This page intentionally left blank
z/OS Connect at a High Level

This is the picture we showed back in Unit 1:

*By “batch” we mean a long-running job that uses the WOLA “host a service” API to listen for calls coming over from z/OS Connect*
z/OS Connect is Extensible

InfoCenter search string: \texttt{twlp_admin_extend}

Extending the Liberty profile

You can expand the capability of the Liberty profile by using product extensions. You can write your own Liberty features and install them onto an existing Liberty profile server, or you can package them for delivery to your users.

About this task

- **8.5.5.0** This section describes how to develop features for a product extension, how to install features to the built-in "usr" product extension, and how to use your features in an application server. The Liberty profile provides various System Programming Interfaces (SPIs) that you can use to extend the runtime environment; you can also use more advanced features such as operating the Liberty profile server from your Java™ applications programmatically. **8.5.5.0** The Java API documentation for each Liberty profile SPI is detailed in the Programming Interfaces (APIs) section of the information center, and is also available as a separate .zip file in one of the javadoc subdirectories of the \$\{wlp.install.dir\}/dev directory.

For an overview of writing product extensions for the Liberty profile, see \texttt{Liberty profile: Product extension}.

For full details of how to extend the Liberty profile, see the following subtopics:

- Related tasks:
  - \texttt{z/OS 8.5.5.2 Creating a z/OS Connect service provider}
  - \texttt{z/OS 8.5.5.2 Creating a z/OS Connect interceptor}
  - \texttt{z/OS 8.5.5.2 Creating a z/OS Connect data transformer}

A “service provider” is the mechanism used to reach the backend. The built-in service provider is WOLA. You may add your own.

A “data transformer” is code that transforms JSON to the format required by the backend. The built-in transformer is based on the CICS utility to transform JSON to COBOL COPYBOOK or other language data structure.

An “interceptor” is function called for each request. The built-in interceptors are audit (SMF) and authorization. You may add your own to do custom function.
z/OS Connect Can Be Made Highly Available

Liberty Profile can be made highly available through duplicated server instances. z/OS Connect makes this relatively simple because it's stateless by nature:

Instances of Liberty Profile on different LPARs can share a server.xml and be mirror copies of each other. Or have separate copies that are maintained to be very consistent with one another.

REST is a stateless mechanism, so no need to affinity route.

If backend systems are present on each LPAR then environment is duplicated and HA possible.

Setup ...
Setup

How to do initial enablement of z/OS Connect
z/OS Connect and WOLA are Features of Liberty

And as such, they need to be enabled. This is done with the featureManager shell script, which is provided with Liberty Profile z/OS:

```
Installing File System

<mount>
  /bin
    featureManager

Installs the z/OS Connect feature into Liberty
Installs the WOLA feature into Liberty

./featureManager install zosConnect-1.0 --when-file-exists=ignore

./featureManager install zosLocalAdapters-1.0 --when-file-exists=ignore

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

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

Note: you must have Internet access to run these commands
```

Configuring services ...
Configuring Services

Configuring the z/OS Connect to host services to backend programs
It's All About the `server.xml` File

Always keep in mind that z/OS Connect is only what the contents of `server.xml` tell it to be. The key to understanding z/OS Connect is understanding the XML structure.

Key Points:

- Liberty Profile is, fundamentally, defined by its configuration (`server.xml`)
- z/OS Connect relies on what's in `server.xml` to understand how it is configured and how it is to behave
- There's no magic to this … there is a lot of detail, but no magic
Abstract Flow of Processing of XML Definitions

The following flowchart is intended to offer an abstract view of XML definition processing, which will help put the XML examples that follow in context:

This represents one service; you may have many services defined. Each would have similar structure.
Example of XML Relationships

Here's an example of what the previous flowchart was illustrating … the service definition maps to both a service provider and a data transformation definition:

That is the essential nature of understanding the server.xml for z/OS Connect – service definition to provider definition and including any optional interceptor information.
Essential server.xml Configuration

Before we get into any service and provider definition details, let's take a look at a few essential things that must be in place first:

```
<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>
```

These two `<feature>` definitions enable z/OS Connect and WOLA in the instance of Liberty Profile z/OS

These are required security definitions.

This shows the simplest way to achieve the minimum security requirements:
- Internal (non-SAF) key and trust store
- Basic auth rather than default client certificate authentication
- Internal (non-SAF) user registry
- Internal (non-SAF) mapping of user to role

Structure of service definition ...
Structure of a Service Definition

This may have more XML elements than shown here, but this gives a sense for the key elements and what the service definition provides:

```
<zosConnectService id="CICS_create"
  serviceName="CICS-create"
  dataXformRef="xformJSON2Byte"
  interceptorsRef="interceptorList_1"
  serviceRef="wolaCICS" />
```

1. The `serviceName` value determines the URI that will invoke this service:
   `http://<host>:<port>/zosConnect/services/CICS-create?action=invoke`

2. The `dataXformRef` value points to another section of `server.xml` where the data transform definition for this service can be found. (More on data transform coming up.)

3. The `interceptorsRef` value points to another section of `server.xml` where the optional interceptor list for this service can be found. (More on interceptors coming up.)

4. The `serviceRef` value points to another section of `server.xml` where the service provider definition can be found. That is what defines how to access the backend system for this service.
The URI Clients Would Use for the Defined Service

Using the XML example on the previous chart, let's reinforce the understanding of the URI clients would use to invoke the service:

```
<zosConnectService id="CICS_create"
  serviceName="CICS-create"
  invokeURI="/myCICSCreate"
  dataXformRef="xformJSON2Byte"
  interceptorsRef="interceptorList_1"
  serviceRef="wolaCICS" />
```

This is required

This is optional

```
https://<host>:<port>/zosConnect/services/CICS-create
```

HTTP PUT or POST with JSON

```
https://<host>:<port>/myCICSCreate
```

Both achieve the same thing … the “invokeURI” is simply a way to make the URI simpler and shield clients from understanding it's z/OS Connect

Service provider …
The Service Provider – Access to Backend

The next piece of the puzzle is the service provider, which defines how z/OS Connect is going to access the backend resource:

```
<zosConnectService id="CICS_create"
    serviceName="CICS-create"
    invokeURI="/myCICSCreate"
    dataXformRef="xformJSON2Byte"
    interceptorsRef="interceptorList_1"
    serviceRef="wolaCICS" />
```

Service definition from the previous chart

```
<localAdaptersConnectService id="wolaCICS"
    RegisterName="CICSREG"
    ServiceName="OLACB01"
    connectionFactoryRef="wolaCF" />
```

Two pieces in place – service used to map URI; then provider maps connection to the backend

The WOLA registration name used by the external address space

The backend program name to be called over WOLA

The WOLA JCA resource adapter ID to be used for the WOLA connection

Multiple services ...
Multiple Services, Multiple Providers

You are not limited to a single service, of course ... you may configure many services with many backend provider connections:

There's no practical limit to this ... one, several, hundreds ... your choice.

```xml
<zosConnectService id="ProgramA"
  serviceName="ProgramA"
  serviceRef="CICS1A" />
<zosConnectService id="ProgramB"
  serviceName="ProgramB"
  serviceRef="CICS2B" />
<zosConnectService id="ProgramC"
  serviceName="ProgramC"
  serviceRef="CICS2C" />
<zosConnectService id="ProgramD"
  serviceName="ProgramD"
  serviceRef="Batch" />

<localAdaptersConnectService id="CICS1A"
  RegisterName="CICS1REG"
  ServiceName="A" />
<localAdaptersConnectService id="CICS2B"
  RegisterName="CICS2REG"
  ServiceName="B" />
<localAdaptersConnectService id="CICS2C"
  RegisterName="CICS2REG"
  ServiceName="C" />
<localAdaptersConnectService id="Batch"
  RegisterName="BATCHREG"
  ServiceName="D" />
```
IMS and z/OS Connect

IMS has implemented z/OS Connect in a slightly different way. The same principles apply, but the service provider is another feature to Liberty Profile, not WOLA:

- Part of IMS Enterprise Suite
- IMS Mobile Feature is a z/OS Connect “Service Provider”
- Everything else the same – discovery, data conversion, authorization interceptors, audit interceptors


Data conversion ...
Data Conversion

Converting JSON to the data format of the backend program
Data Conversion at a High Level

This subject focuses on converting JSON to the data format expected by the backend system:

```json
{
  "account": "00012345",
  "firstName": "Smith",
  "lastName": "John",
  "action": "Balance",
}
```

To do this, z/OS Connect needs to understand the relationship between the two formats so it can do the transformation. That's the role of the data transform utility, which produces “bind” files and JSON schema files.

Passthrough or data convert …
With No Data Conversion, JSON in and JSON out

Data conversion is optional; but if you don't use it, then understand that z/OS Connect will pass JSON unchanged to backend and expect JSON on return:

If a service has no data conversion definition, then z/OS Connect will simply pass the received JSON to the backend program unchanged.

This will work only if the backend program understands JSON and the JSON format passed back.

If z/OS Connect passes JSON to the backend program, then it expects to get JSON in return.

If something other than JSON is returned, then an error is thrown by z/OS Connect.

You as administrator of z/OS Connect must have some knowledge of the data requirements.

High-level schematic ...
High-Level Schematic of Data Conversion Process

z/OS Connect uses a utility based on the CICS data conversion utility. It involves creating “bind” files and making those files available to z/OS Connect:

- **Data structure** (example: COPYBOOK)

- **Supplied Data Conversion Shell Script**
  - Supplied JCL to run the shell scripts and define input and output locations

- **Generated Bind and JSON Schema Files**
  - This tells z/OS Connect where files can be found and the suffix values

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

<zosConnectDataURL
   id="xformJSON2Byte"
   bindFileLoc="/u/user1/bindfiles"
   bindFileSuffix=".wsbind"
   requestSchemaLoc="/u/user1/json"
   requestSchemaSuffix=".json"
   responseSchemaLoc="/u/user1/json"
   responseSchemaSuffix=".json">
</zosConnectDataURL>
```

More detailed ...
More Detailed Schematic With Notes

This is from the WP102439 Quick Start Guide. See speaker notes for explanation of numbered blocks:

1. Request COBOL COPYBOOK
2. Batch job that calls the BBGLS2JS proc and passes in parameters to control data conversion
3. Inline parameters
4. BBGLS2JS procedure
5. WRITE
6. z/OS Connect

WP102439 at ibm.com/support/techdocs
The REST APIs

Using the supplied REST APIs to monitor and manage the environment
REST APIs in Context

The REST APIs we now refer to are supplied with z/OS Connect. They are used to discover services and manage the z/OS Connect instance:

The service APIs you define in server.xml for your backend programs

The supplied REST APIs ... these will not appear in server.xml ... they are built-in

This “Role ID” is a very loose term at the moment ... initially all IDs will have access to all REST APIs, but later we'll discuss “authorization interceptors” which provide a more granular access model to the APIs
The Supplied REST APIs for Discovery / Management

Here's a listing of the APIs and a brief description of what each provides:

- `https://host:port/zosConnect/services`
  Returns a JSON object with the configured services the user is authorized to access

- `https://host:port/zosConnect/services/service_name`
  Returns a JSON object with the details for the named service

- `https://host:port/zosConnect/services/service_name?action=status`
  Returns a JSON object with the status of the service (started or stopped)

- `https://host:port/zosConnect/services/service_name?action=getRequestSchema`
  Returns a JSON object with the request schema for the named service

- `https://host:port/zosConnect/services/service_name?action=getResponseSchema`
  Returns a JSON object with the response schema for the named service

- `https://host:port/zosConnect/services/service_name?action=getStatistics`
  Returns a JSON object with request statistics for the named service

- `https://host:port/zosConnect/services/service_name?action=stop`
  Stops the named service and returns a JSON object with the status

- `https://host:port/zosConnect/services/service_name?action=start`
  Starts the named service and returns a JSON object with the status

These can be used by developers to understand the APIs exposed by z/OS Connect ... and by administrators to control services
REST API Example – Get List of Configured Services

Here's an example of the REST APIs in action:

Method: GET
URL: https://wg31.washington.ibm.com:9443/zosConnect/services

REST API to get a list of configured services

In this example two configured services are returned
REST API Example – Get Request Schema

Here's an example of the `action=getRequestSchema` REST API:

```
Audit interceptor …
```

```
getResponseSchema is also available
This allows a developer to not only know what services are available, but what the invoke pattern is and what the JSON request/response patterns are. They may then write their REST/JSON client to the API.

API Management utilities can query this interafce and catalog the results, providing a repository of API information for developers to use.
```
Interceptors

Callout points for each request received
High-Level of Interceptor Framework

Think of “interceptors” as code that is called for each request for a service:

If a service has a reference to an interceptor list, the interceptors defined in that list are invoked for each request that is received by z/OS Connect.
Overview of Interceptor Framework

Interceptors represent optional code that is called upon receive of each request that comes into z/OS Connect:

Optional
You may configure them if you wish, or not, depending on your needs.

May be configured at global level (all) or service level (some)
This allows you to apply interceptors to those services you wish, without necessarily having to crawl through every one updating XML. It is also possible to “opt out” of globally defined interceptors. Quite a bit of flexibility.

Uses interceptors “lists”
A interceptor “list” is simply a bit of XML that specifies which interceptors to invoke and the order in which to invoke them. You can define multiple lists, with some services calling one, and some services calling another.

Extensible
z/OS Connect comes with a Service Provider Interface (SPI) for interceptors so you can write your own if you wish.

Two IBM-supplied interceptors
IBM supplies two: one writes SMF 120.11 records, the other is provides a way to limit what the authenticated user can do in z/OS Connect
Brief Review of Audit Interceptor

The “audit interceptor” is a supplied piece of code that will write SMF 120.11 records. Those records contain information on z/OS Connect usage:

<table>
<thead>
<tr>
<th>Server Identification Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Name</td>
</tr>
<tr>
<td>Sysplex Name</td>
</tr>
<tr>
<td>Jobname</td>
</tr>
<tr>
<td>Job Prefix</td>
</tr>
<tr>
<td>Address Space</td>
</tr>
<tr>
<td>Stoken</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>z/OS Connect User Data Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival Time</td>
</tr>
<tr>
<td>Completion Time</td>
</tr>
<tr>
<td>Target URI</td>
</tr>
<tr>
<td>Input JSON Length</td>
</tr>
<tr>
<td>Response JSON Length</td>
</tr>
<tr>
<td>Method Name</td>
</tr>
<tr>
<td>Service Name</td>
</tr>
<tr>
<td>Userid</td>
</tr>
</tbody>
</table>

Notes:

- Liberty Profile z/OS itself does not cut SMF records; this is a function of z/OS Connect
- SMF 120.11 is relatively new format, so your analysis tool may or may not understand that format yet
- No CPU numbers in 120.11
- Can capture CPU at started task level for Liberty
- z/OS Connect is a pass-through function; it will use some CPU (mostly zIIP).

Multiple services same backend work ...
Two Services Invoking Same Backend Work

Having two defined services invoke the same backend work is doable. But why?

```
<zosConnectService id="Mobile Traffic"
  serviceName="Mobile"
  invokeURI="/Mobile"
  serviceRef="CICSWOLA" />

<zosConnectService id="Internal Traffic"
  serviceName="Internal"
  invokeURI="/Internal"
  serviceRef="CICSWOLA" />

<localAdaptersConnectService id="CICSWOLA"
  RegisterName="CICSREG"
  ServiceName="TRANXYZ" />
```

Authorization interceptor ...
Brief Review of Authorization Interceptor

The “authorization interceptor” is a supplied piece of interceptor code that will check to see if the user has the authority to perform the action requested:

- **Administrator**
  - Full authority
- **Operator**
  - Start, Stop, etc.
- **Invoke**
  - Invoke service only

The mechanics of how this is done and how it is configured is a topic for the unit on security.
Setup of Security Topic

To set the stage for the details on security that are to come next
Intentionally Avoided Security Topic So Far

We have avoided the topic of security for the most part. Some hints of it came through in parts, but in general we've downplayed it. Here's what we'll have after lab:

1. **z/OS Connect Servlet Protected**
   This means we require authentication and SSL; we can't avoid it. But we can make it simpler at first by not using SAF.

2. **Definitions in server.xml**
   The SSL definitions, the user registry, the role definition, and the switch to basic auth from client cert default done here.

3. **Definitions in SAF**
   Some STARTED, SERVER and CBIND profiles are required to support Liberty as STC and WOLA.

4. **Basic Auth from Client**
   z/OS Connect default is client certificate, but basic auth is easier to use initially.

5. **No ID assertion into CICS**
   Call to CICS program done under ID assigned to WOLA BBO$ link server task.