instances of Liberty Profile z/OS running – one for development, one for test and one for production – and all three used the same prefix, then any ID granted `READ` to that `EJBROLE` would have access to all three instances of Liberty Profile z/OS.

Suppose you wanted to keep them separate? Then you would have each instance of Liberty Profile z/OS specify a different prefix value. Development might use a value of `BBGZDEV`, test a value of `BBGZTEST` and production a value of `BBGZPROD`. With that in place, then you could create three separate `EJBROLE` definitions:

```
RDEFINE EJBROLE BBGZDEV.zos.connect.access.roles.zosConnectAccess ...
RDEFINE EJBROLE BBGZTEST.zos.connect.access.roles.zosConnectAccess ...
RDEFINE EJBROLE BBGZPROD.zos.connect.access.roles.zosConnectAccess ...
```

And now you can control access more granularly. If Fred is a developer, then you grant `READ` to Fred for the `BBGZDEV` `EJBROLE`. Fred would then have access to the development instance of Liberty Profile and z/OS Connect, but not the test or the production instances.

The other SAF profiles (`SERVER` and `APPL`), so plan appropriately when using unique profile prefix values. For now, understand it is possible. The value is the more granular access it affords.

Workaround to do basic authentication using local repository

Back on page 29 the instructions had you update the `server.xml` with information to provide basic authentication against a “registry” that was defined in the `server.xml`, and not require any checking against SAF.

And that works perfectly well ... *unless* you have the SAF `SERVER` class profile for `SAFCRED` defined and your server ID has `READ` to it. If so, then you must do two more things¹⁰³:

1. Make sure whatever ID you have coded in `server.xml` is also in SAF. The ID in SAF can be a dummy copy with no password¹⁰⁴ ... but the ID must exist.
2. A `SERVER` class profile for the default security prefix needs to be in place.

Here's the workaround. Do the following:

- ☐ Determine if the ID `FRED` already exists in SAF. If so, then that requirement is met. Again, the password in the `server.xml` does *not* need to match the password for `FRED` in SAF. The dummy password of “fredpwd” in the XML is fine.
- ☐ If no `FRED` ID already exists, then you may either (a) create it, or (b) choose an ID that does already exist. Any ID will do. Once you have chosen that ID, code *that* ID in the `server.xml` back on page 29 rather than “Fred”. Then use that ID for the basic authentication. There are two instances of “Fred” in that XML that would be replaced with whatever ID you choose for this.
- ☐ Add the following `SERVER` class profile:


```
RDEFINE SERVER BBG.SECPFY.BBGZDFLT UACC(NONE)
```
- ☐ Grant the ID under which your Liberty Profile server (see page 8) runs `READ` access to the profile:


```
PERMIT BBG.SECPFY.BBGZDFLT CLASS(SERVER) ACCESS(READ) ID(liberty_id)
```
- ☐ Refresh the `SERVER` class:

¹⁰³The cause of this behavior is being investigated. The instructions here provide a workaround.

¹⁰⁴The password coded in the `server.xml` basic registry is what's used when you are prompted with the basic authentication panel.

SETOPTS RACLIST(SERVER) REFRESH

- Stop and restart your Liberty Profile server.

Angel process and the new WOLA support

Liberty Profile z/OS and the Angel process was first made available with WAS z/OS V8.5.0.0. z/OS Connect and WOLA support was first made available with WAS z/OS 8.5.5.2.

The Angel process was designed to be started and left running essentially forever. Only one Angel process is needed per z/OS LPAR¹⁰⁵ regardless of how many Liberty Profile server instances are on the LPAR. Even then the Angel is only *required* for certain things¹⁰⁶. It was designed to rarely, if ever, require a restart.

However, the new WOLA support of Liberty Profile z/OS requires the 8.5.5.2 level of code be loaded with the instance of the Angel process. That implies a stop and restart of Angel process so it can pick up and load the 8.5.5.2 code with the WOLA support.

Back on page 13 we had you copy the Angel JCL start procedure and modify the `SET ROOT=` value to point to your `INSTALL_DIR`, which would contain the 8.5.5.2 code. Then on page 14 we had you start the Angel process. We never again had you stop the Angel. There was no need – once started with the 8.5.5.2 code, the Angel can remain up.

Key Point: If you have a z/OS LPAR environment where an Angel process is running with some level of code prior to that needed by the WOLA support, you will need to restart the Angel at the new level. You can check to see what level of code an instance of the Angel process is running by issuing the following command:

F `<jobname>,VERSION`

where `<jobname>` is the job name of the Angel started task. The WOLA support is indicated with the following message:

CWWKB0053I ANGEL VERSION 2

If you see this:

CWWKB0053I ANGEL VERSION 1

then you're back level. You would need to stop your Angel, change the `SET ROOT=` pointer in the JCL start procedure to point to where 8.5.5.2 is installed, then restart the Angel process. Then issue the `MODIFY VERSION` command again to verify you're at the proper level for WOLA.

¹⁰⁵ If you try to start a second, the second instance will fail to come up.

¹⁰⁶ The WOLA support being one such thing that requires the Angel. The JDBC Type 2 support using RRS is another. The z/OS `MODIFY` command to process dump actions is a third.

Reference: server.xml examples

In this section we will offer complete `server.xml` examples from key checkpoints in the flow of this document. These files are also included in the ZIP file that accompanies the Techdoc¹⁰⁷.

Checkpoint 1: End of WOLA Enablement

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>zosLocalAdapters-1.0</feature>
  </featureManager>

  <zosLocalAdapters
    wolaGroup="GROUP"
    wolaName2="NAME2"
    wolaName3="NAME3" />

  <connectionFactory id="wolaCF" jndiName="eis/ola">
    <properties.ola />
  </connectionFactory>

  <!-- To access this server from a remote client -->
  <httpEndpoint id="defaultHttpEndpoint"
    host="*"
    httpPort="9080"
    httpsPort="9443" />

</server>
```

Checkpoint 2: After audit interceptor configuration

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

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
  </featureManager>

  <keyStore id="defaultKeyStore" password="Liberty"/>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <basicRegistry id="basic1" realm="zosConnect">
    <user name="Fred" password="fredpwd" />
  </basicRegistry>

  <authorization-roles id="zos.connect.access.roles">
    <security-role name="zosConnectAccess">
      <user name="Fred"/>
    </security-role>
  </authorization-roles>

  <zosConnectManager
    globalInterceptorsRef="interceptorList_g" />

</server>
```

¹⁰⁷<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102439>

```

<auditInterceptor id="audit" />

<zosConnectInterceptors
  id="interceptorList_g"
  interceptorRef="audit"/>

<zosConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />

<localAdaptersConnectService id="wolaService"
  registerName="ZOSCONN"
  serviceName="ServiceName"
  connectionFactoryRef="wolaCF" />

<zosConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  dataXformRef="xformJSON2Byte"
  serviceRef="wolaCICS" />

<zosConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/u/user1/bindfiles"
  bindFileSuffix=".bnd"
  requestSchemaLoc="/u/user1/json"
  responseSchemaLoc="/u/user1/json"
  requestSchemaSuffix=".json"
  responseSchemaSuffix=".json">
</zosConnectDataXform>

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />

<zosLocalAdapters
  wolaGroup="GROUP"
  wolaName2="NAME2"
  wolaName3="NAME3" />

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<!-- To access this server from a remote client add a host -->
<httpEndpoint id="defaultHttpEndpoint"
  host="*"
  httpPort="9080"
  httpsPort="9443" />

</server>

```

Checkpoint 3: after authorization interceptor configuration

```

<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zosConnect-1.0</feature>
    <feature>zosLocalAdapters-1.0</feature>
    <feature>zosSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

  <webAppSecurity allowFailOverToBasicAuth="true" />

  <safRegistry id="saf" />
  <safAuthorization id="saf" />
  <safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

  <zosConnectManager
    globalAdminGroup="GADMIN"
    globalOperationsGroup="GOPERS"
    globalInvokeGroup="GINVOKE"
    globalInterceptorsRef="interceptorList_g" />

  <authorizationInterceptor id="auth" />
  <auditInterceptor id="audit" />

  <zosConnectInterceptors
    id="interceptorList_g"
    interceptorRef="auth,audit"/>

  <zosConnectService id="zosconn"
    invokeURI="/myBatchBackend"
    serviceName="batch-backend"
    serviceRef="wolaService" />

  <localAdaptersConnectService id="wolaService"
    registerName="ZOSCONN"
    serviceName="ServiceName"
    connectionFactoryRef="wolaCF" />

  <zosConnectService id="CICS"
    invokeURI="/myCICSBackend"
    serviceName="CICS-backend"
    dataXformRef="xformJSON2Byte"
    invokeGroup="SINVOKE"
    serviceRef="wolaCICS" />

  <zosConnectDataXform id="xformJSON2Byte"
    bindFileLoc="/u/user1/bindfiles"
    bindFileSuffix=".bnd"
    requestSchemaLoc="/u/user1/json"
    responseSchemaLoc="/u/user1/json"
    requestSchemaSuffix=".json"
    responseSchemaSuffix=".json">
  </zosConnectDataXform>

```

```

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />

<zsoLocalAdapters
  wolaGroup="GROUP"
  wolaName2="NAME2"
  wolaName3="NAME3" />

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<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" />

<!-- To access this server from a remote client add a host -->
<httpEndpoint id="defaultHttpEndpoint"
  host="*"
  httpPort="9080"
  httpsPort="9443" />

</server>

```

Checkpoint 4: after client certificates for authentication

```

<server description="new server">

  <!-- Enable features -->
  <featureManager>
    <feature>jsp-2.2</feature>
    <feature>appSecurity-2.0</feature>
    <feature>zsoConnect-1.0</feature>
    <feature>zsoLocalAdapters-1.0</feature>
    <feature>zsoSecurity-1.0</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

  <safRegistry id="saf" />
  <safAuthorization id="saf" />
  <safCredentials unauthenticatedUser="WSGUEST" profilePrefix="BBGZDFLT" />

  <zsoConnectManager
    globalAdminGroup="GADMIN"
    globalOperationsGroup="GOPERS"
    globalInvokeGroup="GINVOKE"
    globalInterceptorsRef="interceptorList_g" />

  <authorizationInterceptor id="auth" />

```

```

<auditInterceptor id="audit" />

<zsoConnectInterceptors
  id="interceptorList_g"
  interceptorRef="auth,audit"/>

<zsoConnectService id="zosconn"
  invokeURI="/myBatchBackend"
  serviceName="batch-backend"
  serviceRef="wolaService" />

<localAdaptersConnectService id="wolaService"
  registerName="ZOSCONN"
  serviceName="ServiceName"
  connectionFactoryRef="wolaCF" />

<zsoConnectService id="CICS"
  invokeURI="/myCICSBackend"
  serviceName="CICS-backend"
  dataXformRef="xformJSON2Byte"
  invokeGroup="SINVOKE"
  serviceRef="wolaCICS" />

<zsoConnectDataXform id="xformJSON2Byte"
  bindFileLoc="/u/user1/bindfiles"
  bindFileSuffix=".bnd"
  requestSchemaLoc="/u/user1/json"
  responseSchemaLoc="/u/user1/json"
  requestSchemaSuffix=".json"
  responseSchemaSuffix=".json">
</zsoConnectDataXform>

<localAdaptersConnectService id="wolaCICS"
  registerName="CICSREG"
  serviceName="OLACB01"
  connectionFactoryRef="wolaCF" />

<zsoLocalAdapters
  wolaGroup="GROUP"
  wolaName2="NAME2"
  wolaName3="NAME3" />

<connectionFactory id="wolaCF" jndiName="eis/ola">
  <properties.ola />
</connectionFactory>

<sslDefault sslRef="DefaultSSLSettings" />
  <ssl id="DefaultSSLSettings"
    keyStoreRef="CellDefaultKeyStore"
    trustStoreRef="CellDefaultTrustStore"
    clientAuthenticationSupported="true"
    clientAuthentication="true"/>
  <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"

```

```
<!-- To access this server from a remote client add a host -->  
<httpEndpoint id="defaultHttpEndpoint"  
             host="*"   
             httpPort="9080"   
             httpsPort="9443" />  
  
</server>
```

Knowledge Center Links

The topic page for z/OS Connect can be found here:

```
ibm.com/support/knowledgecenter/SSAW57_8.5.5  
/com.ibm.websphere.wlp.nd.multipatform.doc/ae/twlp_setup_zosconnect.html
```

That is a long URL. For those reading this from a PDF, here is a link → [Knowledge Center](#).

Direct access to specific articles may be achieved by searching on the unique file name string for the article. Below is a list of some of the articles found useful during the development of this document. Each search string is itself a hyperlink to the specific page.

<i>Topic</i>	<i>Specific Search String</i>
z/OS Connect message payload transformations	twlp_zconnect_msg_payload_conv
Creating binding and schema files for use with z/OS Connect	twlp_zconnect_binding_and_schema
z/OS Connect REST APIs	twlp_zconnect_rest
Audit Interceptors and SMF 120.11	twlp_zconnect_audit

Document Change History

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

<i>August 02, 2014</i>	Original document at time of Techdoc creation
<i>August 05, 2014</i>	Various updates after initial document was created.
<i>August 21, 2014</i>	More miscellenaous updates.
<i>September 19, 2014</i>	Updated to include information on a problem that surfaces running WOLA for the first time. The symptom is an SEC3 abend of the batch program with RC=20000100. That is likely caused by zOSMF 2.1 running with its Angel operating at some level of code lower than 8.5.5.2.
<i>January 13, 2015</i>	Updated to reflect z/OS Connect inside CICS region support delivered to CICS TS 5.2 via DType APAR PI25503.

End of WP102439
