z/OS Connect Enterprise Edition V2.0

Getting Started Guide
for CICS and IMS

Version Date: July 22, 2016

See "Document Change History" on page 115 for a description of the changes in this version of the document

z Systems at your (RESTful) service

IBM z/OS Connect

WP102604 at
ibm.com/support/techdocs

This document was written by:

Don Bagwell (dbagwell@us.ibm.com)
Nigel Williams (nigel_williams@uk.ibm.com)
Carl Farkas (FARKAS@fr.ibm.com)

(If you have comments or feedback on the contents of this document, please send an e-mail to one of us.)

Valuable assistance provided by **Eric Phan**, **Philippe Richard**, **Edward McCarthy**, **Aymeric Affouard**, **Haley Fung**, **Yee-Rong Lai**, and **Kyle Charlet**.

Table of Contents

Overview

This document will provide a task-oriented outline for getting started with z/OS Connect Enterprise Edition V2.0. The document is organized in the following way:

Program number and FMID

Program number **5655-CEE**
FMID **HZCN200**

Recommended Maintenance

Version	Support Document URL
2.0.0.1	http://www-01.ibm.com/support/docview.wss?uid=swg1PI56615

The IBM Knowledge Center

This document leverages the content found in the IBM Knowledge Center for IBM z/OS Connect EE V2.0, which is found at this location:

http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/welcome/WelcomePage.html

IBM Techdocs

This document, as well as other collateral related to IBM z/OS Connect EE V2.0, can be found at the following Techdoc location:

http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102604

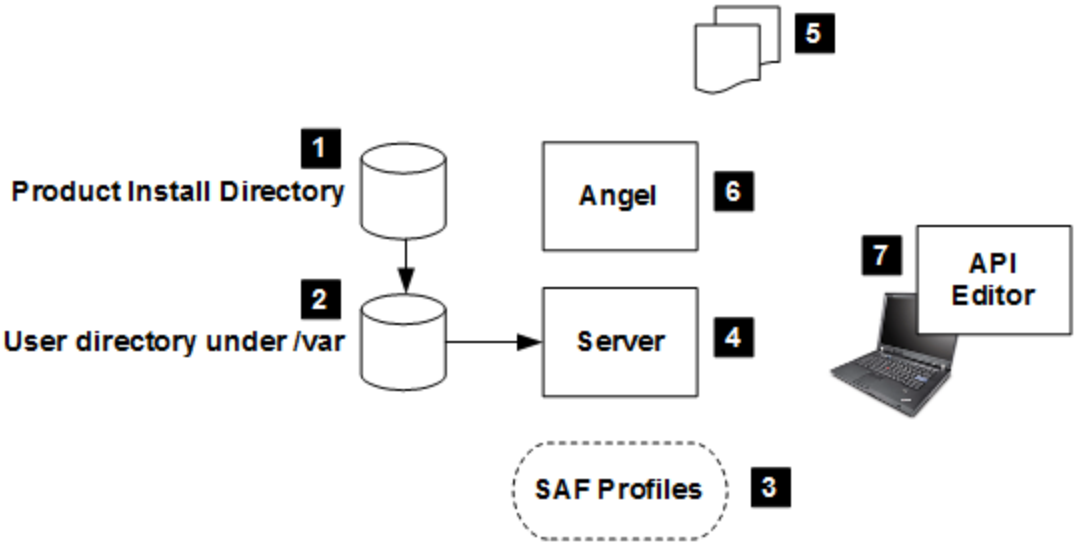
IBM z/OS Connect Enterprise Edition V2.0 web page

Here is the URL for the product web page:

http://www.ibm.com/software/products/en/zos-connect-enterprise-edition

Installation and Initial Setup

Picture overview of the steps in this section



Notes:

1. You use SMP/E to install z/OS Connect EE V2.0. This is a relatively standard SMP/E install process. The result is a set of file systems mounted at the location you specify.
2. You run the `zconsetup` shell script to create a set of directories under `/var` where the z/OS Connect EE V2.0 server directories reside.
3. Certain SAF profiles are required to allow z/OS Connect EE V2.0 to operate as a started task, and for the access into CICS using WOLA to work.
4. You create the server with a simple shell script command.
5. You copy sample JCL procs to your procedure library.
6. The Angel process is only required in some circumstances, and you may already have an Angel present on your system which you may use for z/OS Connect. We guide you through the process of determining the Angel you will use – existing, or new.
7. You install the z/OS Connect EE V2.0 API Editor tooling on your workstation.

Essential prerequsites

You will need the following:

- z/OS 1.13 or 2.1 or higher
- IBM 64-bit SDK for z/OS, Java Technology Edition V7.1.0 or V8.0.0

Do the following:

- ☐ Verify your level of z/OS is 1.13 or 2.1 or higher
- ☐ Check to see if you have a valid 64-bit IBM Java SDK for z/OS, either V7.1.0 or V8.0.0. If one does not exist, then work with your system administrator to have it installed.
- ☐ Note the path to the 64-bit Java SDK you intend to use for z/OS Connect EE V2.0:

JAVA_HOME	

SMP/E install of z/OS Connect EE V2.0 and post-install setup

z/OS Connect EE V2.0 is installed on z/OS using SMP/E. You will require someone with SMP/E skills to accomplish this.

The Knowledge Center page for installing z/OS Connect EE V2.0 is here:

http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/pdir/topics/pdir.html

Do the following:

- ☐ Perform the SMP/E steps indicated in the instructions to install z/OS Connect EE V2.0.
- ☐ Run the `zconsetup` shell script to create the extensions directory and symlink. In the Knowledge Center, the article for this step is found here:

  http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/setup_prod_feature.html

○	Verify the product install file system is mounted R/W. The `zconsetup` shell script will create a symlink in this file system.
○	Open a shell (Telnet or SSH) and go to `/<install_root>/bin`
○	Log on with an ID that has authority to create a symlink and create directories
○	Run the script with this command: **`./zconsetup install`**
○	Remount the product install file system as R/O.

- ☐ Visually inspect the file system. You should see something like this:

Directory	*Purpose*
`/usr/lpp/IBM/zosconnect/v2r0/bin`	Product Code
`/usr/lpp/IBM/zosconnect/v2r0/runtime/lib/`	Feature Files
`/usr/lpp/IBM/zosconnect/v2r0/wlp/etc/extensions`	Symlink to the extensions directory under `/var`
`/var/zosconnect/servers`	Server instances[1].
`/var/zosconnect/v2r0/extensions`	Product feature properties files
`/var/zosconnect/v2r0/extensions/zosconnect.properties`	Properties file

Liberty Angel process

Some functions of z/OS Connect EE V2.0 will require the Angel process[2] [3].

Further, you may already have an Angel process if you have z/OS 2.1 and z/OSMF started[4], or you have created one for some other purpose. If that is the case, then you *may* be able to use that Angel for your z/OS Connect EE V2.0 servers.

If you have an existing Angel process, you may try to re-use that for z/OS Connect EE V2.0. However, if you see: `CWWKB0307E: The angel process on this system is not compatible with the local communication service. The current angel version is 2, but the required angel version is 3.`

Then configure the Angel JCL start procedure to point to the z/OS Connect EE V2.0 installation path and restart the Angel.

1 A subdirectory is created under here for each server instance that is created. Initially there will be no subdirectories. Server creation is described under "Server creation and initial validation" on page 11.
2 Most notably, WOLA for access into CICS. Other functions may as well. It is best to anticipate and have the Angel present for those cases where it is needed.
3 For more on Liberty z/OS and the Angel process: http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110
4 z/OSMF 2.1 is based on Liberty z/OS, and it requires the Angel for access to z/OS authorized services.

SAF updates in support of z/OS Connect EE V2.0

The SAF security profiles for z/OS Connect EE V2.0 are best planned and created ahead of time. This will best utilize your time working with your security administrator.

Note: For the *initial* setup we will keep things simple and host some elements of the security model in the server's `server.xml` file. To understand how to move beyond these simple security definitions, see "Beyond the simple server.xml security elements" on page 107. What follows are z/OS security elements that must be in place before operating the z/OS Connect EE V2.0 server.

Group and server IDs

ID	*Purpose*	*Your Planned Value*
Liberty Group	A group to which Liberty IDs are connected	
Angel Process	ID under which the Angel process operates	
Liberty Server	ID under which the Liberty server operates	

Note: if you already have an Angel process in place, you do *not* need to create another Angel ID. You simply make use of the existing Angel and its ID. Also, it is not required that the Liberty IDs be connected to a common group. We illustrate that here as one approach.

Work with your security administrator and do the following:

- ☐ Plan the values you will use for your Angel ID and server ID, and the group ID.
- ☐ Use the following examples as guides and create the group and IDs:

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
```

Assign the new ID a password

```
ALTUSER liberty_id PASSWORD(password) NOEXPIRED
```

STARTED profiles

The `STARTED` profiles assign the identity when the server is started as a z/OS started task. They are based on the JCL start procedure name. z/OS Connect EE V2.0 comes with sample JCL, and you may keep the default JCL proc names or create your own.

Server	*Default JCL*	*Your Planned JCL proc name*
Angel	BBGZANGL	
Server	BAQSTRT	
Note: we explain where the supplied sample JCL start procedures are located under "Start as z/OS started task" on page 12. Here you are simply planning the start procedure name so you can create the `STARTED` profile for it.		

Note: if you already have an Angel process in place, you do *not* need to create a new JCL proc or `STARTED` profile. You simply make use of the existing Angel JCL proc and its ID.

Work with your security administrator and do the following:

- ☐ Plan your JCL start procedure names (either default or your own values)
- ☐ Use the following examples as guides and create the STARTED profiles:

Creates the STARTED profile for the Angel Process

```
RDEF STARTED angel_proc.* 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 server_proc.* 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

The SERVER profiles grant access to authorized services z/OS Connect EE V2.0 may need. Some of the SERVER profiles are not strictly required for z/OS Connect EE V2.0, but you may decide to create all the profiles indicated just to have them on hand in case you need them later. See the notes that follow for a brief explanation of which are optional and why.

Work with your security administrator and do the following[5]:

- ☐ Use the following examples as guides and create the 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.SAFCRED UACC(NONE) OWNER(SYS1)
PERMIT BBG.AUTHMOD.BBGZSAFM.SAFCRED 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)
```

5 These SERVER profiles can be used by *any* Liberty z/OS, whether z/OS Connect EE V2.0 or not. You may already have these profiles created. If so, then you do *not* need to create the profile, you need only grant your server ID READ to the profile.

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
```

Notes:

- SAFCRED – needed if you intend to use SAF for security elements such as registry, certificates and EJBROLEs. For initial validation you do not need this, but for any real-world usage of z/OS Connect EE V2.0 you will.
- ZOSWLM – needed if you wish to classify work using WLM. Initially you won't do this, but later you might. Better to create now and have available when you need it.
- TXRRS – needed for access to RRS for transaction coordination. You should not need this for z/OS Connect EE V2.0 as it does not create global transactions and therefore does not need the services of RRS for that purpose. You may want to create and have on hand for *other* Liberty servers not running z/OS Connect EE V2.
- ZOSDUMP – needed if you wish to using the MODIFY interface to the Liberty z/OS server to process a dump operation. This is good to have available if IBM support requests a dump for your z/OS Connect EE V2.0 server.
- The others are required for WOLA, which is used between z/OS Connect EE V2.0 and CICS[6]. Connectivity to IMS does not at present use WOLA and therefore they are not strictly required for IMS. Still, creating the full set and having them available for later is something you may wish to consider.

CBIND profiles

The SAF CBIND profile controls what IDs may WOLA register into the Liberty z/OS server. It is only needed if you are using WOLA as part of your z/OS Connect EE V2.0 use case[7].

The CBIND profile is based on the WOLA "three-part name" used by the Liberty z/OS server. Therefore, you must have in mind the WOLA three-part name you will use before you create the CBIND profile.

WOLA elements	*Common Default Values*	*Your Planned Values (upper-case)*
wolaGroup=	GROUP	
wolaName2=	NAME2	
wolaName3=	NAME3	

Note: the three-part name must be unique on the z/OS LPAR. Each component of the three part name may be up to eight characters in length and consist of numbers or letters.

Work with your security administrator and do the following:

☐ Use the following example as a guide and create the CBIND profile:

6 As well as z/OS Connect EE V2.0 to a long running task. See "WOLA and long running task" on page 106 for more on this.
7 Access to CICS uses WOLA; access to IMS does not. If your initial use-case is IMS, then you may skip over this section.

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
```

Note: "accessing_id" is the ID that is attempting to register into the Liberty server using WOLA.

Server creation and initial validation

With z/OS Connect EE V2.0 installed and the required SAF profiles in place, you are ready to create your server and perform initial validation of the environment.

Server creation and start as UNIX task

The Knowledge Center URL for this task is:

http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/creating_zC_server.html

Do the following:

- ☐ Open a Telnet, SSH or OMVS session to your z/OS system. *Log in as the ID you planned for the server.*
- ☐ Export your `JAVA_HOME` to the environment:

  `export JAVA_HOME=`*`path_to_your_64-bit_Java_SDK`*

- ☐ Test to make sure the ID has the ability to instantiate a JVM. Do the following:

○	Change directories to the `/bin` directory of your `JAVA_HOME` location

○	Issue the command: **`./java -version`**
	You should receive something like this:
	`java version "1.7.0"`
	`Java(TM) SE Runtime Environment (build pmz6470sr6fp1-20140108_01(SR6 FP1))`
	`IBM J9 VM (build 2.6, JRE 1.7.0 z/OS s390x-64 ... (JIT enabled, AOT enabled)`
	`J9VM - R26_Java726_SR6_20140106_1601_B181350`
	`JIT - r11.b05_20131003_47443.02`
	`GC - R26_Java726_SR6_20140106_1601_B181350`
	`J9CL - 20140106_181350)`
	`JCL - 20140103_01 based on Oracle 7u51-b11`
	If it fails (with an error `JVMJ9VM011W EDC5204E`) then it is likely because your ID does not have the ability to get the memory needed to create the JVM. Adjust[8] the size parameters of the userid with something like this:
	`ALU user-name TSO(SIZE(1048576)) OMVS(ASSIZEMAX(1073741824) MEMLIMIT(1G))`

  When you are able to successfully check the version of Java, then proceed.

- ☐ Go to the `/bin` directory where z/OS Connect EE V2.0 is installed.
- ☐ Issue the following command to create a server:

  `zosconnect create `*`server_name`*` --template=zosconnect:default`

  Where *`server_name`* is any value you wish, such as `server1`. Capture the name of your server here:

Your server name value: *(case sensitive)*	

8 You may need to work with your system administrator to accomplish this. The key point the ID must be able to instantiate a JVM or you can not proceed. This test checks to see if the ID has the ability. If not, correct the issue.

☐ Go to the `/var/zosconnect/servers` directory and verify that a sub-directory with the name *`server_name`* was created, and under the *`server_name`* directory there exists a `server.xml` file.

TCP ports and `host="*"` element

A few minor updates to `/var/zosconnect/servers/`*`<server_name>`*`/server.xml` are called for at this point. Do the following:

☐ Consult with your TCP networking administrator and see if the default ports of 9080 and 9443 are acceptable. If not, plan the two TCP ports you will use:

HTTP port `httpPort=` in server.xml	
HTTP secure port `httpsPort=` in server.xml	

☐ Edit the `server.xml` file and update and add what's highlighted in yellow:

```
<httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />
```

Notes:

- The `host="*"` element allows a remote client (your browser) to access the Liberty z/OS server on any virtual host. Without this only browsers on `localhost` would be able to access. z/OS does not have a local browser, so including a `host=` attribute is needed. Asterisk (any host) is the easiest to start with.
- The two ports should reflect either the default values (shown) or your planned values.

☐ Save the file.

Start as z/OS started task

Earlier you created the `STARTED` profiles to assign the userid to the started task. z/OS Connect EE V2.0 comes with sample JCL start procedures you copy to your proclib and customize to your environment.

Do the following:

☐ Copy the sample server JCL from:

`/`*`<install_mount>`*`/v2r0/jcl/baqstrt.jcl`

to your proclib[9]. Make sure the resulting proc profile does not have 'numbers' off to the right of the member. If you find them, issue command `unnum` to remove the numbers. That will also set the profile to `NUMBER OFF`.

☐ Rename the proc so it matches the `STARTED` profile you created for the server.

☐ Customize the server JCL:

```
//BAQSTRT  PROC PARMS='defaultServer'
//*
  (comment lines removed to save space in this document)
//*-------------------------------------------------------------------
//* Start the Liberty server
//*
//* STDOUT  - Destination for stdout (System.out)
//* STDERR  - Destination for stderr (System.err)
//* STDENV  - Initial z/OS UNIX environment for the specific
```

9 Use OCOPY (see: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.bpxa500/tsoocopy.htm), or put a 'c' next to the file in ISHELL and copy to your proclib.

```
//*           server being started
//*
// SET ZCONHOME='<Install path>'  1
//*
//ZCON     EXEC PGM=BPXBATSL,REGION=0M,MEMLIMIT=4G,
//         PARM='PGM &ZCONHOME./bin/zosconnect run &PARMS.'
//STDOUT   DD   SYSOUT=*
//STDERR   DD   SYSOUT=*
//STDIN    DD   DUMMY
//STDENV   DD   *
_BPX_SHAREAS=YES
JAVA_HOME=<Java home directory>   2
#JVM_OPTIONS=<Optional JVM parameters>
//*
// PEND
//
```

Notes:

1. Set the *<Install path>* value to the path of the z/OS Connect EE V2.0 install location, for example `/usr/lpp/IBM/zosconnect/v2r0` or whatever your value is. Make sure to enclose the value in single quotes as shown in the JCL.
2. Set `JAVA_HOME=` to the path to your 64-bit IBM Java SDK. For example: `/usr/lpp/java/J7.0_64/`

If you intend to use an already-existing Angel process, then skip over the following steps[10]. Otherwise, follow these steps to create and start an Angel process.

- ☐ Copy the sample Angel JCL from:

  /<install_mount>`/IBM/zosconnect/v2r0/wlp/templates/zos/procs/bbgzangl.jcl`

  to your proclib.
- ☐ Rename the proc so it matches the `STARTED` profile you created for the Angel.
- ☐ Customize the Angel JCL:

```
//BBGZANGL PROC PARMS='',COLD=N
//*--------------------------------------------------------------------
//  SET ROOT='/u/MSTONE1/wlp' 1
//*--------------------------------------------------------------------
-
//* Start the Liberty angel process
//*--------------------------------------------------------------------
//* This proc may be overwritten by fixpacks or iFixes.
//* You must copy to another location before customizing.
//*--------------------------------------------------------------------
//STEP1   EXEC PGM=BPXBATA2,REGION=0M,
//  PARM='PGM &ROOT./lib/native/zos/s390x/bbgzangl COLD=&COLD &PARMS'
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
//* ====================================================================
*/
```

Notes:

1. Change the `SET ROOT=` value so it reflects the install location for z/OS Connect EE V2.0, *including* the `/wlp` sub-directory.

- ☐ Start the Angel with `/S` *<angel_proc>*

10 If you see "CWWKB0307E: The angel process on this system is not compatible with the local communication service. The current angel version is 2, but the required angel version is 3," then update the existing Angel JCL start proc to point to z/OS Connect EE V2.0 and restart the Angel.

- ☐ Verify the Angel received the ID you intended. This validates the `STARTED` profile you created for the Angel process.

Then start the server and verify basic operations:

- ☐ Start the server with the following command

  S *<server_proc>*,PARMS='*<server_name>*'

  Where *<server proc>* is the name you gave your z/OS Connect EE V2.0 server JCL start procedure, and *<server_name>* is the name you gave your created server.

  Notes: The `PARMS=` value is case-sensitive. Issue this command in the z/OS "command extension" (a single forward slash in SDSF) to preserve the case. Issuing `/S <proc>` will fold to uppercase.

  If you want to simplify the start command to *just* `/S <proc>`, then update the first line of the JCL and set `PARMS=` there to include your server name. Then when you issue `/S <proc>` the `PARMS='<server_name>'` on the first line will be used.

- ☐ Verify the server received the ID you intended. This validates the `STARTED` profile you created for the server.
- ☐ Go to the `/var/zosconnect/servers/<server_name>/logs` directory
- ☐ Look in the `messages.log` file. You should see the following indicators of success[11]. See notes that follow:

```
CWWKE0001I: The server server_name has been launched.
CWWKB0103I: Authorized service group LOCALCOM is available.    1
CWWKB0103I: Authorized service group SAFCRED is available.
CWWKB0103I: Authorized service group WOLA is available.
CWWKB0104I: Authorized service group PRODMGR is not available.    2
CWWKB0104I: Authorized service group TXRRS is not available.
CWWKB0104I: Authorized service group ZOSDUMP is not available.
CWWKB0104I: Authorized service group ZOSWLM is not available.
 :
CWWKO0219I: TCP Channel defaultHttpEndpoint has been started and is
now listening for requests on host *  (IPv6) port port.    3
CWWKS4105I: LTPA configuration is ready after 0.612 seconds.
CWWKF0012I: The server installed the following features:
                          [servlet-3.1, ssl-1.0, jndi-1.0, json-1.0,
                    4   zosconnect:zosConnect-2.0, distributedMap-1.0,
                                      appSecurity-2.0, jaxrsClient-2.0].
CWWKF0008I: Feature update completed in 3.101 seconds.
CWWKF0011I: The server server_name is ready to run a smarter planet.
SRVE0169I: Loading Web Module: z/OS Connect.    5
SRVE0250I: Web Module z/OS Connect has been bound to default_host.
CWWKT0016I: Web application available (default_host):
                                                6    http://<host>:<port>/
```

Notes:

1. The "Authorized service group" messages indicate the success of the server to access the Angel process with the `SERVER` profiles you created. For CICS and WOLA you want `LOCALCOM` and `WOLA` to show as "is available."

   Important! If you intend to use z/OS Connect EE V2.0 with CICS, you will need `LOCALCOM` and `WOLA` to show as "is available." If they do not, then check: (a) the Angel process is started, (b) the `SERVER` profiles were created, and (c) the server ID has READ to the `SERVER` profiles. Correct any errors and restart the server.

11 The messages may occur in a slightly different order. That's okay; the important thing is the various success indicators are present.

2. Some "Authorized service group" messages will indicate "not available." PRODMGR will always show as "not available." That is acceptable if you do not plan to use those services. `TXRRS`, `ZOSDUMP` and `ZOSWLM` may not be available depending on what `SERVER` profiles you created and whether the server ID was granted `READ` to the profile.
3. You should see your HTTP port show up in this message.
4. You should see `zosconnect:zosConnect-2.0` show up in the features that were installed.
5. The z/OS Connect web module should show loaded
6. The web application of z/OS Connect will then be available on the host and port of your server.

If your server looks good at this point, then proceed.

Setup of basic security in server.xml in support of z/OS Connect EE V2.0

Here you will set up security definitions in the `server.xml` to provide the *minimum required* (by default[12]) by z/OS Connect EE V2.0.

Key Point: We will keep this as simple as possible at this phase of setup and validation. We do that because we want to get you to the definition of services and APIs as quickly and easily as possible. The security setup we illustrate here works, but is definitely *not* suitable for anything but testing purposes. If you are interested in seeing how to enable SAF to perform these security functions, see "Encryption and Authentication in SAF" on page 107.

Do the following:

- ☐ Go to the `/var/zosconnect/servers/<server_name>` directory
- ☐ Edit the `server.xml` file
- ☐ Add the XML shown here highlighted in yellow (see notes that follow):

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>zosconnect:zosconnect-2.0</feature>
        <feature>appSecurity-2.0</feature>    1
    </featureManager>

    <keyStore id="defaultKeyStore" password="Liberty" />    2

    <webAppSecurity allowFailOverToBasicAuth="true" />    3

    <basicRegistry id="basic1" realm="zosConnect">    4
        <user name="Fred" password="fredpwd" />
    </basicRegistry>

    <authorization-roles id="zos.connect.access.roles">    5
        <security-role name="zosConnectAccess">
            <user name="Fred"/>
        </security-role>
    </authorization-roles>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="<port>"
                  httpsPort="<port>" />
```

12 The requirement for encryption and authentication can be turned off. See "Turning off SSL and Authentication" on page 109.

```
</server>
```

Notes:

1. Enables application security, which z/OS Connect EE V2.0 uses[13].
2. Enables use of a default key/trust store generated by Liberty. This allows SSL from the REST client to z/OS Connect EE V2.0 without having to introduce the complexity of creating and managing certificates at this point.
3. This will result in a userid and password prompt at the REST client, rather than using the default client certificate mechanism.
4. This defines a user registry with a single entry of `Fred` and a password.
5. z/OS Connect EE V2.0 requires the authenticated user to have role access as well. This provides that access.

☐ Save the file.

☐ Go to the `/var/zosconnect/servers/<server_name>/logs` directory and view the `messages.log` file. You should see the following messages at the end of the log indicating the server has been updated:

```
CWWKG0016I: Starting server configuration update.
CWWKS9112A: The web application security settings have changed. The
    1  following properties were modified: allowFailOverToBasicAuth=true
CWWKS0008I: The security service is ready.
CWWKS9120I: Authorization roles with id="zos.connect.access.roles" have
                                        2  been successfully processed.
CWWKO0219I: TCP Channel defaultHttpEndpoint-ssl has been started and
          is now listening for requests on host *  (IPv6) port port. 3
CWWKF0007I: Feature update started.
CWWKG0017I: The server configuration was successfully
                                                updated in 0.297 seconds.
CWWKF0008I: Feature update completed in 0.212 seconds.  4
```

Notes:

1. The failover to basic authentication property was processed
2. The application role was processed
3. The server started listening on its defined secure port, which means the minimum SSL requirements are in place
4. The update has successfully completed.

☐ Point a browser at the following URL:

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

where:

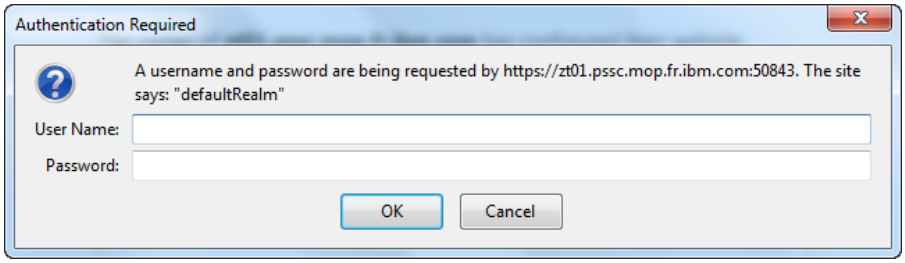
- The protocol is `https` (note the "s")
- *<host>* is the TCP host for your server
- *<port>* is the secure port (httpsPort=) for your server
- The "C" in "zosConnect" is in uppercase (otherwise you'll get a 404 not found error)

☐ Your browser will challenge the security of the connection because the certificate authority that signed the server certificate is the default Liberty CA, and your browser does not recognize that. Accept the challenge[14].

13 This is actually redundant. If you look at the `messages.log` output from earlier, you will see that `appSecurity-2.0` is loaded automatically. That's because z/OS Connect EE V2.0 was loaded, and application indicated it needed `appSecurity-2.0`. So Liberty auto-loaded it. Including it as a `<feature>` does not hurt. It is a good visual reminder of key features required by z/OS Connect EE 2.

14 This will create an error in `messages.log` and an FFDC directory with entries there to capture the error. This is expected.

☐ You should then get a basic authentication prompt:



This is because of the `allowFailOverToBasicAuth="true"` in the `server.xml`.

Provide the userid and password you supplied for the `basicRegistry` entry in the `server.xml` file: **`Fred`** and **`fredpwd`** (*this is case sensitive*).

☐ You should then see:

```
{"zosConnectServices":[]}
```

That is telling you z/OS Connect sees no services currently configured. That is a good sign at this point – it is telling you the Liberty z/OS server recognizes that z/OS Connect EE V2.0 is in fact active, but no services are currently present.

☐ Stop the server with **`/P`** ***`<server_proc>`***[15] [16]. This will give you a clean `messages.log` on the next start, which makes it easier to look for and find the key success messages.

The essentials are in place for you to begin coding up services and using the API editor to create the API artifacts.

z/OS Explorer and z/OS Connect EE V2.0 tooling install

The composition of APIs for z/OS Connect EE V2.0 requires the tooling designed for that purpose. That tooling is called "z/OS Connect EE API Editor," and it is an Eclipse-based tool for creating and editing API definitions.

There are two steps to this process: (1) installing an Eclipse platform, and (2) installing the z/OS Connect EE API Editor tool into the Eclipse platform[17].

Installing an Eclipse runtime platform

The z/OS Connect EE API Editor is a plugin tool to an Eclipse platform, such as:

- IBM Explorer for z/OS Aqua
- IBM IMS Enterprise Suite Explorer for Development V3.2.1
- IBM CICS Explorer V5.3.

If you already have one of those installed, then you may jump to the next section.

The following are the instructions for installing IBM Explorer for z/OS Aqua

☐ Go to the following URL:

`https://developer.ibm.com/mainframe/products/downloads/`

☐ Follow the instructions you find there to install using either IBM Installation Manager or "from scratch" (meaning you do not have Installation Manager).

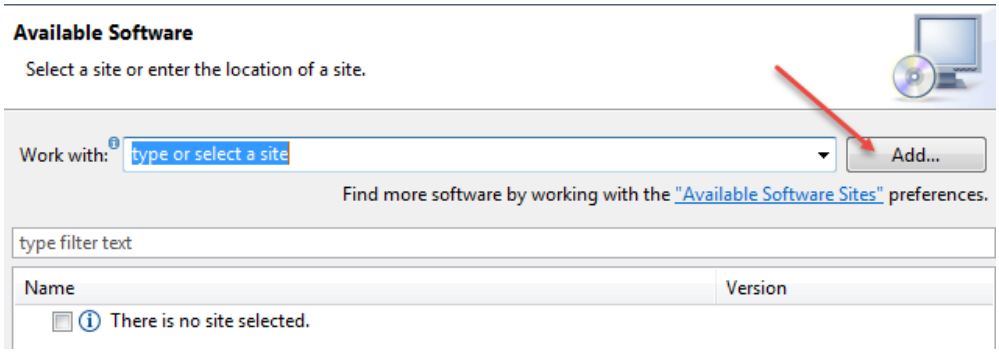
☐ When completed, start IBM Explorer for z/OS Aqua.

15 At initial release of z/OS Connect EE V2.0 the STOP command did not stop the server. Use /C to cancel the server.
16 It's really `/P <jobname>`, but earlier you started the server with just the proc name, so that becomes the jobname as well.
17 KC: http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/zc_install_editor.html

Installing the z/OS Connect EE V2.0 API Editor

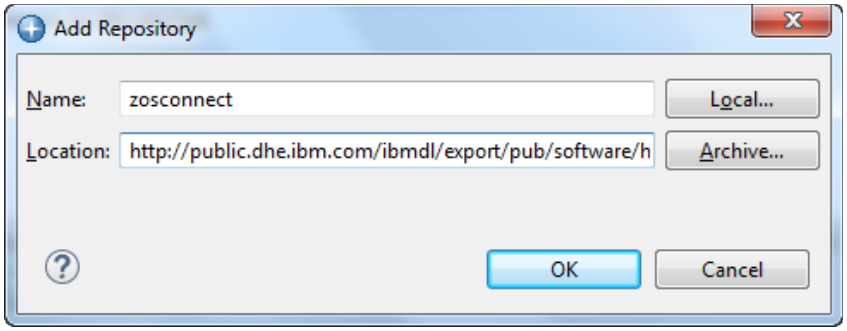
The z/OS Connect EE V2.0 API Editor is a plugin to the Eclipse platform you will use. Installing the plugin is a relatively simple thing:

- ☐ From your open Eclipse platform[18], select *Help* → *Install New Software*.
- ☐ Then click the "Add" button:

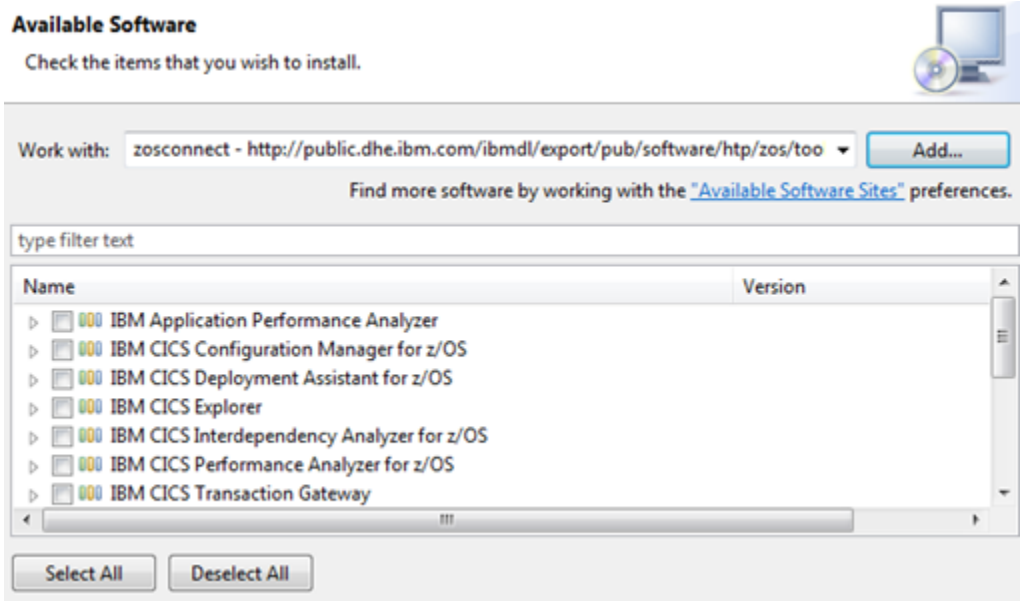


- ☐ Provide a name (such as `zosconnect`) and then for "Location" provide this URL:

`http://public.dhe.ibm.com/ibmdl/export/pub/software/htp/zos/tools/aqua/`

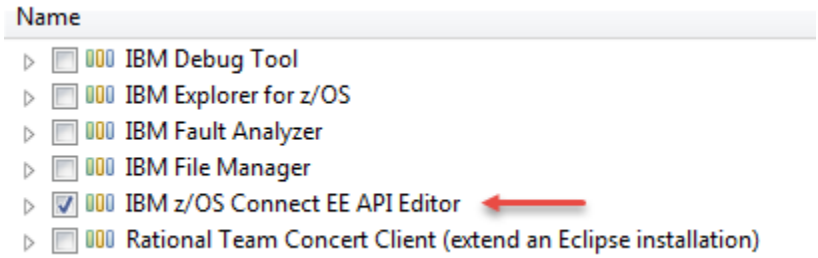


- ☐ Click "OK"
- ☐ It will spend a little time searching for the tools available at that location. You will see a "Pending" indicator. Then it will populate the window with something like this:

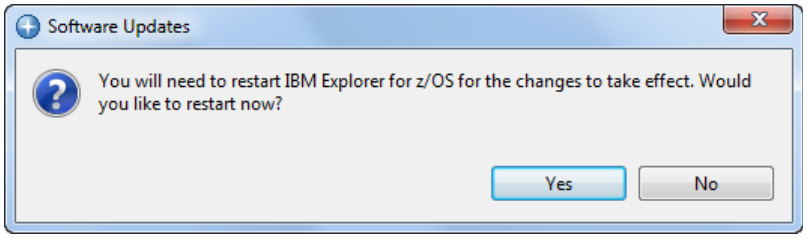


18 Either IBM z/OS Explorer, CICS Explorer, or IMS Explorer.

- ☐ Scroll down, locate and check "IBM z/OS Connect EE API Editor":

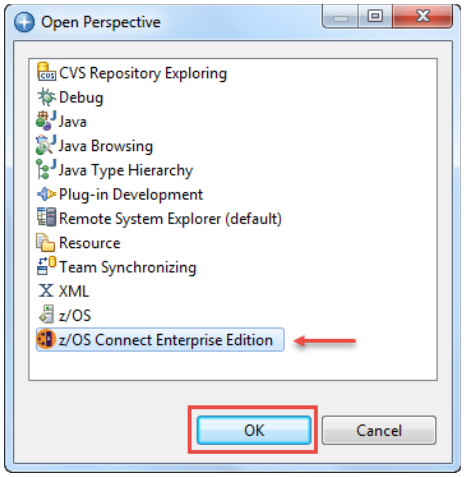


- ☐ Click "Next", then "Next" again … then agree to the license agreement. Then click "Finish."
- ☐ It will indicate you need to restart



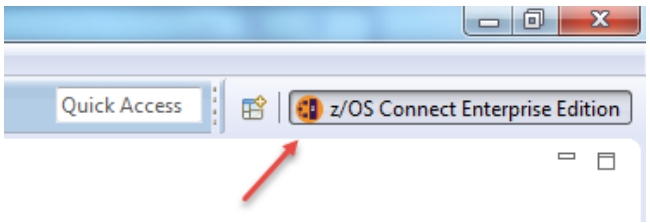
Click "Yes."

- ☐ Eclipse will restart. When it is open, select *Window* → *Open Perspective* → Other and select "z/OS Connect Enterprise Edition":



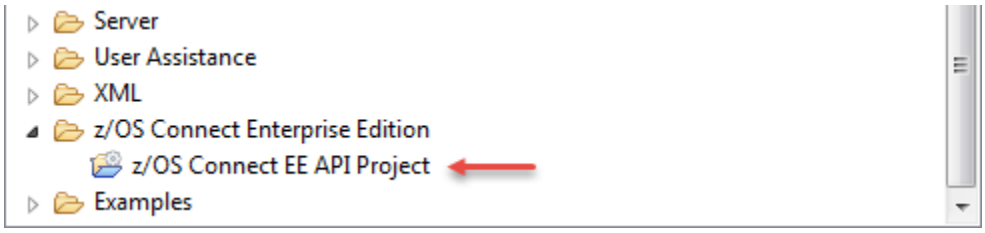
And click "OK".

- ☐ In the upper-right corner you should see something like this:



Note: there may other "perspectives" showing there. The key is seeing the "z/OS Connect Enterprise Edition" perspective indicated and highlighted.

☐ Now click *File* → *New* → *Other*, then scroll down, open the folder "z/OS Connect Enterprise Edition" and *look for* "z/OS Connect EE API Project":



Note: this verifies that the plugin is installed and ready to use. You will use the API editor later in the install/setup process.

☐ Click "Cancel". Close Eclipse if you wish.

Checkpoint: status at this point

At this point you have:

- z/OS Connect EE V2.0 installed
- Key SAF profiles created
- A server created and capable of starting as a z/OS started task
- The basic security structure is in place for z/OS Connect EE V2.0

You have the foundation in place. Now you branch to the backend system of interest to you – CICS (page 21), or IMS (page 45), or a long running task (page 106).

CICS – WOLA Setup, Validation, and the Catalog Manager Sample

Setup of WOLA in server.xml

Earlier you created the server and you used the default template. To use WOLA requires a few more updates to the `server.xml`.

This procedure is outlined in the following Knowledge Center article:

http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/wola_service_provider.html

Update server.xml

Do the following:

- ☐ Go to the `/var/zosconnect/servers/<server_name>` directory
- ☐ Edit `server.xml` and add the lines highlighted here in yellow. See notes:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>zosconnect:zosconnect-2.0</feature>
        <feature>appSecurity-2.0</feature>
        <feature>zosLocalAdapters-1.0</feature>    1
    </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" />    2

    <connectionFactory id="wolaCF" jndiName="eis/ola">
        <properties.ola />
    </connectionFactory>    3

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="<port>"
                  httpsPort="<port>" />

</server>
```

Notes:

1. The `zosLocalAdapters-1.0` feature loads the WOLA feature.

2. This the "three-part name" for the server you planned earlier (page 10). Make sure the value you provide here exactly matches what you planned, and what you used when the `CBIND` profile was created. This is important to allow the CICS region to use WOLA and "register into" the Liberty server using the three-part name.
3. This defines the WOLA JCA resource adapter connection factory.

☐ Save the file.

Start server and verify key messages are present

Here you will start the server and check to make sure key messages appear that indicate the WOLA support is available. Do the following:

☐ Start the server with **`/s`** ***`<proc_name>`***

☐ Go to the `/var/zosconnect/servers/`*`<server_name>`*`/logs` directory and look in the `messages.log` file. Look for the following indicators of successful initialization of the WOLA function:

```
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
 :
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with
    the Liberty profile server using the following name: GROUP NAME2 NAME3 3
 :
J2CA7001I: Resource adapter ola installed in 0.327 seconds.  4
```

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. You verified this earlier.
2. The `WOLA` service group being available is another key indicator that the setup work was successfully completed.
3. 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 (CICS in this document) may register into the Liberty Profile server using this three-part name.
4. The WOLA JCA resource adapter is available.

Setup of WOLA in CICS region

Important: To use WOLA with CICS TS 5.3 requires the fix for APAR PI56615. The fix applies to the WOLA TRUE program that loads into the CICS region. The symptom is a generic BBOX abend in the BBOATRUE program. For more, see "WOLA and CICS TS 5.3 or higher" on page 106.

This involves a few relatively simple CICS system programmer tasks.

Copy WOLA modules to PDSE

Do the following:

☐ Allocate a PDSE 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 under where z/OS Connect EE V2.0 is installed:

  /*<install_mount>*/v2r0/wlp/clients/zos/

  There you will find the WOLA modules:

```
USER:/<install_mount>/czosconnect/v2r0/wlp/clients/zos/-> ls
bboa1cng  bboa1inv  bboa1rcs  bboa1srq  bboa1urg  bboacsrv
bboa1cnr  bboa1rca  bboa1reg  bboa1srv  bboaclnk  bboatrue
bboa1get  bboa1rcl  bboa1srp  bboa1srx  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.ZCONNEE2.LOADLIB'"
```

  You should see something like this:

```
./bboa1cng -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1cng)': executable
./bboa1cnr -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1cnr)': executable
./bboa1get -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1get)': executable
./bboa1inv -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1inv)': executable
./bboa1rca -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1rca)': executable
./bboa1rcl -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1rcl)': executable
./bboa1rcs -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1rcs)': executable
./bboa1reg -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1reg)': executable
./bboa1srp -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1srp)': executable
./bboa1srq -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1srq)': executable
./bboa1srv -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1srv)': executable
./bboa1srx -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1srx)': executable
./bboa1urg -> //'LIBERTY.ZCONNEE2.LOADLIB(bboa1urg)': executable
./bboaclnk -> //'LIBERTY.ZCONNEE2.LOADLIB(bboaclnk)': executable
./bboacntl -> //'LIBERTY.ZCONNEE2.LOADLIB(bboacntl)': executable
./bboacsrv -> //'LIBERTY.ZCONNEE2.LOADLIB(bboacsrv)': executable
./bboatrue -> //'LIBERTY.ZCONNEE2.LOADLIB(bboatrue)': executable
```

- ☐ Browse the target PDSE to verify the modules are present.

Get CSD update samples from Github, customize and submit

An update to the CSD of your CICS region is required. A sample CSD update job is provided here: https://github.com/WASdev/sample.wola

Work with your CICS administrator and do the following:

- ☐ Pull the `CSDUPDAT.jclsamp` sample from Github.
- ☐ Upload it to your z/OS system, customize for your environment and submit.
- ☐ Install the group, for example: `CEDA INSTALL GROUP(xxxx)`
- ☐ Add the group to a list so the definitions are available at CICS initialization; for example: `CEDA ADD GROUP(xxxx) LIST(yyyy)`

Concatenate WOLA modules to DFHRPL

To make the WOLA modules available to the CICS region, do the following:

- ☐ Concatenate the WOLA module library to the `DFHRPL` for the region
- ☐ Start the region and verify the WOLA definitions are in effect:
  - ○ You should see a `BBOOUT` DDNAME in the held output for the region.
  - ○ In `MSGUSER` you should see several messages when you search on the string BBO

Validation of WOLA between Liberty and CICS region

Your CICS region should now have WOLA available to it. The next two steps make ready WOLA for use by z/OS Connect EE V2.0.

Start the WOLA Task Related User Exit (TRUE)

The WOLA Task Related User Exit (TRUE)

Do the following:

- ☐ Log onto a CICS terminal session for the region
- ☐ Issue the command: **`BBOC START_TRUE`**

  You should then see:

  ```
  WOLA TRACE 1: Exit enabled successfully.
  ```

Start the WOLA Link Server Task and register into the Liberty server

The WOLA "Link Server Task" (BBO$) handles the WOLA calls from z/OS Connect EE V2.0 and shields the target programs from needing any awareness of WOLA.

When the Link Server Task is started, it will register into the Liberty z/OS server where z/OS Connect EE V2.0 is running, based on the parameters you supply the Link Server Task.

Do the following:

- ☐ Refresh your understanding of the "three-part name" you used when you configured your server's `server.xml` file[19]. Note them here:

WOLA elements	*Your Values*
`wolaGroup=`	
`wolaName2=`	
`wolaName3=`	

- ☐ Log onto a CICS terminal session for the region
- ☐ Issue the following *as one command*:

```
BBOC START_SRVR RGN=ZCONNREG DGN=group NDN=name2 SVN=name3
                                    SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y
```

Where the values for *group*, *name2* and *name3* are your WOLA three-part name values as specified in the server's `server.xml` and included in the `CBIND` profile. Check the results against this table:

Indication	*Explanation*
Start server completed successfully	Successful registration into the Liberty server
RC=12 / RSN=16	Either the Liberty server is not up, or the three-part name on the `START_SRVR` command does not match the three-part name hosted by the server.
RC=12 / RSN=14	Access restricted by `CBIND`. Either the `CBIND` profile does not exist, or it uses a three-part name value different from what was used here, or the effective ID[20] did not have `READ` to the profile.

19 The "three-part name" is also referenced in the SAF `CBIND` profile. It's important to make sure you are consistent in the use of the exact same three-part name values on the `CBIND`, in the `server.xml`, and on the `BBOC START_SRVR` command.

20 If the Link Server task is started by CICS (INITPARM, PLT, etc.), then it will be the ID of the CICS region. If started manually with the BBOC command, then either the CICS region ID (if security is not enabled for CICS), or the ID you logged into CICS (if security enabled).

SEC3 ABEND	This occurs when you are using the z/OSMF Angel process and that's running at a level prior to 8.5.5.2. The solution here is to stop the z/OSMF Angel, upgrade the Liberty z/OSMF uses to something higher than 8.5.5.2, and restart the Angel.

Notes about automating at CICS region startup

The start of the TRUE and the start of the Link Server task can be automated with startup of the CICS region. The steps to do that are outside the scope of this document. However, here are a few notes about this:

- The Github location where you pulled the CSD update JCL also provides some sample PLT programs that can be used for this purpose.
- The WP101490 Techdoc[21] location has a "WOLA Quick Start Guide" that includes a detailed section on initializing WOLA at the start of the CICS region. Pull the "WOLA Quick Start Guide" under the "Full Function WAS z/OS" header. Then go to "Appendix B: Starting Link Server Task at CICS Startup."

Install Catalog Manager sample in CICS

For this document we will use the CICS "catalog manager" sample application. That application simulates an office supplies store application. It is useful for illustrating z/OS Connect EE V2.0 because it is plausibly "real world" while not being overly-complex[22].

The details of this CICS sample application are provided here:

https://www.ibm.com/support/knowledgecenter/SSGMCP_5.2.0/com.ibm.cics.ts.examapplication.doc/topics/dfhxa_t100.html

Work with your CICS admininstrator and do the following:

- ☐ Enable the catalog manager sample application based on the instructions provided in the URL given above.
- ☐ Verify the sample application is functional by accessing it with the transaction EGUI from a 3270 CICS terminal session.

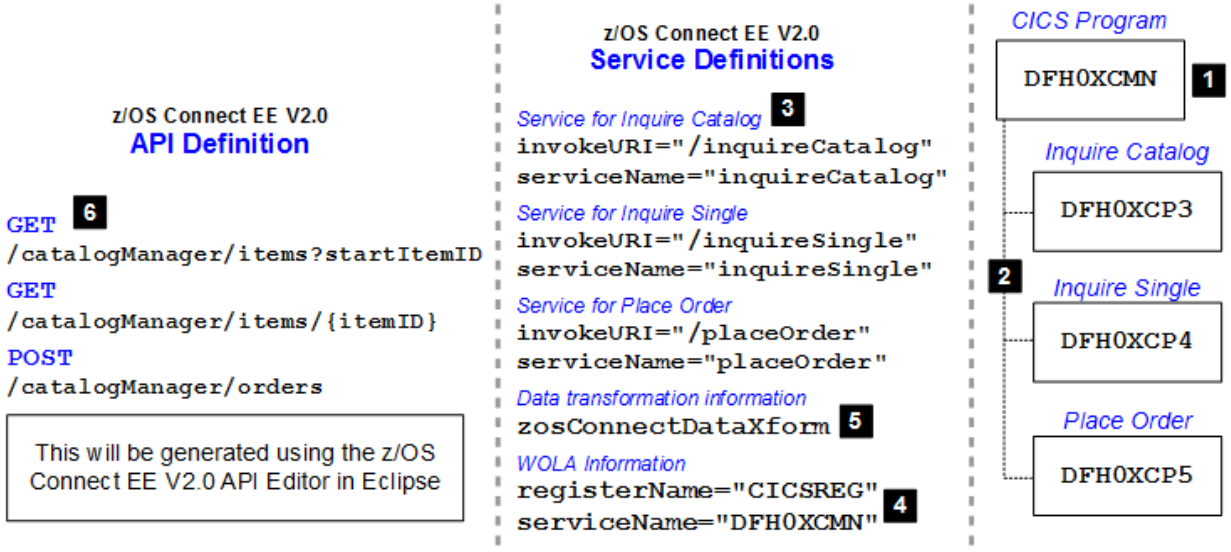
The steps that follow will guide you through creating the service and API definitions to access that sample application using REST and z/OS Connect EE V2.0.

FYI - Plan the Catalog Manager services and API

At this point it is best to pause and plan construction of the z/OS Connect EE V2.0 services and API to be created. The following picture provides a high-level of what you will build in the following sections. The numbered blocks correspond to the notes that follow.

21 http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490

22 If you wish to start with something *really* simple, use the WOLA sample `OLACB01`, which is a simple echo program with a single PIC X(80) data structure that serves as both the request and the response field. See "OLACB01 sample" on page 105.



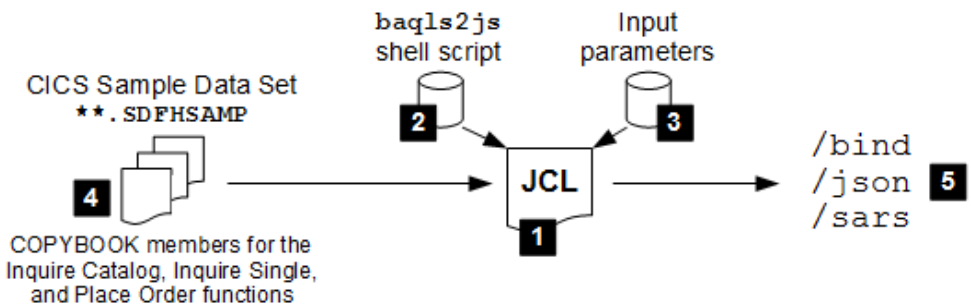
Notes:

1. The CICS Catalog Manager sample is invoked through the `DFH0XCMN` program. That program is what's named on the WOLA `serviceName=` value (block 4).
2. The program has three functions: "Inquire Catalog," "Inquire Single," and "Place Order." The COMMAREA structure for each is supplied as `DFH0XCP3`, `DFH0XCP4` and `DFH0XCP5` respectively. Those are supplied as members in the `**.SDFHSAMP` data set.

   You will run those members through the `BAQLS2JS` utility to create the bind files, JSON schema and "Service Archive" (SAR) file, which is imported into the z/OS Connect EE V2.0 API Editor.
3. In the z/OS Connect EE V2.0 server's `server.xml` file you will configure service definitions, one for each function in the CICS program. The "invoke URI" patterns for each are: /`inquireCatalog`, /`inquireSingle` and /`placeOrder` respectively.
4. All three services will use WOLA to get to the CICS region where the Catalog Manager program runs. The WOLA definition specifies the WOLA registration over which the communications will flow, as well as the program name to invoke (see block 1).
5. All three services will use the same location where the bind files and JSON schema files are kept. The artifacts maintained at this location are generated by the `BAQLS2JS` utility.
6. Finally, the API will be constructed as shown on the chart. *Inquire Catalog* and *Inquire Single* will use the HTTP `GET` verb, while Place Order will use the `POST` verb. For more on the design of the API, see "A discussion of why the Catalog Manager API was designed as it was" on page 95.

Create the data conversion and SAR artifacts

The CICS program has three members that need to be run through the `BAQLS2JS` utility to produce the BIND files, the JSON schema, and the Service Archive (SAR) file. You will accomplish those steps in this section. Here's an *overview* of the process:



Notes:

1. Relatively simple JCL that invokes `BPXBATCH` to run the `baqls2js` shell script.
2. The `baqls2js` shell script is what does the transform from `COPYBOOK` to JSON, bind and SAR files.
3. We will show the parameters to `baqls2js` being supplied in a separate data set. Some parameters can become long when full path information is provided. Rather than try to force that within the 80-character limit of JCL, we will show the parameters in a separate FB 256 PDS member[23].
4. The Catalog Manager sample provides three `COPYBOOK` members in the CICS sample library. You will point to that library and members as input to this process.
5. The `baqls2js` will produce the output JSON, bind and SAR files to the location specified:

Bind Files	These are binary format files that contain information about the data fields present and how the JSON fields relate to the COMMAREA fields. z/OS Connect EE V2.0 reads these bind files and uses the information contained in them to perform the data conversion between JSON and COMMAREA.
JSON Schema	These provide information on how the JSON is to be formatted to match the fields in the COMMAREA.
SAR Files	SAR files are ZIP-format files produced by the `baqls2js` utility that contain both JSON schema files and meta-data related to the service. The SAR files are used as input to the z/OS Connect EE V2.0 API Editor for it to understand the service and data fields present.

Do the following:

☐ Create a directory called `dataXform` under your server path:

`/var/zosconnect/servers/<server_name>/dataXform`

Then create three subdirectories under that:

`/var/zosconnect/servers/<server_name>/dataXform/json`
`/var/zosconnect/servers/<server_name>/dataXform/bind`
`/var/zosconnect/servers/<server_name>/dataXform/sars`

The output from `BAQLS2JS` will be placed into those directories.

☐ The ID under which `BAQLS2JS` runs (very likely your TSO ID, as `BAQLS2JS` will run as a submitted JCL batch job) will need write access to those directories. For now, `chmod` the directories `777` for maximum flexibility:

```
cd /var/zosconnect/servers/<server_name>
chmod -R 777 ./dataXform
```

☐ Allocate a small FB 256 PDS to hold the input parameters.

☐ Create a member in that PDS named `INQCAT` (for "Inquire Catalog"). Make sure `NUMBER OFF` and `CAPS OFF` is set in the profile for the member. Supply the following parameters[24]:

```
PGM /<install_path>/bin/baqls2js    1
PDSLIB=hlq.SDFHSAMP     2
REQMEM=DFH0XCP3    3
RESPMEM=DFH0XCP3      4
MAPPING-LEVEL=4.0
CHAR-VARYING=COLLAPSE
JSON-SCHEMA-REQUEST=/<json_path>/inquireCatalog_request.json     5
JSON-SCHEMA-RESPONSE=/<json_path>/inquireCatalog_response.json   6
```

23 The 256 value is arbitrary; it is simply a number big enough so a long parameter can easily fit.
24 The Knowledge Center article that describes this utillity and provides a reference for the parameters is found here: http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/rwlp_zconnect_conv_datatrans.html

```
LANG=COBOL
LOGFILE=/<log_path>/inquireCatalog.log        7
PGMINT=COMMAREA
PGMNAME=DFH0XCMN
WSBIND=/<bind_path>/inquireCatalog.wsbind     8
SERVICE-ARCHIVE=/<sar_path>/inquireCatalog.sar    9
SERVICE-NAME=inquireCatalog          10
```

Notes:

1. Update with the install path to the `/bin` directory of your z/OS Connect EE V2.0 install.
2. Update with the high-level qualifier for the `**.SDFHSAMP` data set where the sample program data members are kept.
3. Point to the request member for Inquire Catalog (which is `DFHX0CP3`).
4. Point to the response member for Inquire Catalog (which is the same: `DFHX0CP3`).
5. Supply the output location and file name for the *request* JSON schema. The path is: `/var/zosconnect/servers/<server_name>/dataXform/json`

   Note: z/OS Connect EE V2.0 will require file name to match the service name. This is for the Inquire Catalog service. Look back on page 25 you will see we planned the service name for that to be `inquireCatalog`. Therefore, the request JSON schema is that name appended with `_request.json`.
6. Supply the output location and file name for the *response* JSON schema. The name must match the service name, therefore: `inquireCatalog_response.json`. The path location is the same as the request schema: `/var/zosconnect/servers/<server_name>/dataXform/json`
7. Supply the output location and file name for the log file of this run. This can go anywhere and be named whatever you wish.
8. Supply the output location and file name for the WSBIND[25] file. The name must match the service name, therefore: `inquireCatalog.wsbind`. The path location is: `/var/zosconnect/servers/<server_name>/dataXform/bind`
9. Supply the output location and file name for the Service Archive (SAR) file. This should match the service name, therefore: `inquireCatalog.sar`. The path location is: `/var/zosconnect/servers/<server_name>/dataXform/sars`
10. Provide the service name: `inquireCatalog`.

☐ Save the member.

☐ In any FB 80 data set of your choosing, create a member called `INQCAT` and create the the JCL and the conversion parameters based on that model. See notes that follow.

```
//BAQLS2JS JOB (),'ZOS CONNECT EE',REGION=0M,NOTIFY=&SYSUID
//*
//ASSIST EXEC PGM=BPXBATCH
//STDPARM DD DSN=<parm_dataset>(INQCAT),DISP=SHR
//STDOUT DD SYSOUT=*    1          2
//STDERR DD SYSOUT=*
//STDENV DD *
JAVA_HOME=/usr/lpp/java/J8.0_64   3
//
/*
```

Notes:

1. Supply the name of the FB 256 data set name you allocated earlier.

25 APAR PI55019 fixes a problem where the suffix *was required* to be `WSBIND`. With the APAR other suffixes are permitted. If you don't have that APAR, then make sure the bind file suffix is `*.WSBIND` and it should work.

2. Supply the parameter member name for this execution of the job. For Inquire Catalog the member you created was `INQCAT`.
3. Point to the location for the 64-bit IBM Java SDK.

- ☐ Submit the job. If RC=0, look in the output locations for the files produced. If other than RC=0, check the log file and job output and correct errors and submit again.
- ☐ Do the same process for `inquireSingle`:
  - ○ Create another parameter input member called `INQSNG`
  - ○ Copy the `INQCAT` member to `INQSNG`
  - ○ Edit `INQSNG` and change all instances of `inquireCatalog` to **`inquireSingle`**
  - ○ Change the `REQMEM` and `RESPMEM` values to **`DFH0XCP4`**
  - ○ In your job JCL, change the input member value from `INQCAT` to **`INQSNG`**
  - ○ Submit the job. Look for RC=0.
- ☐ Do the same process for `placeOrder`:
  - ○ Create another parameter input member called `PLORDR`
  - ○ Copy the `INQCAT` member to `PLORDR`
  - ○ Edit `PLORDR` and change all instances of `inquireCatalog` to **`placeOrder`**
  - ○ Change the `REQMEM` and `RESPMEM` values to **`DFH0XCP5`**
  - ○ In your job JCL, change the input member value from `INQSNG` to **`PLORDR`**
  - ○ Submit the job. Look for RC=0.
- ☐ At this point you should have:

```
/var/zosconnect/servers/<server_name>/dataXform
├──/bind
│    o inquireCatalog.wsbind
│    o inquireSingle.wsbind
│    o placeOrder.wsbind
├──/json
│    o inquireCatalog_request.json
│    o inquireCatalog_response.json
│    o inquireSingle_request.json
│    o inquireSingle_response.json
│    o placeOrder_request.json
│    o placeOrder_response.json
└──/sars
     o inquireCatalog.sar
     o inquireSingle.sar
     o placeOrder.sar
```

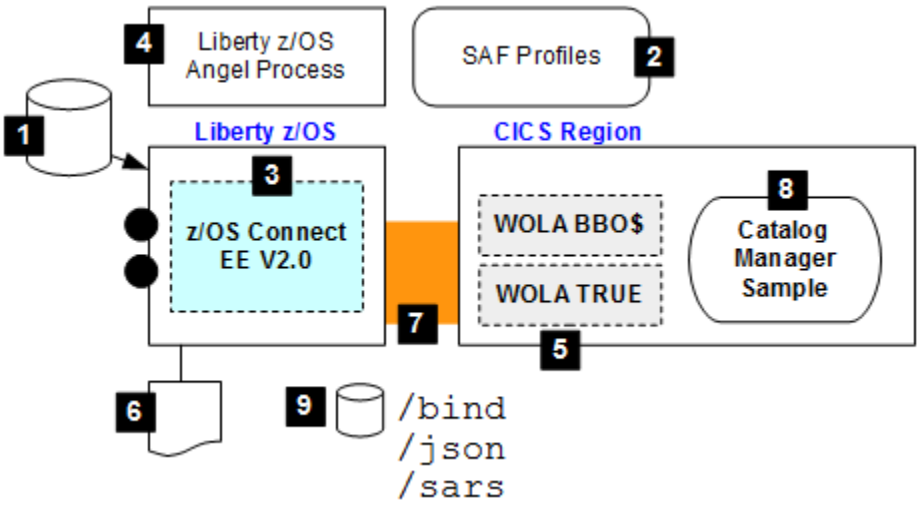
- ☐ Recursively `chown` and `chmod` the output directories so your Liberty z/OS server ID has access to them:

```
cd /var/zosconnect/servers/<server_name>
chown -R <serverID>:<serverGroup> ./dataXform
chmod -R 750 ./dataXform
```

Note: At this point your TSO ID may or may not have access to write to those directories. That means if you run the `BAQLS2JS` job again it may fail due to permissions. Adjust the permission of the directories based on your local UNIX security practices.

Mid-point status check

Here's a picture illustrating your status at this point:



Notes:

1. z/OS Connect EE V2.0 installed via SMP/E
2. SAF profiles in support of z/OS Connect and WOLA created (`STARTED`, `SERVER`, `CBIND`)
3. Liberty z/OS server created and started as a z/OS started task
4. Liberty z/OS Angel process started because required for WOLA
5. WOLA TRUE and Link Server Task installed into CICS region
6. Server's `server.xml` updated with basic security in support of z/OS Connect EE V2.0 as well as the definitions in support of WOLA
7. WOLA registration from CICS region into Liberty z/OS server validated
8. The Catalog Manager sample program installed into the CICS region
9. The services and API planned, and the JSON, WSBIND and SAR files generated based on the CICS sample program COPYBOOK members.

That is quite a bit of set up work in preparation for configuring z/OS Connect EE V2.0. The next step is to configure the z/OS Connect EE V2.0 service definition and data transformation elements into the `server.xml` and validate them.

After that will come the use of the z/OS Connect EE V2.0 API Editor and the creation of the API.

Setup of service definitions for catalog manager sample in server.xml

Your `server.xml` should look like this, based on earlier updates:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>zosconnect:zosconnect-2.0</feature>
        <feature>appSecurity-2.0</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">
        <properties.ola />
    </connectionFactory>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="<port>"
                  httpsPort="<port>" />

</server>
```

You are ready to update with z/OS Connect service definitions and the data transform XML. Do the following:

- ☐ Update the `server.xml` with the XML highlighted here in yellow:

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>zosconnect:zosconnect-2.0</feature>
        <feature>appSecurity-2.0</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>

    <zosconnect_zosConnectService id="inquireCatalogService"
        invokeURI="/inquireCatalog"
        serviceName="inquireCatalog"
        dataXformRef="xformJSON2Byte"
        serviceDescription="List items in the catalog"
        serviceRef="wolaCICS" />

    <zosconnect_zosConnectService id="inquireSingleService"
        invokeURI="/inquireSingle"
```

```
        serviceName="inquireSingle"
        dataXformRef="xformJSON2Byte"
        serviceDescription="Inquire on an item in the catalog"
        serviceRef="wolaCICS" />

    <zosconnect_zosConnectService id="placeOrderService"
        invokeURI="/placeOrder"
        serviceName="placeOrder"
        dataXformRef="xformJSON2Byte"
        serviceDescription="Order an item from the catalog"
        serviceRef="wolaCICS" />

    <zosconnect_zosConnectDataXform id="xformJSON2Byte"
1       bindFileLoc="/var/zosconnect/servers/<server_name>/dataXform/bind"
        bindFileSuffix=".wsbind"
        requestSchemaLoc="/var/zosconnect/servers/<server_name>/dataXform/json"
        responseSchemaLoc="/var/zosconnect/servers/<server_name>/dataXform/json"
        requestSchemaSuffix=".json"
        responseSchemaSuffix=".json">
    </zosconnect_zosConnectDataXform>

    <zosconnect_localAdaptersConnectService id="wolaCICS"
        registerName="ZCONNREG"  2
        serviceName="DFH0XCMN"
        connectionFactoryRef="wolaCF" />

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

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

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="<port>"
                  httpsPort="<port>" />

</server>
```

Notes:

1. Update *<server_name>* to reflect your actual value.
2. Set `registerName=` to the value used by CICS when it registered into your z/OS Connect EE V2.0 server. Here we illustrate `ZCONNREG` because earlier we showed the `BBOC START_SRVR` command using that value. But in your actual environment the `BBOC START_SRVR` *may* have been issued with a different `RGN=` value. Match the value here in the XML with whatever the actual value used by CICS when it registered.

☐ Save the file.

☐ Look in the `messages.log` file for the server. You should see something like this:

```
CWWKG0016I: Starting server configuration update.
CWRLS0010I: Performing recovery processing for local WebSphere server (Server).
  (Messages indicating creation of translation logs)
WTRN0135I: Transaction service recovering no transactions.
CWWKG0017I: The server configuration was successfully updated in 0.993 seconds.
```

At this point you have service definitions to match the Catalog Manager sample in CICS, but you do not yet have an API created and deployed. But you can still test your environment to prove connectivity and access to programs over WOLA.

☐ For validation purposes we will show using a browser REST client plugin. The browser you use is up to you. The URLs for Chrome and Firefox are here:

Chrome	https://chrome.google.com/webstore/category/apps Search for "Advanced REST Client"
Firefox	https://addons.mozilla.org/en-us/firefox/addon/restclient/

Install the REST client of your choosing into your browser.

☐ Open a normal browser tab (not the REST client) and send the following URL:

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

You should receive a certificate challenge because the server certificate is signed by a CA that is not known to the browser. Accept the challenge.

Note: This is the key to this step – the REST clients typically do not handle an untrusted CA well; by manually establishing the trust it allows the REST client in another tab to establish the secure connection to z/OS Connect EE V2.0.

You will then receive the basic authentication prompt. Supply the ID (**`Fred`**) and password (**`fredpwd`**). You should receive in return a difficult-to-read[26] string of JSON that represents the three services you now have deployed:


```
{"zosConnectServices":[{"ServiceName":"placeOrder","ServiceDescription":"Order an item from the
catalog","ServiceProvider":"WOLA-1.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect
/services/placeOrder"},{"ServiceName":"inquireSingle","ServiceDescription":"Inquire on an item in the
catalog","ServiceProvider":"WOLA-1.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect
/services/inquireSingle"},{"ServiceName":"inquireCatalog","ServiceDescription":"List items in the
catalog","ServiceProvider":"WOLA-1.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect
/services/inquireCatalog"}]}
```


☐ Open the REST client in a *new* tab. Then use the REST client to send the following request:

Header:	`Content-Type` | `Application/json`
Method:	`POST`
URL:	`https://<host>:<port>/inquireCatalog`

JSON:	`{`
	`  "DFH0XCMNOperation": {`
	`  "ca_request_id": "01INQC",`
	`  "ca_inquire_request": {`
	`  "ca_list_start_ref" : "0010",`
	`  "ca_item_count" : "100"`
	`  }`
	`  }`
	`}`

You should receive a "200 OK" message along with a lengthy set of JSON that represents the items in the catalog. This verifies your `inquireCatalog` service.

☐ Stay in the REST client and verify the `inquireSingle` service:

Header:	`Content-Type` | `Application/json`
Method:	`POST`
URL:	`https://<host>:<port>/inquireSingle`

26 The browser doesn't understand how to format the JSON, so it simply displays it as it received it.

JSON:	<pre>{
	  "DFH0XCMNOperation": {
	    "ca_request_id": "01INQS",
	    "ca_inquire_single": {
	       "ca_item_ref_req" : "0010"
	  }
	 }
	}</pre>

☐ Finally, verify the `placeOrder` service:

Header:	`Content-Type` | `Application/json`
Method:	`POST`
URL:	`https://<host>:<port>/placeOrder`

JSON:	<pre>{
	  "DFH0XCMNOperation": {
	    "ca_request_id": "01ORDR",
	    "ca_order_request": {
	       "ca_userid": "Edward",
	       "ca_charge_dept": "IBM",
	       "ca_item_ref_number": "0010",
	       "ca_quantity_req": "1"
	   }
	  }
	}</pre>

Verifying the services prior to composing and deploying APIs provides a degree of assurance the basics of z/OS Connect EE V2.0 are operational. You're ready to compose and deploy an API.

API composition in z/OS Connect EE V2.0 API Editor

This section will take you through the step-by-step actions required to create the API to retrieve the details for a specific item. Do the following:

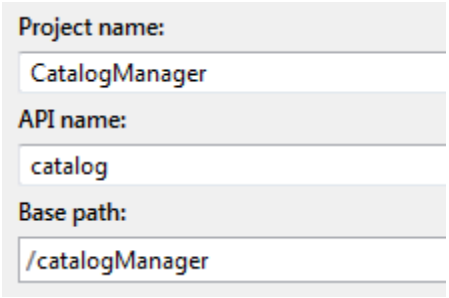
☐ Download *in binary* the `inquireSingle.sar` file to your workstation from:
`/var/zosconnect/servers/<server_name>/dataXform/sars/`

☐ Open your Eclipse program and switch to the "z/OS Connect Enterprise Edition" perspective[27].

☐ Open the Project Explorer view if it is not already open.

☐ Then: *File → New → z/OS Connect EE API Project*

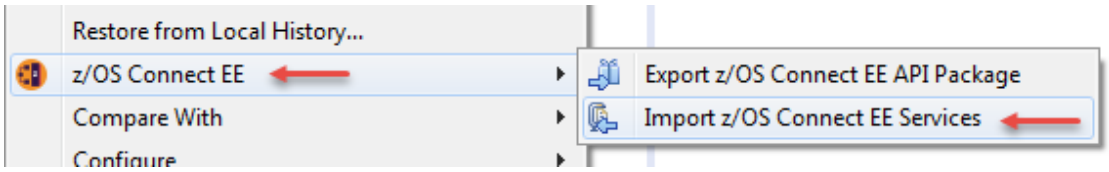
☐ Provide the Project name, API name and Base path as shown:



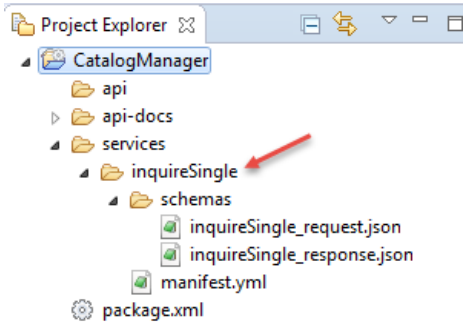
27 Depending upon the environment in which you installed your z/OS Connect EE API Editor, your particular Eclipse may have several different "perspectives" (arrangement of the windows). The instructions and the screen captures in the following pages presume that you are in the z/OS Connect EE perspective. You can always select your "perspective" by selecting *Window -> Open Perspective -> Other*... and in the dialog that follows, select "z/OS Connect Enterprise Edition" and press OK.

Note: the "Base path" is used by the REST client when forming the URI for the request. The base path is appended with the API path (you will see that in an upcoming step) to form the URI that is sent in to z/OS Connect EE V2.0.

- ☐ Click "Finish"
- ☐ Right click on "CatalogManager" in the project explorer view, and then *z/OS Connect EE → Import z/OS Connect EE Services*:



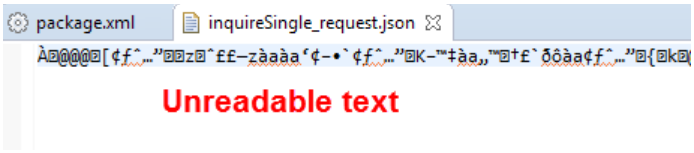
- ☐ Click on the "File System" button, then navigate to where you stored the SAR file. Select the `inquireSingle.sar` file and click the "Open" button. Then click the "OK" button.
- ☐ At this point you should see the `inquireSingle` service and the two JSON schema files in the Project Explorer:



This next step is very important. It determines whether or not your copy of z/OS Connect EE V2.0 on the host system has APAR PI53740[28]. If not, we can work around the problem in the API Editor. But first we must determine if there is a problem.

- ☐ Right click on one of the JSON schema files, and select *Open With → Text Editor*.

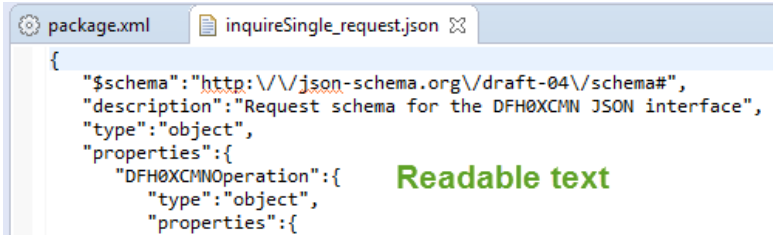
  If you see this:



  Then you have a problem. The creation of the SAR file on the z/OS system packaged the JSON schema files as EBCDIC, and the API Editor requires ASCII.

  If you have this symptom, go to "Workaround to the problem of EBCDIC JSON schema in SAR files" on page 110, then return to this spot.

  But if you see something like this:



28 http://www-01.ibm.com/support/docview.wss?crawler=1&uid=isg1PI53740

Then you have ASCII-based JSON schema files and may proceed with the API composition.

- ☐ In the `package.xml` open tab, provide a path of **`/items/{itemID}`**

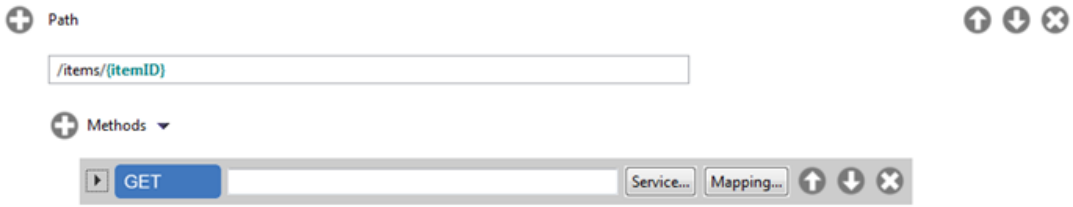
Note: this "path" is appended to the "base path" from earlier to form the URI the REST client sends in. A path parameter (`{itemID}`) is also indicated. That means for this API a REST client would send in something like:

`/catalogManager/items/10`

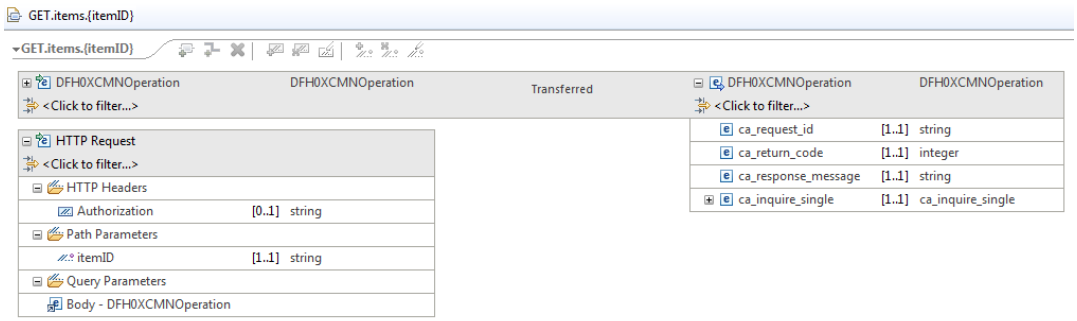
where "catalogManager" is the base path, "items" is the API path, and "10" is the path parameter that will be mapped to a COMMAREA field in one of the upcoming steps.

- ☐ The method planned for the path to get the details of an item is `GET` [29], so the other methods can be removed from the view. Click the X symbol for `POST`, `PUT` and `DELETE`:

Result:

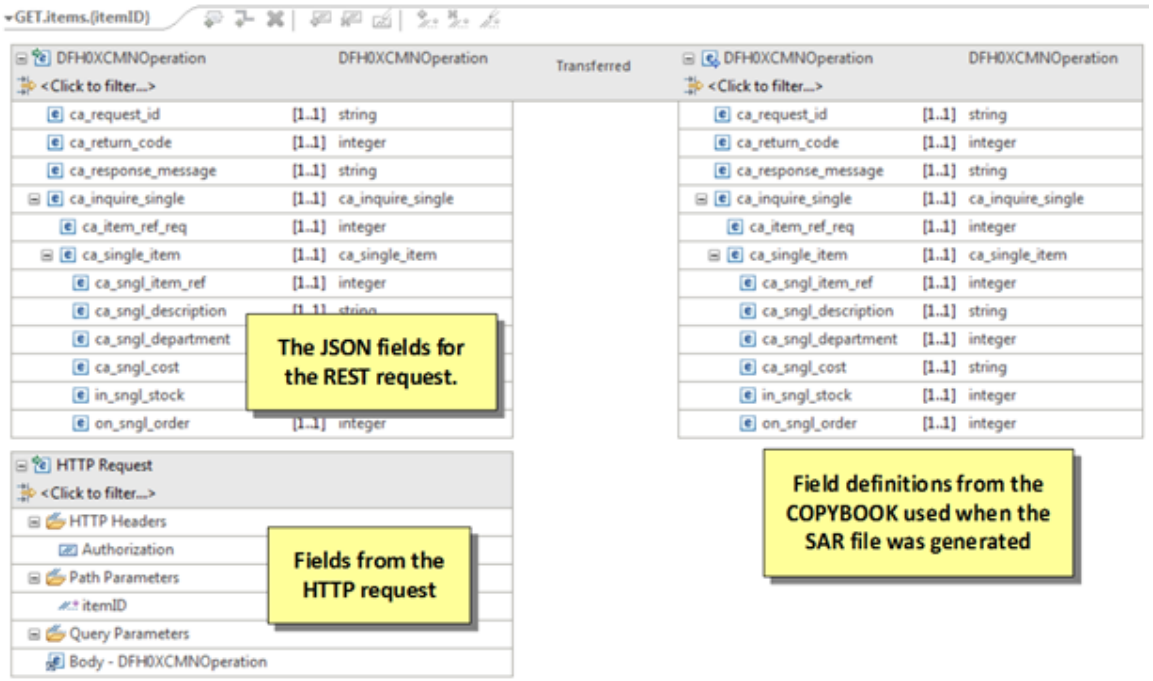


- ☐ Click the "Service" button, select `inquireSingle` from the list of services (there should be only one), and then "OK."
- ☐ Click *File* → *Save*. This is required before the mapping can be started.
- ☐ Click on the "Mapping" button, then select "Open Request Mapping." You should see:



29 See "A discussion of why the Catalog Manager API was designed as it was" on page 95 for why a GET verb is used for this.

☐ Right click somewhere in the window and select "Expand All." You should see something like what's shown here. Please take a moment to read the notes that follow.



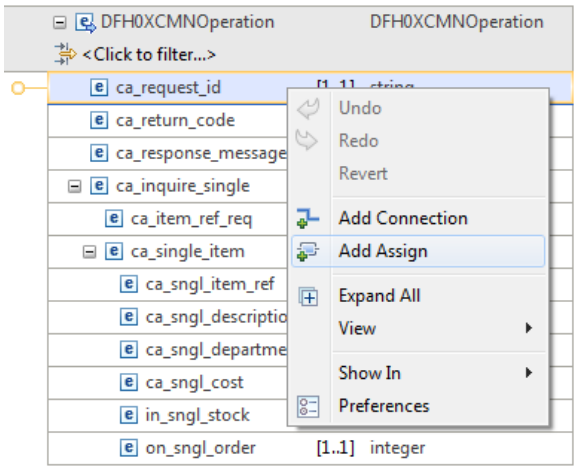
Notes:

- We added the yellow text boxes. You will not see those.
- For a **request** map, three sections of fields are represented:

Right side	This section represents the COMMAREA fields seen in the COPYBOOK when the `baqls2js` utility was run to produce the SAR files.
Upper-left side	This section represents the fields exposed to the REST client. *Initially* there is a one-to-one mapping; that is, *intially* all the COMMREA fields are exposed to the REST client. But you control which fields are exposed and which are hidden by your actions in the API Editor.
Lower-left side	This section represents the information fields available on the HTTP request, such as authorization headers, path parameters, query parameters, or information in the body of a request.

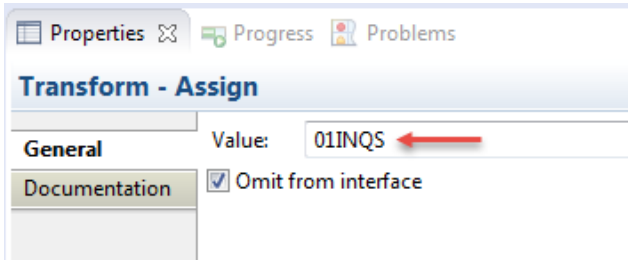
- The general approach is to start with the right side COMMAREA fields and map them according to your requirements. That is what you will see in the steps that follow: you will assign a static value to one of the right-side fields, you will "remove" (hide) several fields so they are not exposed to the REST client, and you will "move" the path parameter value to one of the COMMAREA fields.

☐ On the right side of the screen, right click on `ca_request_id` and select "Add Assign":



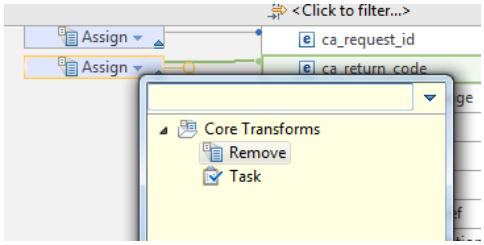
Note: based on our knowledge of the Catalog Manager program, we know the `ca_request_id` field for "inquire single" is **`01INQS`**. So we are going to assign that field the static value `01INQS` in the API Editor. z/OS Connect EE V2.0 will populate that field with `01INQS` whenever this API is called.

☐ Left-click the "Assign" box, then go down to the "Properties" tab (at the bottom of the screen) and supply a value of **`01INQS`** for this field:

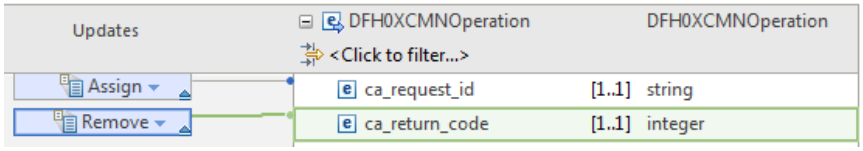


That assigns the static value to the field. The checkbox "Omit from interface" means the REST client does not see this field as part of the API.

☐ On the right side of the screen, right-click on **`ca_return_code`** and select "Add Assign". Then click the down arrow on the new "Assign" element that appears. Select the "Remove" option:

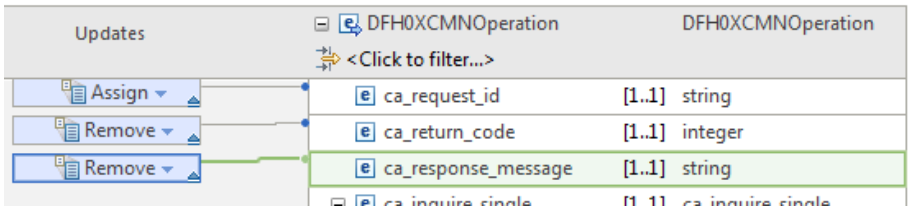


The "Assign" element should now appear as a "Remove" element:



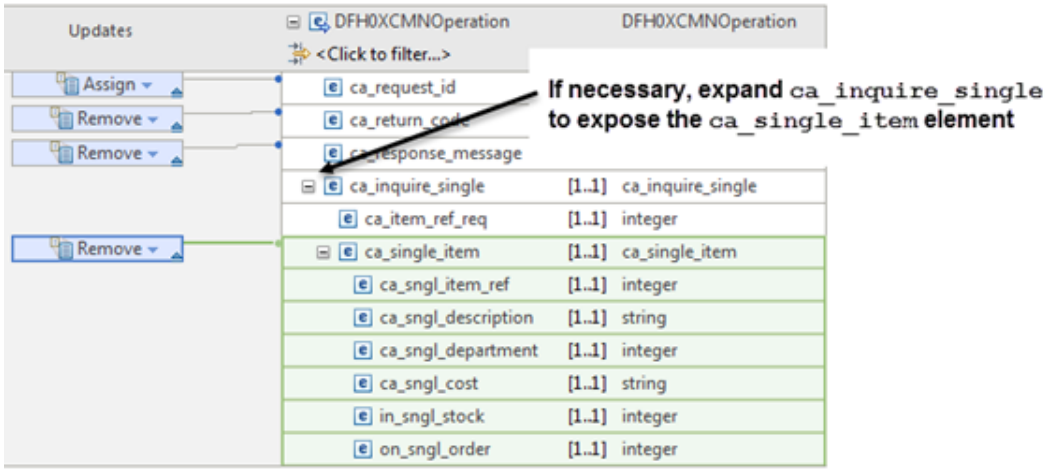
Note: This removes the field from the left-side REST client view of the API. That field is not required as input. By "removing" it we simplify the REST API by eliminating that field from the REST interface.

☐ Do the same for **`ca_response_message`**:



The `ca_response_message` field is removed because it is not relevant to a request for details on an item in the catalog. By removing it we simplify the API further.

☐ Do the same for **`ca_single_item`** :

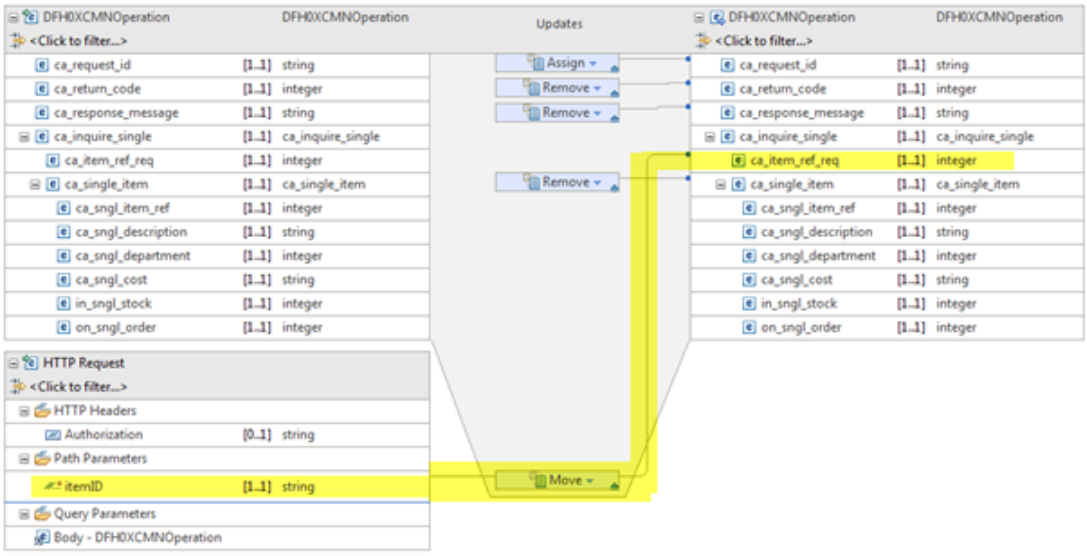


These fields in the COPYBOOK are used for the *response*. This is the *request* map. So we remove these fields to simplify the API further still.

The next step is to build a "Move" connection between the path parameter `itemID` on the left to `ca_item_ref_req` on the right.

☐ Right-click on the `itemID` under "Path Parameters" on the left and select "Add Connection." A connection line will appear and will follow your mouse.

☐ Move your mouse over to `ca_item_ref_req` on the right side, then left-click your mouse. That will build the connection from `itemID` to `ca_item_ref_req` and add a "Move" element:

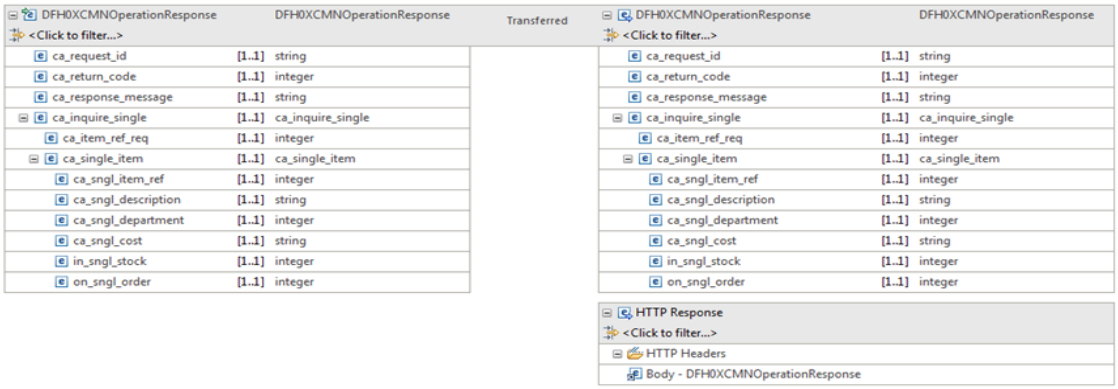


Note: At this point all the fields on the right side of the request map are either assigned with a static value, removed from the REST client interface, or the value is supplied by a move from the HTTP request information. That means the REST API is very simple: the verb is GET, there is no JSON requirement, and the single input value (for the item ID) is passed in as a path parameter.

- ☐ *File* → *Save*, then close the request mapping tab.

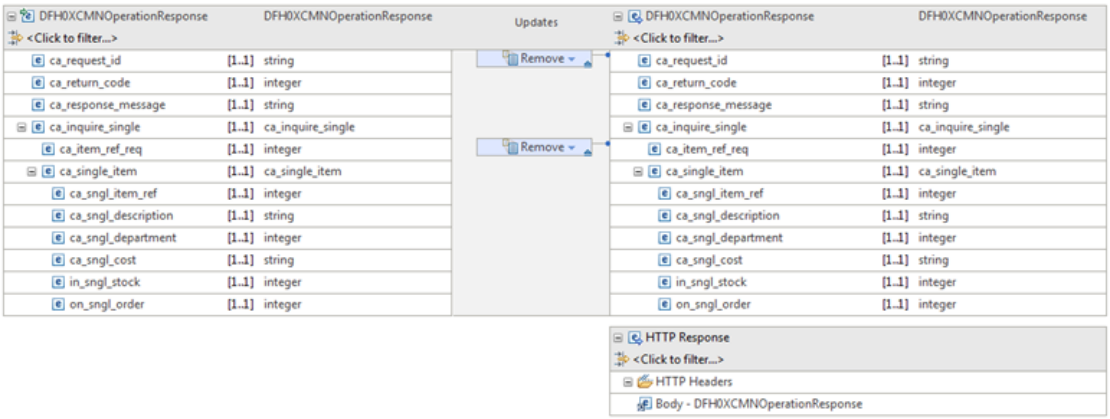
That was the *request* mapping. We also need to map the *response*. That's next.

- ☐ In the `package.xml` tab, click the "Mapping" button again, and select "Open Response Mapping."
- ☐ Right click somewhere in the window and select "Expand All." You should see this:



Note: this map allows us to determine what information is to be sent *back* to the REST client. The right side represents the COMMAREA fields, and the left side represents the fields that will be provided in the JSON response. In the steps that follow you will "remove" two of the fields from the response and allow the other information to flow back to the REST client in the JSON object that is returned.

- ☐ On the right side of the screen, right-click on **`ca_request_id`** and select "Add Assign." Then select the down arrow on the Assign element and select "Remove." We do this because we do not want to return `ca_request_id` in the JSON response message.
- ☐ Do the same for **`ca_item_ref_req`**, which may require you to expand ca_inquire_single first. Your editor screen should look like this:



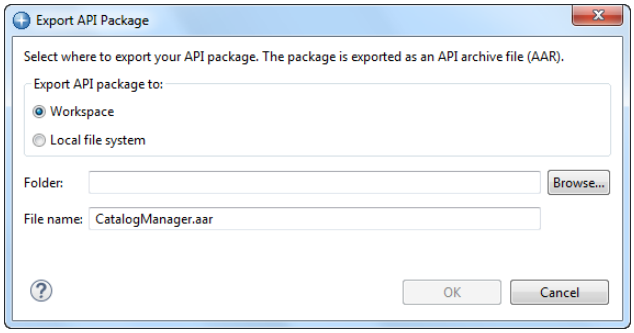
Note: we have removed two fields and allowed the rest to map over to the JSON that will flow back to the client.

- ☐ *File* → *Save*. Then close the response mapping tab.
- ☐ Close the `package.xml` tab.

Deployment of APIs into z/OS Connect EE V2.0

Your API is ready for export from the API Editor and upload to z/OS. The API has only the `inquireSingle` service represented, but for gaining experience with the API Editor and deploying APIs, it was sufficient.

- ☐ In the API Editor, right-click on the CatalogManager project, then select "z/OS Connect EE," then "Export z/OS Connect EE API Package."
- ☐ Select "Local file system," then specify the folder where the AAR package will go, then click on OK:



- ☐ Upload the `CatalogManager.aar` file *in binary* to some location on your z/OS system.
- ☐ Open a Telnet, SSH or OMVS window to your system.

> **Note:** The deployment process will create directories under the Liberty server's `/resources` directory, so the ID you log in with (or su to) must have authority to create directories there.

- ☐ Change directories to the `/<install_path>/bin` directory for z/OS Connect EE V2.0.
- ☐ Make sure JAVA_PATH is set and points to your 64-bit Java:

  `export JAVA_HOME=path_to_your_64-bit_Java_SDK`

- ☐ Issue the following command[30] [31] *as one line*:

  `./apideploy -deploy -a /<path>/CatalogManager.aar -p`
  `/var/zosconnect/servers/<server_name>/resources/zosconnect/apis`

  Where:

  - *<path>* is where your uploaded `CatalogManager.aar` file exists
  - *<server_name>* is your server name

  Note: The target path directories will be created if they do not exist.

  You should see:

```
BAQD0007I: The CatalogManager API is created successfully in the server API
path /var/zosconnect/servers/<server_name>/resources/zosconnect/apis/catalog.
Restart the server to deploy the API.
```

- ☐ Check the ownership of the directories for the deployed API, starting at:

  `/var/zosconnect/servers/<server_name>/resources/zosconnect/`

  The ownership may be that of the ID under which the `apideploy` command was run. That may be okay provided your Liberty z/OS server ID has `READ` to those directories and files.

  If you wish, you can set the ownership back to your Liberty z/OS server ID and group:

30 The Knowledge Center article on the apideploy utility is found here: http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/api_deploy_package.html

31 You can put this in a shell script as well. Be sure to set `JAVA_HOME` variable and provide path to `apideploy` location.

```
cd /var/zosconnect/servers/<server_name>/resources
chown -R <serverID>:<serverGroup> ./zosconnect
```

- ☐ From CICS, stop the link server task[32]:

  `BBOC STOP_SRVR RGN=ZCONNREG`

  Where the `RGN=` is the registration name you used earlier.

- ☐ Stop your z/OS Connect EE V2.0 server.
- ☐ Update the `server.xml` and add the following:

```
:  (XML before this point removed to save space in document)
 <authorization-roles id="zos.connect.access.roles">
     <security-role name="zosConnectAccess">
         <user name="Fred"/>
     </security-role>
 </authorization-roles>

 <zosconnect_zosConnectAPIs location="">
     <zosConnectAPI name="catalog" />
 </zosconnect_zosConnectAPIs>

 <zosconnect_zosConnectService id="inquireCatalogService"
     invokeURI="/inquireCatalog"
     serviceName="inquireCatalog"
     dataXformRef="xformJSON2Byte"
     serviceDescription="Inquire on an item in the catalog"
     serviceRef="wolaCICS" />
:  (XML after this point removed to save space in document)
```

Notes:

- The `location=""` means the default location for APIs is used, which is under the server's `/resources/zosconnect/apis` directory. That's the same directory you deployed the API to using the `apideploy` utility.
- The `name=` attribute matches the value you supplied for "API Name" in the API Editor dialog box when you created the API. It also matches the directory name under `/resources/zosconnect/apis` that was created when this API was deployed.

- ☐ Save the `server.xml` file.
- ☐ Restart the z/OS Connect EE V2.0 Liberty server.
- ☐ From the CICS region, restart the link server task and re-establish the WOLA registration into the z/OS Connect EE V2.0 server:

```
BBOC START_SRVR RGN=ZCONNREG DGN=group NDN=name2 SVN=name3
                                    SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y
```

Where *group*, *name2* and *name3* matches the values found in your `server.xml` file.

Validation

This is very similar to what you did earlier to validate the services. The difference is you will use the HTTP verb and URI pattern defined in the API.

- ☐ Open a normal browser tab (not the REST client) and send the following URL:

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

  If you receive a certificate challenge it is because the server certificate is signed by a CA that is not known to the browser. Accept the challenge if you receive one.

32 You are about to take down your Liberty z/OS server, which will break any existing WOLA registrations. Stopping the link server task brings down the registration, and provides a clean state to re-establish the registration once the server is back up.

Note: This is only needed because we're using the default security setup for z/OS Connect EE V2.0. In the "real world" you'd use a well-known CA to sign the server certificate, and the REST client would have knowledge of the well-known CA.

You will then receive the basic authentication prompt. Supply the ID (**Fred**) and password (**fredpwd**). You should receive in return a difficult-to-read[33] string of JSON that represents the three services you now have deployed.

☐ Open the REST client in a *new* tab. Then do the following:

Method:	GET
URL:	https://*<host>*:*<port>*/catalogManager/items/0010
JSON:	*No JSON is used; the verb is GET and the itemID is carried as a path parameter.*

You should receive a "200 OK" message along with JSON that represents the item 0010, which is the 24-pack of black ball point pens.

☐ Finally, retrieve the "API docs" for your deployed API:

GET https://*<host>*:*<port>*/catalogManager/**api-docs**

In the response body you should get back a Swagger 2.0 document that represents the API. It will have only inquireSingle represented because that's all that's defined in the API you deployed.

(Optional) Deploy API with all three services

The API you created contained only the `inquireSingle` service.

If you would like to see all three services, then see "Importing the supplied API project into Eclipse" on page 94. That provides details on downloading and importing an Eclipse project with the services `inquireCatalog`, `inquireSingle` and `placeOrder` defined.

Do the following:

☐ Delete the existing project from Eclipse[34].

☐ Import the full project using the instructions provided on page 94.

☐ Explore the API and see how things are mapped. You do not need to change anything.

☐ Export the API Archive (AAR) file just as you did before.

☐ Deploy the API just as you did before, but with a `-w` switch on the end to update the API:

```
./apideploy -deploy -a /<path>/CatalogManager.aar -p
      /var/zosconnect/servers/<server_name>/resources/zosconnect/apis -w
```

You should see:

```
BAQD0009I: The catalog API is updated in the server API path
/var/zosconnect/servers/<server_name>/resources/zosconnect/apis/catalog.
```

☐ Stop and restart the z/OS Connect EE V2.0 server.

☐ Restart the WOLA connection from the CICS region.

☐ Test the API:

List the details of a specific item

Method:	GET
URL:	https://*<host>*:*<port>*/catalogManager/items/0010
JSON:	*No JSON is used; the verb is GET and the itemID is carried as a path parameter.*

33 The browser doesn't understand how to format the JSON, so it simply displays it as it received it.
34 The project name CatalogManager is what you used earlier. You can't have two projects by the same name in a given workspace.

List 15 items of the catalog starting at a specific item reference number

Method:	`GET`
URL:	`https://<host>:<port>/catalogManager/items?startItemID=0010`
JSON:	*No JSON is used; the verb is GET and the starting number is a query parameter*

Place an order

Method:	`POST`
Header:	`Content-Type application/json`
URL:	`https://<host>:<port>/catalogManager/orders`

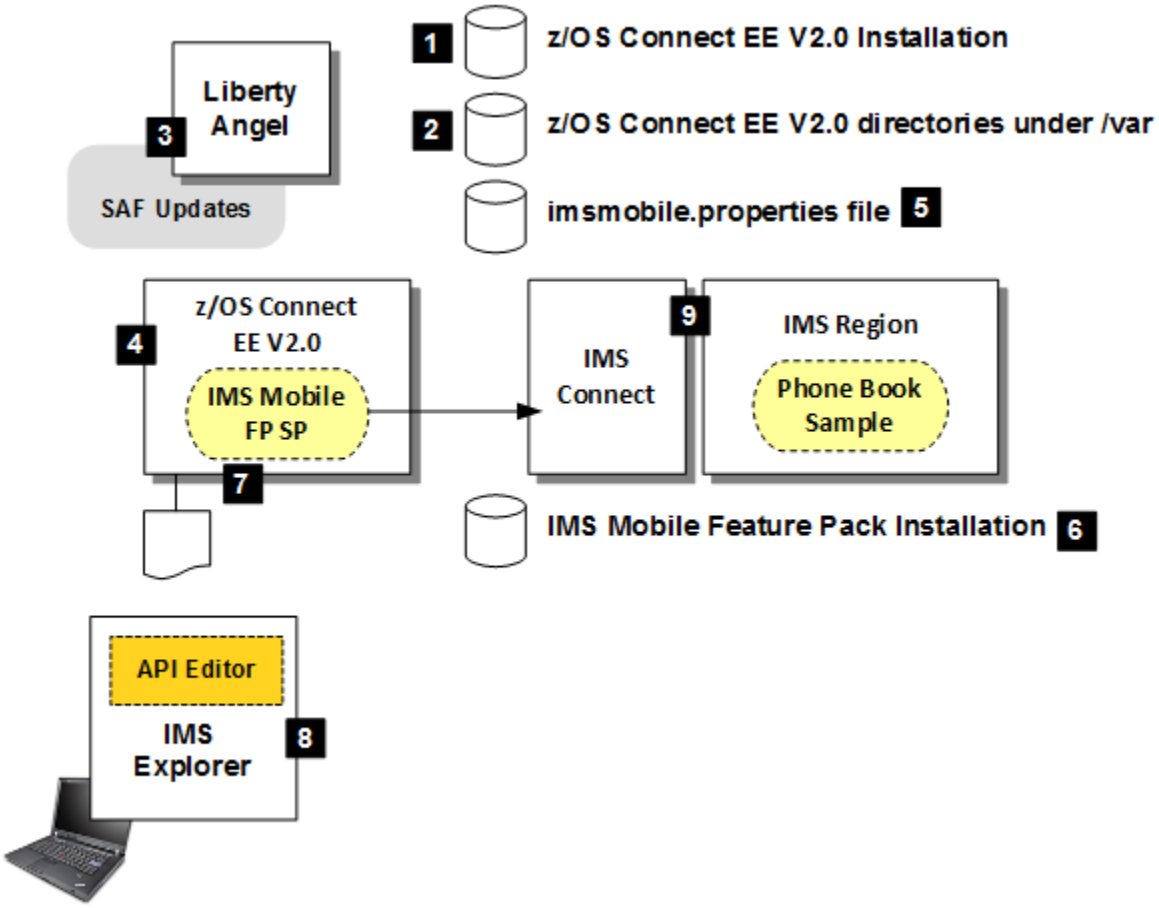
JSON:	`{`
	`  "DFH0XCMNOperation": {`
	`  "ca_order_request":{`
	`  "ca_item_ref_number":10,`
	`  "ca_quantity_req":1`
	`  }`
	`  }`
	`}`

IMS – Service Provider Setup, Validation and the Phone Book Sample

If your primary interest is IMS, you *may* have jumped directly to this section to perform installation and setup. You may have to go back and perform certain setup steps from earlier. We will offer specific instructions here which sections to visit and perform.

Overview of installation and setup artifacts in support of IMS

Review the following picture and the notes that follow. Then follow the step-by-step instructions that come after.



Notes:

1. The z/OS Connect EE V2.0 installation is the same regardless of the backend system. This we covered under "SMP/E install of z/OS Connect EE V2.0 and post-install setup" on page 7. The result is a file system mounted at a mount point you set during installation.
2. z/OS Connect EE V2.0 uses a file structure under `/var`. This is created by running the zconsetup shell script as instructed under "SMP/E install of z/OS Connect EE V2.0 and post-install setup" on page 7.
3. z/OS Connect EE V2.0 when used with IMS does not require as many SAF updates as is required when used with CICS and WOLA[35]. But *some* SAF profiles *are* needed. Those will be spelled out in the section below.
4. A Liberty z/OS server must be set up before the configuration of the IMS connector can be done. We covered that under "Server creation and initial validation" on page 11.
5. A very simple create and update of a file under the `/var` directory is needed to tell Liberty z/OS about the IMS Mobile feature pack location. The details of this are spelled out below.

35 That's because WOLA requires authorized access, and that implies the Angel Process must be present and key SERVER and CBIND profiles in place. z/OS Connect EE V2.0 and IMS uses a TCP-based connector, not WOLA.

6. The "IMS Mobile Feature Pack" is installed with a simple command that is executed against an install file you upload to your system. The details of this are spelled out below.
7. A few updates to the server.xml of the server are needed to configure the IMS Mobile Feature Pack connector so it may communicate with IMS Connect. This makes the z/OS Connect EE V2.0 server an "IMS Gateway Server" (gateway to IMS Connect and IMS). Details for this are provided below.
8. You will need IMS Explorer to perform configuration updates of your z/OS Connect EE V2.0 server as well as create the data transformation SAR files. The z/OS Connect EE V2.0 API Editor is a plugin to Eclipse, and that can run in IMS Explorer. Details for installing both are provided below.
9. Finally, you will need an IMS region and an instance of IMS Connect to use as your backend. The sample illustrated in this document is the "Phone Book" sample, which is commonly available on IMS systems.

Installation, setup and initial validation of z/OS Connect EE V2.0 and IMS connectivity

Install z/OS Connect EE V2.0 and perform initial setup

See "SMP/E install of z/OS Connect EE V2.0 and post-install setup" starting on page 7 and follow the instructions found there.

SAF updates in support of z/OS Connect EE V2.0 and IMS

The instructions found under "SAF updates in support of z/OS Connect EE V2.0" on page 8 provide details on the SAF profiles needed. The table below summarizes the profiles:

`STARTED`	Used to assign an ID to the Liberty z/OS server started task. This is required regardless of the backend system (CICS or IMS)
`SERVER`	These provide access to authorized services. The connectivity to IMS does not use WOLA, so the WOLA-related SERVER profiles are not needed. The profiles for access to WLM and SAF you may need, depending on how you use your Liberty z/OS server. If acceptable, create the SERVER profiles and have them ready if needed in the future.
`CBIND`	This is used to grant access to an ID attempting to create a WOLA registration into the Liberty z/OS server. For IMS you do not need this.
Angel Process	For basic validation of z/OS Connect EE V2.0 and IMS you will not need this. You may already have an Angel process if you have z/OS 2.1 or higher and z/OSMF. If acceptable, create the Angel Process and have it ready if needed in the future.

Server create and initial validation

See "Server creation and initial validation" on page 11. The initial server setup is the same regardless of the backend system, CICS or IMS. After the server is set up there are different configuration steps for CICS and IMS.

Prepare for installation of IMS Mobile Feature Pack

The IMS Mobile Feature Pack is a set of files installed into a file system directory. z/OS Connect EE V2.0 is made aware of the location with an update to a "properties" file under the `/var` directory[36]. Before you do any of that, however, a little bit of preparation work is in order. Do the following:

☐ Note here where you will install the IMS Mobile Feature pack files:

For example: `/usr/lpp/ims/imses/V3R2/imsmobile`

36 KC article: http://www-01.ibm.com/support/knowledgecenter/SS9NWR_3.2.0/com.ibm.ims.mobilezc32.doc/mobilezc_install.htm

- ☐ If that mount point does not exist, then create it.
- ☐ Create a ZFS file system and mount it at the appropriate mount point. For example:

```
DEFINE -
     CLUSTER -
       ( -
           NAME(IMSES.V3R2.MOBILEFP.ZFS) -
           LINEAR -
           CYL(300 50) -
           VOLUME(*) -
           SHAREOPTIONS(3) -
       )
PROFILE MSGID WTPMSG
MOUNT TYPE(ZFS) +
  MODE(RDWR) +
  MOUNTPOINT('/usr/lpp/ims/imses/V3R2/imsmobile') +
  FILESYSTEM('IMSES.V3R2.MOBILEFP.ZFS')
```

Note: where the file system name is whatever value you wish that conforms to your local naming standards, and the mount point is what you planned and created for this purpose.

- ☐ Finally, create a directory of `/wlp-ext` in the new file system and under the mount point. The path now will look something like this:

  `/usr/lpp/ims/imses/V3R2/imsmobile/wlp-ext`

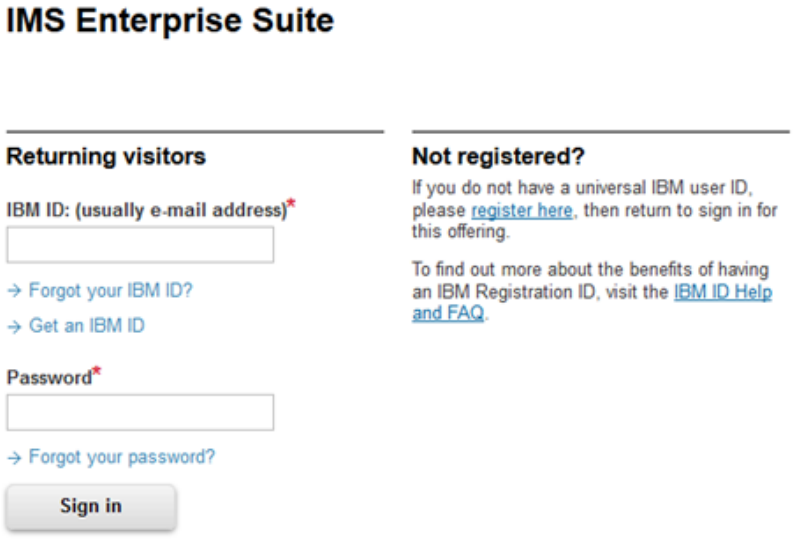
Download the IMS Mobile Feature Pack and IMS Explorer

Do the following:

- ☐ Go to the following URL:

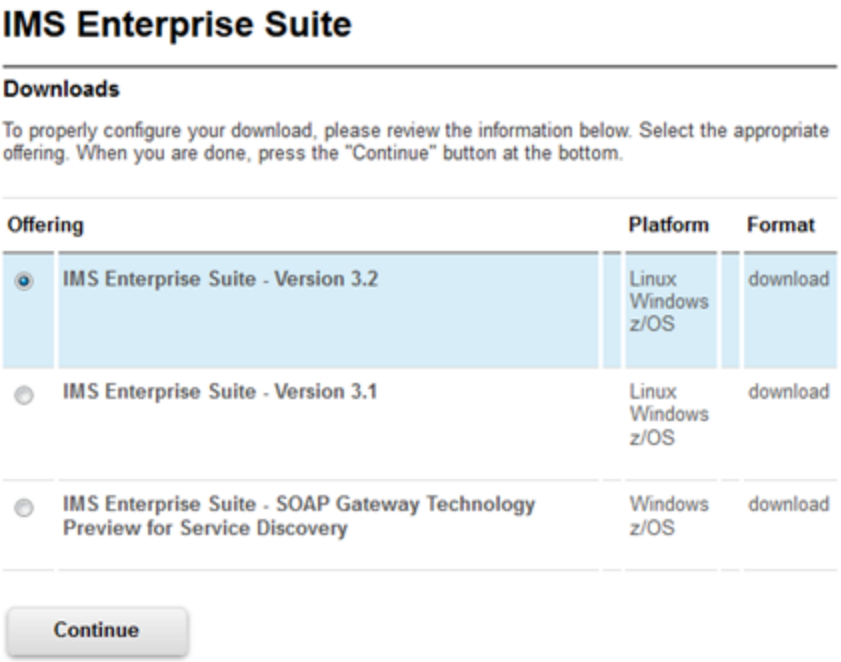
  https://www.ibm.com/marketing/iwm/iwm/web/preLogin.do?source=swg-imsentersuite

  You will see:



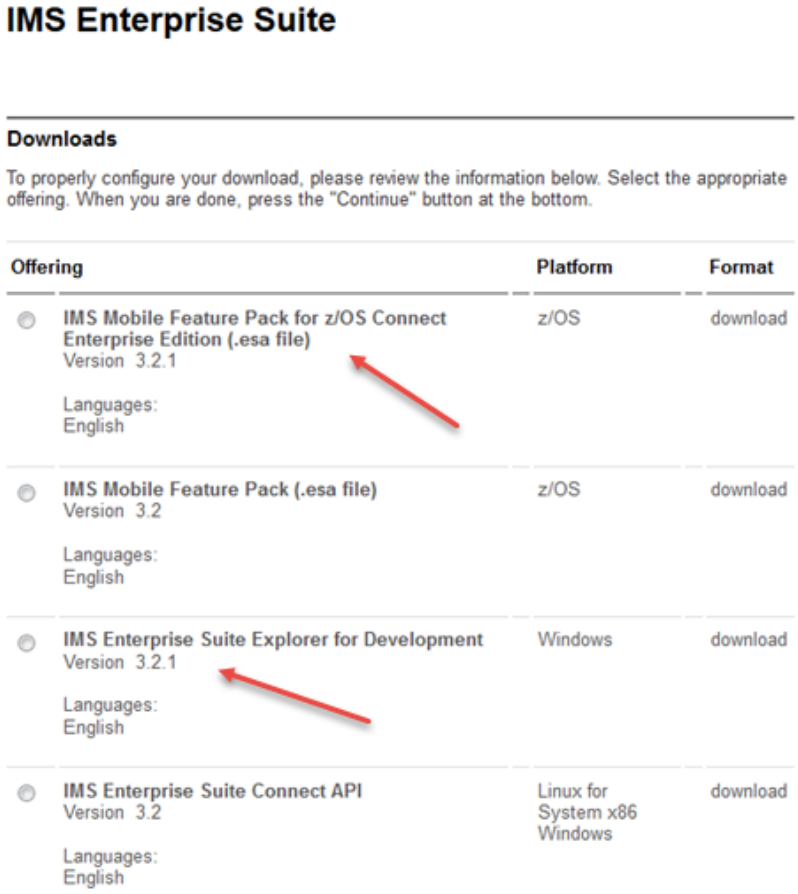
  Either log in with your "IBM ID" and password, or if you do not yet have one, use the "Get an IBM ID" link to acquire an ID.

- When you log in, you will see the following:

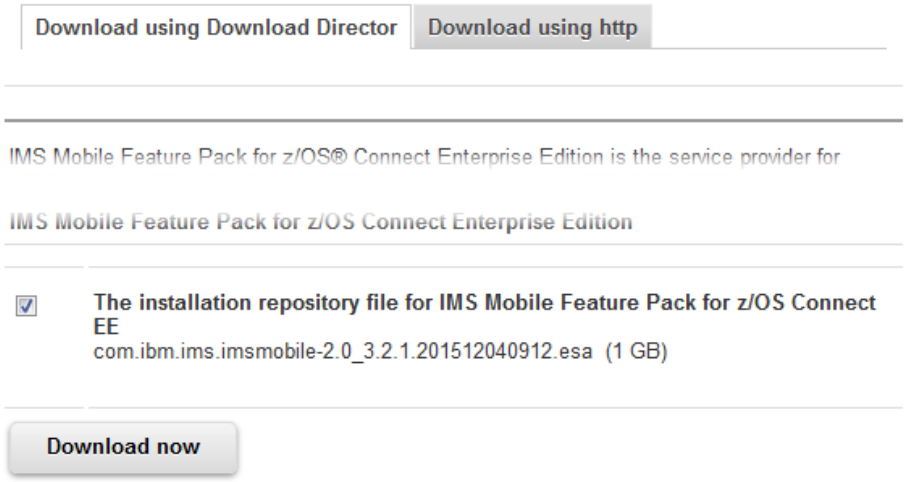


Choose "IMS Enterprise Suite – Version 3.2" and click "Continue"

- The next page will present you with selections for download. You want to focus on the following two:



- ☐ Select the "IBM Mobile for z/OS Connect Enterprise Edition (.esa file)" and click "Continue."
- ☐ Review the license information, check the "I agree" checkbox and then click on "I confirm."
- ☐ Then select either the "Download using Download Director" or "Download using http" tabs, and then click "Download now."



The ESA file that is downloaded is approximately 35MB in size (*not* 1GB as shown).

Note: This will will be uploaded to your z/OS system and installed into the Mobile Feature pack file system you created and mounted a few steps earlier. We will cover the details of that installation a little later.

- ☐ Work your way back to the screen with the selections for download. Check the "IMS Enterprise Suite Explorer for Development Version 3.2.1" option and then click "Continue."
- ☐ Again, select either "Download Director" or "Download using http" tabs, and then click "Download now":

Note: This file is a "repository" that is used by IBM Installation Manager on your workstation to install the IMS Explorer tool. Will will cover the details of that installation a little later as well. This file is a bit larger than 1.0GB.

- ☐ At this point you should have two files on your workstation:

```
com.ibm.ims.imsmobile-2.0_3.2.1.201512040912.esa
IMSES_IMSEXPLORER_V3.2.1.0.zip
```

Create imsmobile.properties file

A little background on this file before you create it: this file provides the installation characteristics for this particular Liberty extension. The extension is the IMS Mobile Feature Pack.

You do not yet have the IMS Mobile Feature Pack installed. But you know *where* you planned for it to be installed – that is, in the `/wlp-ext` directory in the file system you created earlier.

The process used to install the IMS Mobile Feature Pack "esa" file you downloaded earlier will read the `imsmobile.properties` file you are about to create, read the location specified there, and then install the IMS Mobile Feature Pack files into that location.

That is why you have to create the `imsmobile.properties` file first, then install the IMS Mobile Feature Pack – the properties file is used by the installation process to know where to install the files.

Do the following:

- ☐ From your earlier work setting up z/OS Connect EE V2.0, you should already have the following directory in place:

  `/var/zosconnect/v2r0/extensions/`

  Verify that the directory exists. If it does *not*, go back to "SMP/E install of z/OS Connect EE V2.0 and post-install setup" on page 7 and make sure you ran the `zconsetup` shell script.

  If it exists, then you should also see the file `zosconnect.properties` in that directory. That file was created by the `zconsetup` shell script.

- ☐ Create a file named **`imsmobile.properties`** in that directory.

- ☐ Edit the file (using OEDIT) and add these two lines:

  `com.ibm.websphere.productId=com.ibm.ims.mobile`
  `com.ibm.websphere.productInstall=`*`/<path>`*`/wlp-ext`

  Where */<path>* is the mount point for the file system you planned for and created back on page 46. The `/wlp-ext` directory is one you created on page 47. You created it in the file system in which the Feature Pack will be installed.

- ☐ Save the file.

- ☐ Change the ownership of the file so it matches the ownership you see for the `zosconnect.properties` file.

- ☐ Change the file permissions to `664`.

Install IMS Mobile Feature Pack

You are now ready to install the IMS Mobile Feature Pack. Do the following:

- ☐ Upload in binary the following file from your workstation to z/OS and place it in a temporary location where you have room for a file of approximately 35MB:

  `com.ibm.ims.imsmobile-2.0_3.2.1.201512040912.esa`

  Note the path to the file here:

- ☐ This process requires some `/tmp` space to operate. Check your `/tmp` directory, and if it has a lot of files in it, delete what you do not need to make space for this process.

- ☐ Open a Telnet, SSH or a OMVS session

- ☐ Go to the following directory:

  /<zosconnect_install>/wlp/bin

  Where <zosconnect_install> is the location where z/OS Connect EE V2.0 is installed, for example: /usr/lpp/zosconnect/V2R0. You should see a file in that directory named installUtility.

- ☐ Issue the following command *as one line*:

  ./installUtility install

  /<path>/com.ibm.ims.imsmobile-2.0_3.2.1.201512040912.esa --to=imsmobile

  Where:

  - A blank exists between install and the /<path> to the esa file
  - <path> is the location of the esa file

- ☐ Select "1" to agree to the license agreement.
- ☐ You should then see the following signs of success:

```
Step 1 of 2: Installing imsmobile-2.0...
Step 2 of 2: Cleaning up temporary files...
All assets were successfully installed.
Start product validation...
Product validation completed successfully.
```

Note: If after "Step 2 of 2" you see "EDC5133I No space left on device," that is likely due to space problems in /tmp. Clear your /tmp directory and try again.

- ☐ Go to the /<path>/wlp-ext directory[37] and verify the files are present. You should see a /lib directory, and under that quite a few JAR files and a /features directory.
- ☐ Make sure the UNIX permissions allow READ to these directories and files by the ID under which your Liberty z/OS server operates.
- ☐ Mount the file system as READ ONLY (from the R/W required to installed).

Update server.xml in support of IMS Mobile Feature Pack

The following comes from this Knowledge Center article:

http://www.ibm.com/support/knowledgecenter/SS9NWR_3.2.0/com.ibm.ims.mobilezc32.doc/mobilezc_install.htm

Note: What follows assumes you did the install and server setup and skipped over the CICS section. You *can* configure IMS and CICS support concurrently. But to keep things simple and keep the focus on IMS, here we are assuming *just* IMS.

Do the following:

- ☐ Add the following (highlighted in yellow) to the <featureManager> section of the server.xml file:

```
<featureManager>
    <feature>zosconnect:zosconnect-2.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>imsmobile:imsmobile-2.0</feature>
</featureManager>
```

37 Where <path> is what you planned for and created on page 46.

☐ Then, add the following variable to the `server.xml`:

```
<featureManager>
    <feature>zosconnect:zosconnect-2.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>imsmobile:imsmobile-2.0</feature>
</featureManager>

<variable name="imsmobile.install.dir"
    value="<path_to_IMS_feature_pack>" />

  :
```

Where *<path_to_IMS_feature_pack>* is the full path and directory where you installed the IMS Mobile Feature Pack. See page 47 for the path you established, or look in the `/var/zosconnect/v2r0/extensions/imsmobile.properties` file for the location. In our example earlier, the location was:

`/usr/lpp/ims/imses/V3R2/imsmobile/wlp-ext`

Your location may be different.

☐ Next, add the following (highlighted in yellow) to the `server.xml`. See the notes that follow that provide an explanation of each field:

```
<featureManager>
    <feature>zosconnect:zosconnect-2.0</feature>
    <feature>appSecurity-2.0</feature>
    <feature>imsmobile:imsmobile-2.0</feature>
</featureManager>

<imsmobile_imsServiceManager imsRegistryHome="./registry" />

  :
```

Notes:

- This reference for this is provided in the Knowledge Center under this URL: http://www.ibm.com/support/knowledgecenter/SS9NWR_3.2.0/com.ibm.ims.mobilezc32.doc/mobilezc_install.htm
- The element `imsRegistryHome=` tells Liberty z/OS where to create the Derby database that will serve as the "registry" (information storage location). The value `./registry` means a directory will be created under the Liberty server directory[38] with name registry and the Derby database artifacts created there[39].

  Note: This "registry" (Derby database) is written to when you use IMS Explorer to develop your services, as well as used by the IMS Mobile Feature Pack code in z/OS Connect EE V2.0 during runtime.

- The Knowledge Center article speaks of a "technical userid," a "technical group" and a "technical userid password." Those definitions are used only when no other authentication information is available to pass to IMS Connect when RACF=Y is defined there. You will supply the authentication information by logging onto z/OS Connect EE V2.0, so the "technical" definitions in `server.xml` will not be needed.

☐ Save the file.

☐ If the server is stopped, then start the server. Otherwise, allow the dynamic update capability of Liberty to process the changes.

38 This is the directory where the `server.xml` file resides.
39 The registry may be located anywhere you wish. Locating it under the server's directory is a good way to keep everything organized.

- ☐ Look in the `messages.log` file for the server. You should see something like the following messages indicating successful processing of the changes:

```
O [AUDIT   ] GMOIG7777I: The IMS Mobile feature for z/OS Connect
Enterprise Edition initialized successfully.

I [27] DerbyDatastore: GMOIG7777I: The IMS Mobile feature for z/OS
Connect Enterprise Edition initialized successfully.
```

- ☐ You should also see the following changes:
  - ○ A `/registry` directory under your Liberty server directory
  - ○ The following two files in your server directory:
    `ims-connections.xml`
    `ims-services.xml`
  - ○ Two `<include>` elements at the top of your `server.xml` file pointing to the two new files indicated in the previous bullet

The first test will use the default "IMSPingService" function to verify z/OS Connect recognizes the service, and it recognizes the other elements of the IMS implementation.

Note: This test does not exercise a connection to IMS Connect. You will do that after you use IMS Explorer to configure a service and interaction definition.

- ☐ For validation purposes we will show using a browser REST client plugin. The browser you use is up to you. The URLs for Chrome and Firefox are here:

Chrome	https://chrome.google.com/webstore/category/apps Search for "Advanced REST Client"
Firefox	https://addons.mozilla.org/en-us/firefox/addon/restclient/

  Install the REST client of your choosing into your browser.

- ☐ Open a normal browser tab (not the REST client) and send the following URL:

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

  Where *<host>* is the host where your Liberty z/OS server is running, and *<port>* is the *secure* https port.

  You should receive a certificate challenge because the server certificate is signed by a CA that is not known to the browser. Accept the challenge.

  Note: This is done because most REST clients are not very good at handling encryption when the server certificate is self-signed, as is the case with your Liberty z/OS server at the moment.

  You will then receive the basic authentication prompt. Supply the ID (**`Fred`**) and password (**`fredpwd`**). You should receive in return a difficult-to-read[40] string of JSON that represents all the services that are auto-created with the IMS support[41]:

{"zosConnectServices":[{"ServiceName":"IMSPingService","ServiceDescription":"IBM-provided IMS mobile service that pings the gateway server and an IMS Connect subsystem","ServiceProvider":"imsmobile-2.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect/services/IMSPingService"},{"ServiceName":"IMSConnectionAdmin","ServiceDescription":"IBM-provided IMS mobile administrative service for managing IMS connections","ServiceProvider":"imsmobile-2.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect/services/IMSConnectionAdmin"},{"ServiceName":"IMSTranAdmin","ServiceDescription":"IBM-provided IMS mobile administrative service for manaing IMS transactions","ServiceProvider":"imsmobile-2.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect/services/IMSTranAdmin"},{"ServiceName":"IMSTranMessageAdmin","ServiceDescription":"IBM-provided IMS mobile administrative service for managing IMS transaction messages","ServiceProvider":"imsmobile-2.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect/services/IMSTranMessageAdmin"},{"ServiceName":"IMSPropertiesAdmin","ServiceDescription":"IBM-provided IMS mobile administrative service for managing IMS interaction properties","ServiceProvider":"imsmobile-2.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect/services/IMSPropertiesAdmin"},{"ServiceName":"IMSServiceAdmin","ServiceDescription":"IBM-provided IMS mobile administrative service for managing mobile services","ServiceProvider":"imsmobile-2.0","ServiceURL":"https://zt01.pssc.mop.fr.ibm.com:50843/zosConnect/services/IMSServiceAdmin"}]}

40 The browser doesn't understand how to format the JSON, so it simply displays it as it received it.

41 A little later you will use IMS Explorer to configure z/OS Connect EE V2.0 services. Unlike with CICS, where you hand-edited service definitions into the `server.xml` file, with IMS the IMS Explorer tool will do that. It will do that by interacting with these auto-created services in z/OS Connect EE V2.0 that are created when the IMS Mobile Feature Pack function is installed.

☐ Next, open the REST client and send the following request:

HTTP verb:	**PUT**
URL: *(one line)*	`https//<host>:<port>/zosConnect/services /IMSPingService?action=invoke`
JSON body:	none required

If you are prompted for an ID and password, enter `Fred` and `fredpwd`.

We are looking for a "200 OK" message:

```
1.  Status Code        : 200 OK
2.  Content-Language   : en-US
3.  Content-Length     : 119
4.  Content-Type       : application/json
5.  Date               : Mon, 08 Feb 2016 18:14:36 GMT
6.  X-Powered-By       : Servlet/3.1
```

And a JSON response of:

```
{
  "modelVersion": 2,
  "message": "The ping request for the IMS gateway server was successful.",
  "buildNumber": "201512040912"
}
```

☐ Close all instances of the browser. We do this to clear authentication information. We want to log into z/OS Connect EE V2.0 with an ID *different* from "Fred."

As mentioned, that did *not* exercise the connectivity to the backend IMS Connect and IMS region. You will do that after you install IMS Explorer and create some definitions there.

(Optional) Install IBM Installation Manager on your workstation

Earlier (page 49) we had you download the IMS Explorer installation respository, which was a 1GB zip file. That zip file is used by IBM Installation Manager (IM) as input to install the IMS Explorer product.

In this section we will guide you to installing IBM Installation Manager. If you already have it, then skip to the next section where we guide you through installing IMS Explorer.

Do the following:

☐ Go the following URL:

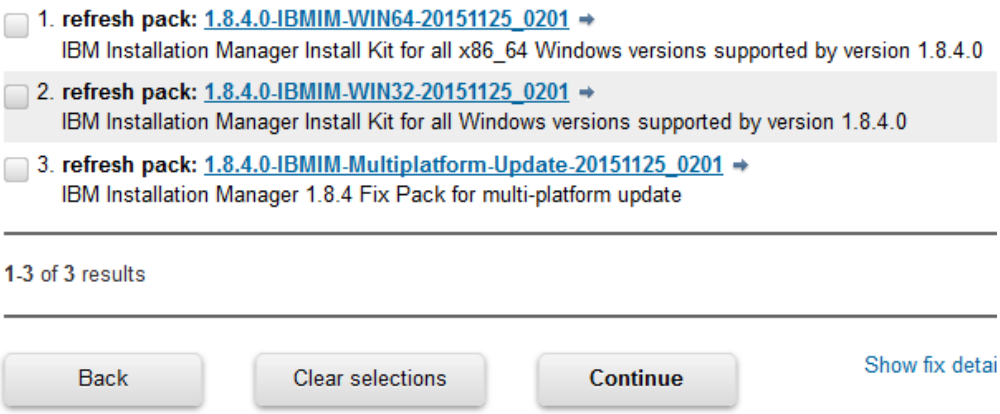
`http://www-01.ibm.com/support/docview.wss?uid=swg24040291`

That is the dowload page for IBM Installation Manager 1.8.4.

☐ Scroll to the bottom. You should see a table with operating systems:

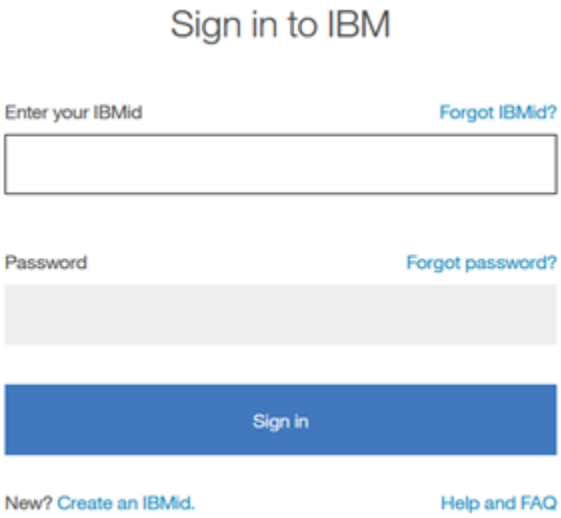
Download	RELEASE DATE	LANGUAGE	SIZE(Bytes)	Download Options What is Fix Central(FC)?
AIX	10 Dec 2015	English	1	FC
HPUX	10 Dec 2015	English	1	FC
Linux	10 Dec 2015	English	1	FC
Mac OSX	10 Dec 2015	English	1	FC
OS400/IBM i	10 Dec 2015	English	1	FC
Solaris	10 Dec 2015	English	1	FC
Windows	10 Dec 2015	English	1	FC
z/OS	10 Dec 2015	English	1	FC

- ☐ Identify the operating system for your workstation and then click on the "FC" link over to the right.
- ☐ For example, for Windows you would see:



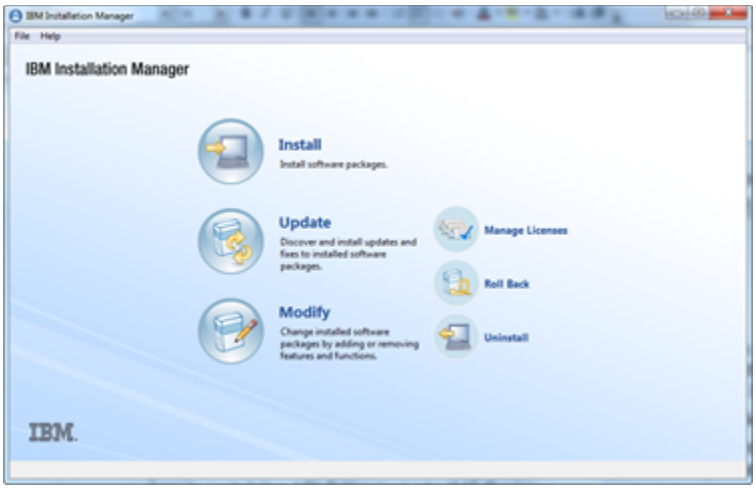
  And there you would select your level of Windows and click "Continue."

- ☐ Sign in with your "IBMid" (or if you do not have one, create an ID):



- ☐ Then select whether to download using "Download Director" or "HTTPS". Then click "Continue."
- ☐ Click on the download link that appears and save the ZIP file.
- ☐ When the download is complete, exact the ZIP into a temporary directory.
- ☐ Double-click on the `install.exe` file.
- ☐ Click "Next," then accept the license agreement, "Next" and then "Install."
- ☐ When it completes, click the "Restart Installation Manager" button[42].
- ☐ You should now see:

42 IM uses a stub of itself to install itself. ☺



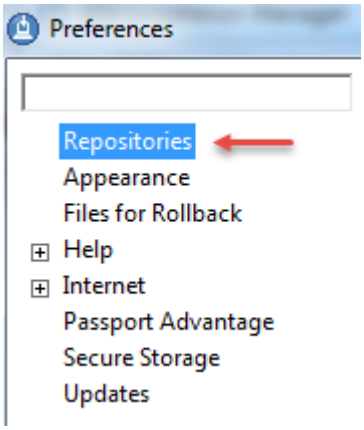
And that is IBM Installation Manager.

Install IMS Enterpise Suite Explorer for Development 3.2.1 (if you do not already have it)

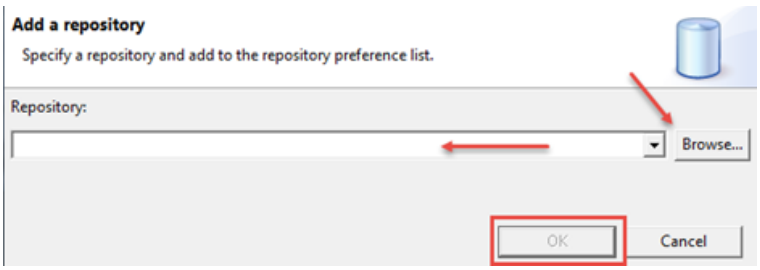
As mentioned earlier, IMS Enterprise Suite Explorer is a key component in the configuration and management of a z/OS Connect EE V2.0 server instance that uses IMS as a backend system of record. The level supported with z/OS Connect EE V2.0 is IMS Explorer 3.2.1.

Earlier (page 49) you downloaded IMS Explorer in the form of a ZIP file. That was actually an Installation Manager "repository" (or, "source input file"). To install you point IM at the ZIP file. Do the following:

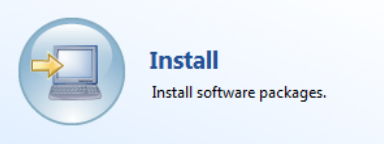
- ☐ In Installation Manager, click on *File* → *Preferences*, then select "Repositories" from the left side option list:



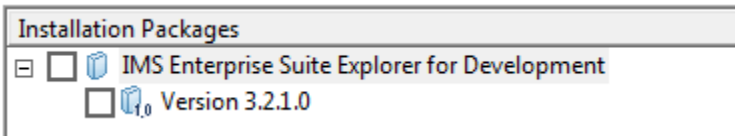
- ☐ On the right side, click on the "Add Repository..." button.
- ☐ Next, use the "Browse" button to navigate to where you stored the IMS Explorer ZIP file you downloaded (page 49). When you have the file selected, click on the "OK" button:



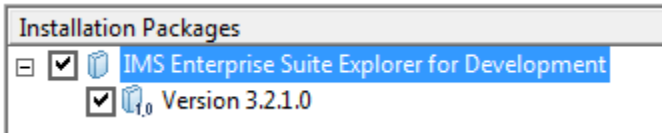
- ☐ You should see the path and file in your list of repositories. Make sure the checkbox to the left of the repository is checked (it should be by default after including the repository path and file in the preferences window).
- ☐ Click the "OK" button to close the "Preferences" window.
- ☐ From the main panel of IM, click the "Install" button"



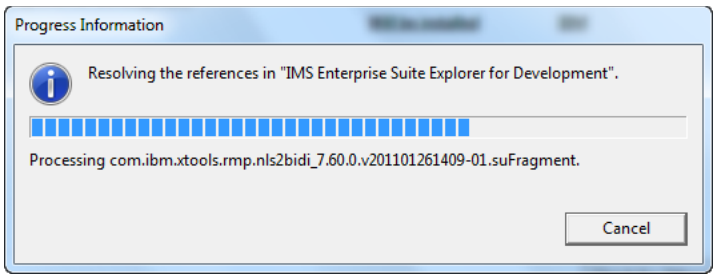
- ☐ If the IMS Explorer repository is the only one in the list, you should now see only the IMS Explorer as available for install:



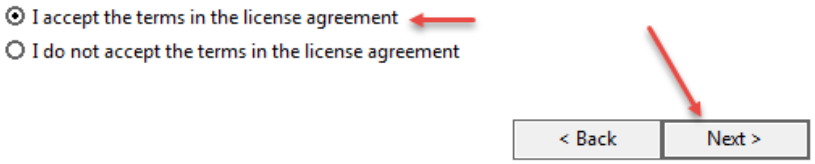
- ☐ Check the first box (and the second will be auto-checked as well):



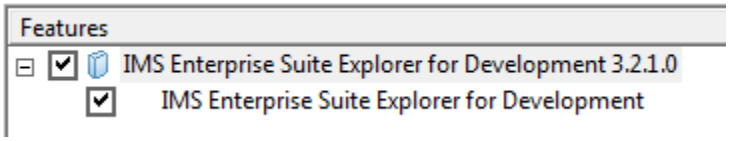
- ☐ Click "Next". You will see it go through a progression of files it is resolving:



- ☐ Then accept the license and click "Next":



- ☐ It will then go through the process of validating files and package, which may take a minute or more. Then it will present you with an "Install Packages" window. Click "Next."
- ☐ It may prompt you with another "Install Packages" window. Again, click "Next."
- ☐ Select your language (English is default), then click "Next."
- ☐ It will come back with a summary of what will be installed:



And on that page click "Next."

- ☐ Finally, an "Install Packages" window will appear with an "Install" button. Click that button. The install process may take several minutes.
- ☐ When it completes, click the "Finish" button.
- ☐ Close Installation Manager.
- ☐ Start IMS Explorer. It will prompt you to "Select a workspace." The default will suffice to start with. Then click "OK."

(Optional) Install the z/OS Connect EE V2.0 API Editor Tool into IMS Explorer

To use z/OS Connect EE V2.0 you will need the API Editor tool installed into *some* Eclipse platform. It does not need to be IMS Explorer, but it *can* be[43].

If you choose to install the z/OS Connect EE V2.0 API Editor into your new copy of IMS Explorer, then go to "Installing the z/OS Connect EE V2.0 API Editor" on page 18. The steps there apply to IMS Explorer just as they apply to z/OS Explorer.

z/OS Connect EE V2.0 and the Phone Book Sample

Phone Book sample in IMS

The "Phone Book" sample is an initial verification program (IVP) supplied with IMS. It is often referred to as "IVTNO." It is documented here:

http://www.ibm.com/support/knowledgecenter/SSEPH2_14.1.0/com.ibm.ims14.doc.ins/ims_jdbcconsamples.htm?lang=en

The Phone Book consists of a transaction program as well as a backing database. The transaction provides the ability to add a contact, delete a contact, update a contact and browse contacts.

We will not provide step-by-step instructions for setting up the Phone Book program. It is a well-known IVT program for IMS.

However:

- ☐ Work with your IMS system programmer to insure the Phone Book sample IVT program and database is in place in the IMS region, and an instance of IMS Connect is in front of that IMS region.
- ☐ Go to the following URL:

  http://www.ibm.com/support/knowledgecenter/SS9NWR_3.2.0/com.ibm.ims.mobilezc32.doc/ims_define_msgmetadata.htm

  That provides the data structure used by the phone book sample. Copy the data structure provided there and place it in a file on your workstation called `IVTNO.txt`.
- ☐ Note the name and location on your workstation for the file:

- ☐ Work with your IMS administrator and note the values for the following:

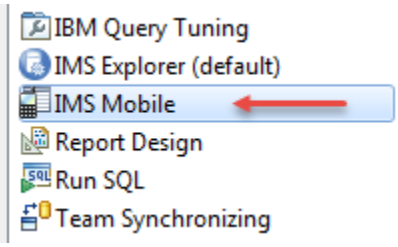
IMS Connect Host	
IMS Connect Port	
IMS Region Name	
RACF=Y in IMS Connect?	☐ Yes ☐ No *If "No," then a userid and password are not needed*
ID and PW to use	

43 You may also have the API Editor installed in several different Eclipse platforms – z/OS Explorer, CICS Explorer and IMS Explorer.

IMS Explorer definitions, Part 1 (connections, interactions, transactions, data import)

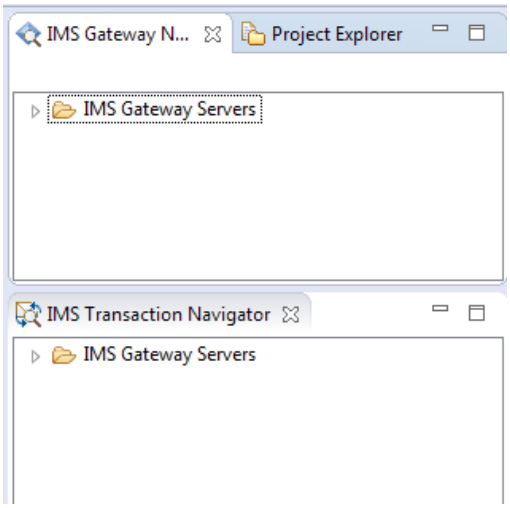
In this section you will create definitions in IMS Explorer that will define your z/OS Connect EE V2.0 server, the IMS Connect server to use, transaction information for the Phone Book sample, and you will bring the COBOL language structure into the tool[44].

- ☐ Make sure your z/OS Connect EE V2.0 server is up and running.
- ☐ Open IMS Explorer if it is not already open.
- ☐ Select *Window* → *Open Perspective* → *Other*, then select "IMS Mobile":

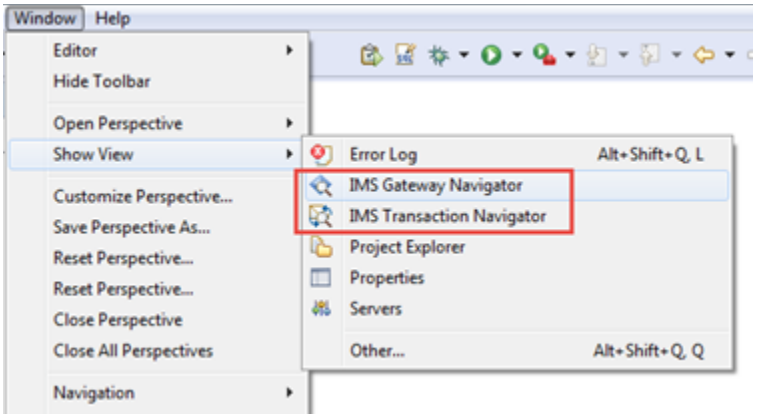


Then click "OK".

- ☐ On the left side you should see both the "IMS Gateway Navigator" as well as the "IMS Transaction Navigator":

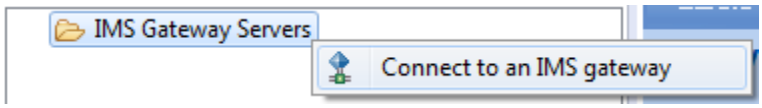


If either or both of those are not present, then select *Window* → *Show View* and select the missing navigator views:

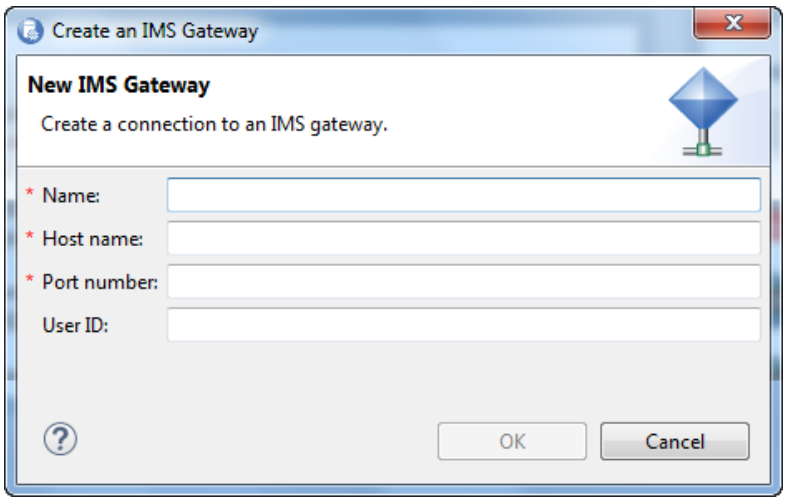


44 The YouTube video at https://www.youtube.com/watch?v=4UPaN-6D348 provides a nice demonstration of this from time mark 12:18 through 31:30.

- ☐ In the IMS Gateway Navigator view, right click on "IMS Gateway Servers" and then click on "Connect to an IMS gateway":



- ☐ Fill in the properties for your IMS Gateway, which in our case is the z/OS Connect EE V2.0 server[45]:

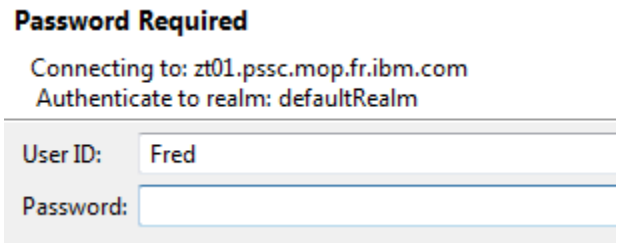


Where:

Name:	Is simply a name to reference this server. For example, ZOSCONN[46].
Host:	The TCP host[47] on which the z/OS Connect EE V2.0 can be reached.
Port number:	The https (secure) port for your z/OS Connect EE V2.0 server.
User ID:	The ID used to authenticate into z/OS Connect EE V2.0[48].

Then click "OK".

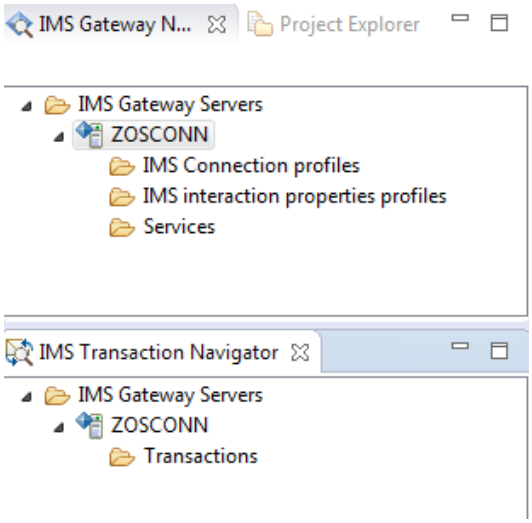
- ☐ It will attempt to connect to the defined z/OS Connect EE V2.0 server. It will see that basic authentication is enabled, so it will present a window for the password:



Supply the password (`fredpwd`) and click "OK".

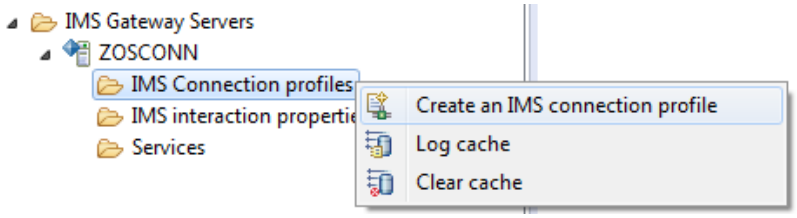
45 From an IMS perspective, z/OS Connect EE V2.0 is seen as a "gateway" to IMS. Hence the phrase "IMS Gateway" server.
46 Use a name meaningful to you. Avoid spaces, apostrophes, slashes or other non-standard characters in this name.
47 Or dotted-decimal IP address.
48 If you are using the "basic" registry in `server.xml`, then the ID is likely `Fred` since that's what we illustrated earlier.

☐ You should now see something like the following in your two "navigator" windows:



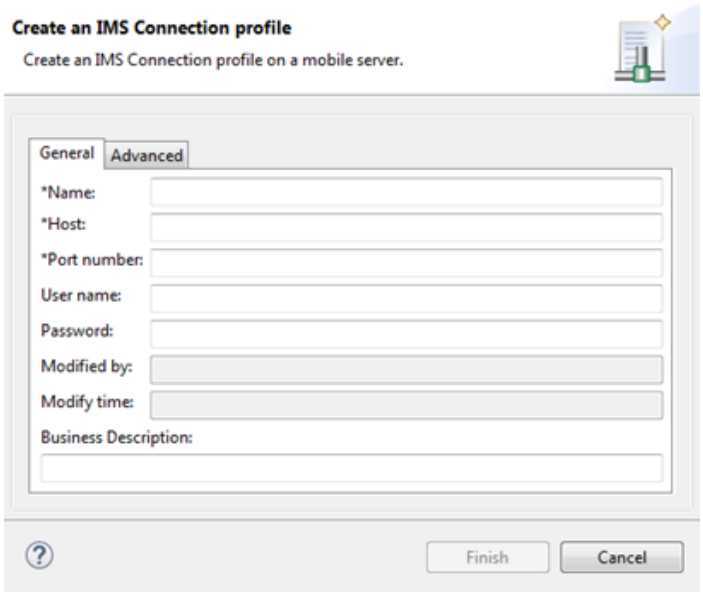
Note: if the folders are not visible, expand until they show as illustrated here.

☐ Next, in the "IMS Gateway Navigator" view (upper left), right click on the "IMS Connection profiles" folder, then click on "Create an IMS connection profile":



This connection profile defines the IMS Connect that will be used by z/OS Connect EE V2.0 when it calls IMS.

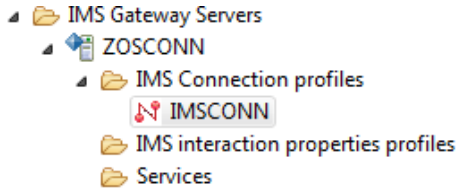
☐ Fill in the values:



Where:

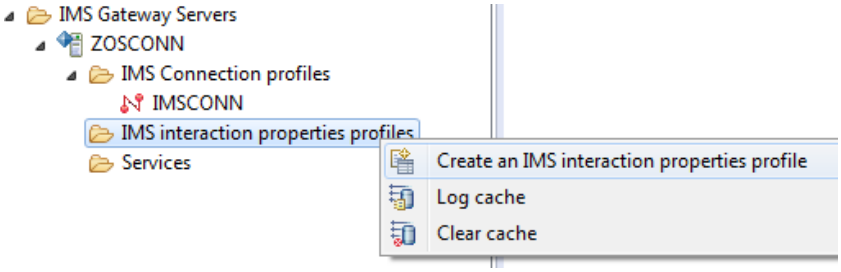
Name:	A value used to reference this definition. Example: IMSCONN[49].
Host:	The TCP host[50] on which the IMS Connect server can be reached.
Port number:	The port on which the IMS Connect server listens.
User name:	The ID z/OS Connect EE V2.0 will pass to IMS Connect. You captured this value back on page 58.
Password:	The password that is associated the with the user name. You captured this value back on page 58.

☐ Click "OK". You should see your new connection:

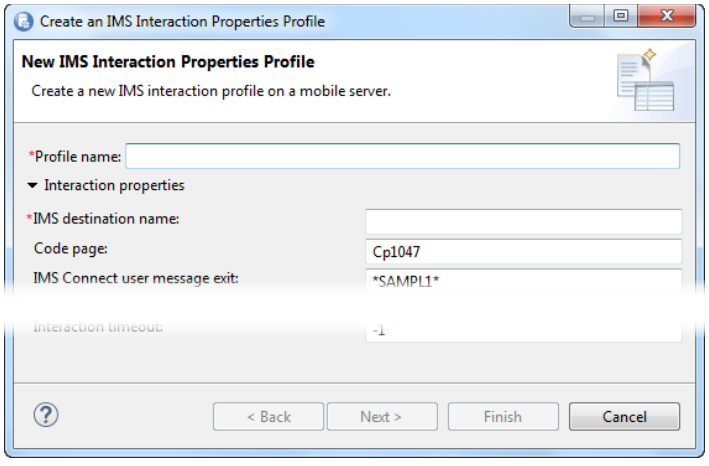


Note: If you do not see the new connection, right click on "ZOSCONN" (the server definition) and then select "Refresh." IMS Explorer will communicate with the z/OS Connect EE V2.0 server and retrieve definition elements and refresh this screen.

☐ In the "IMS Gateway Navigator" view (upper left), right click on "IMS interaction properties profiles" and then click on "Create an IMS interaction properties profile":



☐ Fill in two values: the "Profile name" and the "IMS destination name":

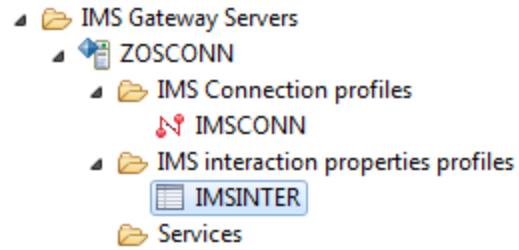


Where:

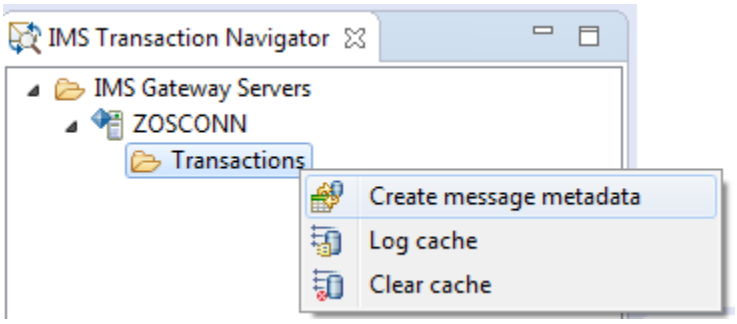
Profile name:	A name used to identify and reference this interaction profile.
IMS destination name:	The IMS region name[51].

49 Use a name meaningful to you. Avoid spaces, apostrophes, slashes or other non-standard characters in this name.
50 Or dotted-decimal IP address.

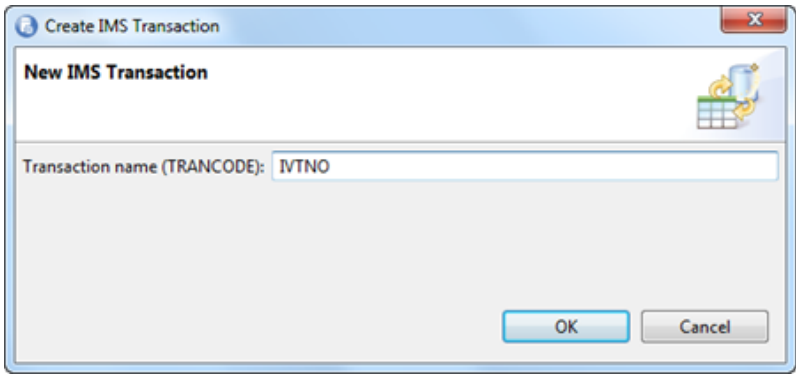
☐ Click "Finish". You should see your new interaction profile:



☐ In the "IMS Transaction Navigator" view (lower left), right click on "Transactions," then click on "Create message metadata":



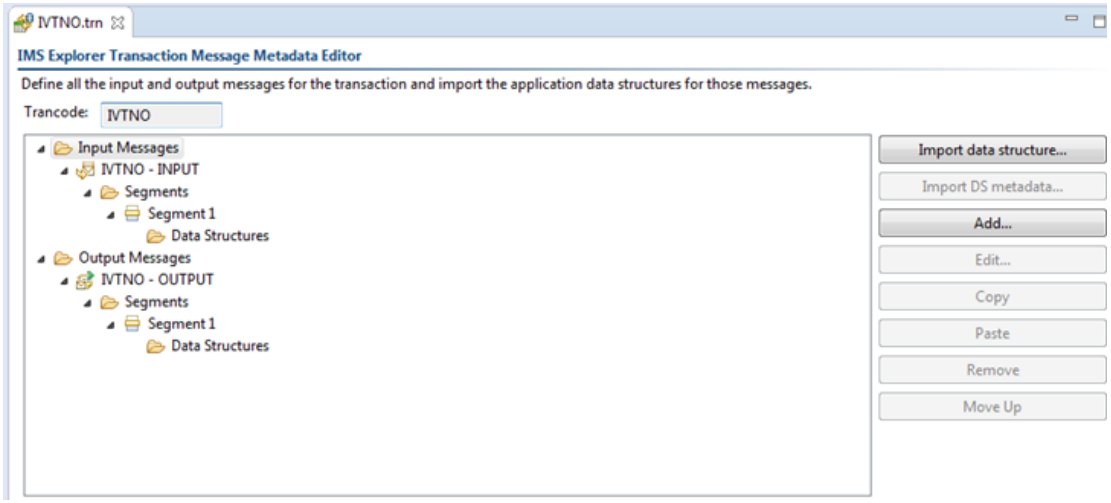
☐ Provide a transaction name of `IVTNO` and click "OK":



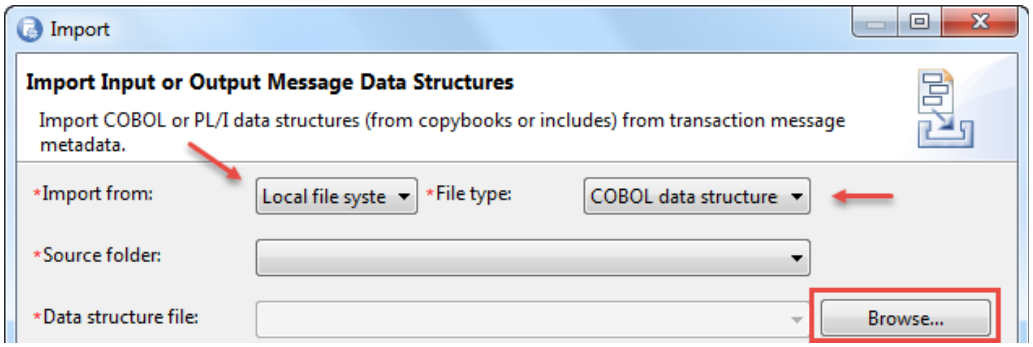
Note: If you have IMS Explorer 3.2.1.2 or higher, this name is arbitrary and you can provide a "Transaction Code Override" value at a later point. Prior to IMS Explorer 3.2.1.2 this field must match the actual transaction code.

☐ You should now see the "Metadata Editor":

51 When z/OS Connect EE V2.0 communicates with IMS Connect, it will pass this value. IMS Connect will then use that value to understand which IMS region to interact with on the backend.



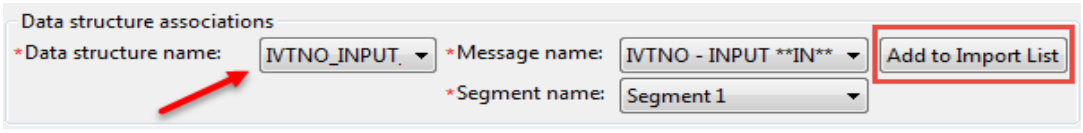
☐ Click on the "Import data structure" button, then do the following:



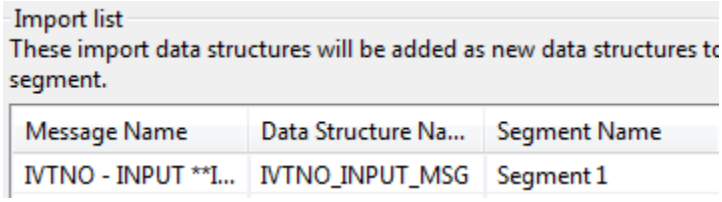
Where:

Import from:	"Local file system" implies from your workstation. Earlier (page 58) we had you download that file. You could FTP it from the z/OS system if you had that connection configured. Manual download and import from file system is easy and that's what we're doing here.
File type:	"COBOL data structure" (as opposed to "Full program", or PL/I)
Data structure file:	Use the "browse" button and navigate to the location (see page 58) and select the file for import.

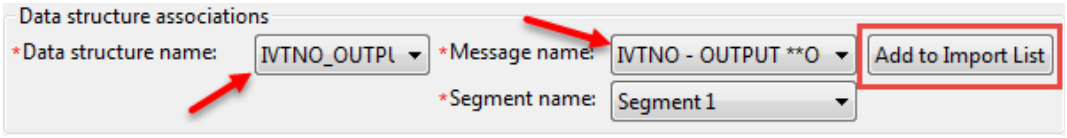
☐ On that same panel, use the "Data structure name" drop-down and select "IVTNO_INPUT" and then click "Add to Import List":



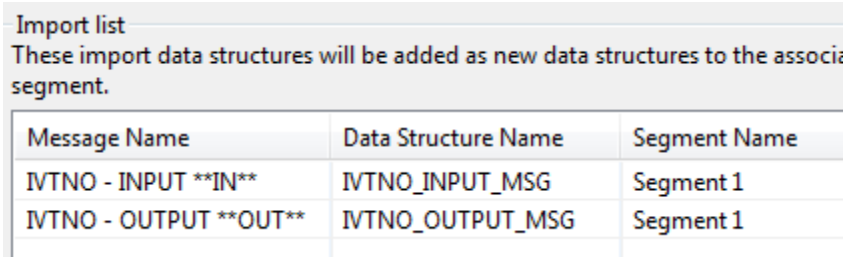
You should see it appear in the import list:



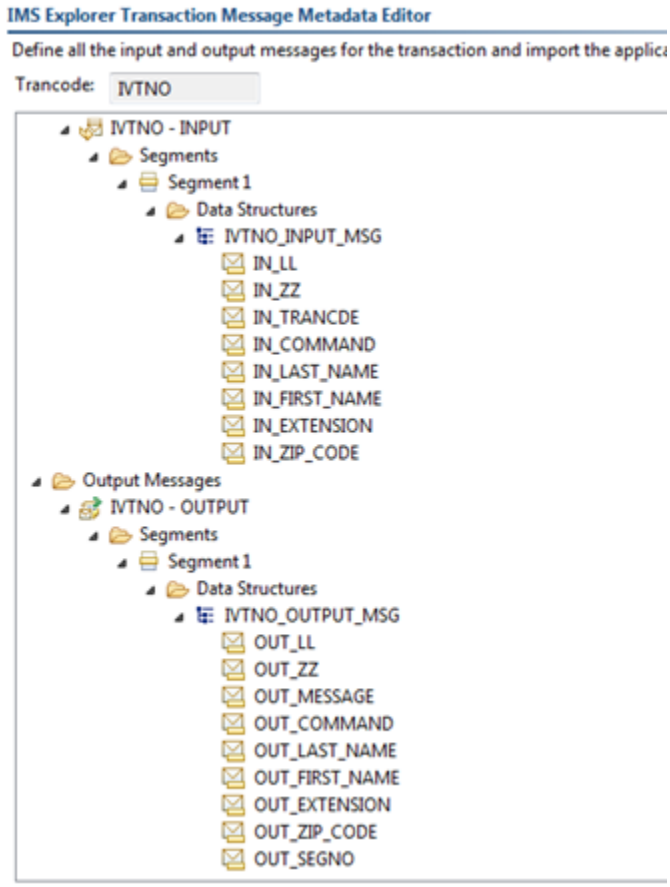
☐ Go back to the "Data structure name" drop-down and select and do the same for the "OUTPUT" structure name:



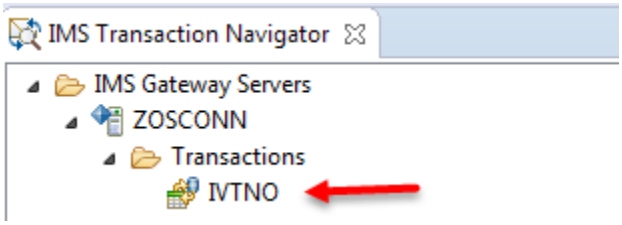
Where both "Data structure name" and "Message name" are for "output". Then click the "Add to Import List." Both should appear in the list:



- ☐ Click the "Finish" button.
- ☐ If you expand the folders in the "Metadata editor" you should see the input fields and output fields under their respective folders:



- ☐ Select *File* → *Save*. This will save the transaction meta-data and synchronize the changes up to your z/OS Connect EE V2.0 server. You should now see your new transaction in the "IMS Transaction Navigator" (lower left):



That's the basic plumbing: you defined your z/OS Connect EE V2.0 server, you defined the IMS Connect server, and you configured the transaction meta-data for the Phone Book application by importing the data structure.

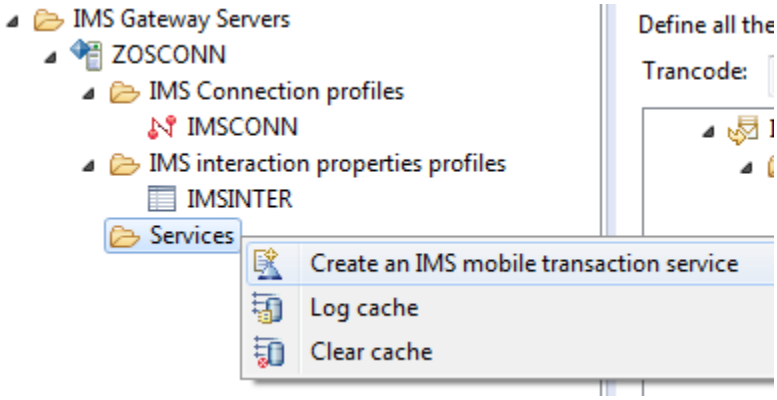
FYI: The changes you made were synchronized to the running z/OS Connect EE V2.0 server using the built-in REST services for IMS Mobile Feature Pack. Some of the changes were stored in the Derby "registry" database, and some were stored in the ims-connections.xml file.

Now it's time to create a z/OS Connect EE V2.0 service to "Add" a contact to the phone book[52].

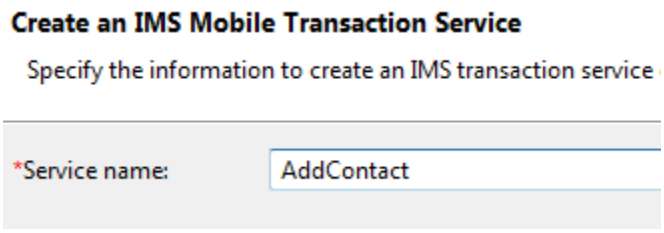
IMS Explorer definitions, Part 2 (service definition)

In the above section, you defined the basic elements necessary to send a request to an IMS backend. The Services definitions bring it all together by making use of these artifacts. Do the following:

- ☐ In the "IMS Gateway Navigator" view (upper left), right click on the "Services" folder, then click on "Create an IMS mobile transaction service":

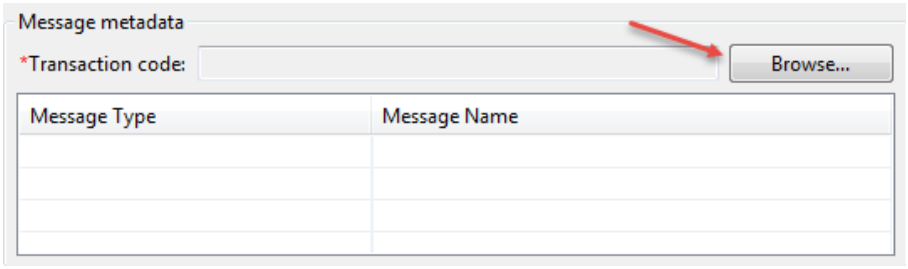


- ☐ In the "Create an IMS Mobile Transaction Service" window that appears, provide a name for the service, such as "AddContact":

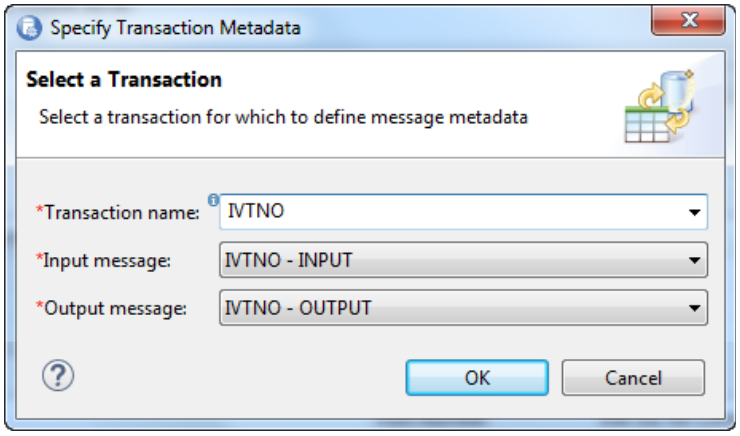


- ☐ For the "Transaction Code," click on the "Browse" button:

52 The transaction provides four operations: add, delete, update and browse. We'll show "Add" here. The other three are similarly defined, with the difference being the command code supplied on input, and the fields you choose to show or hide for each.

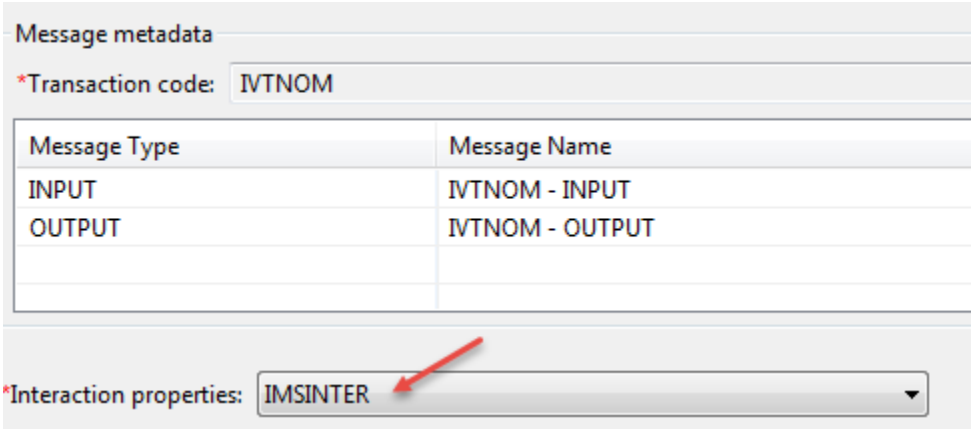


Then select the `IVTNO` transaction name and the input and output messages:

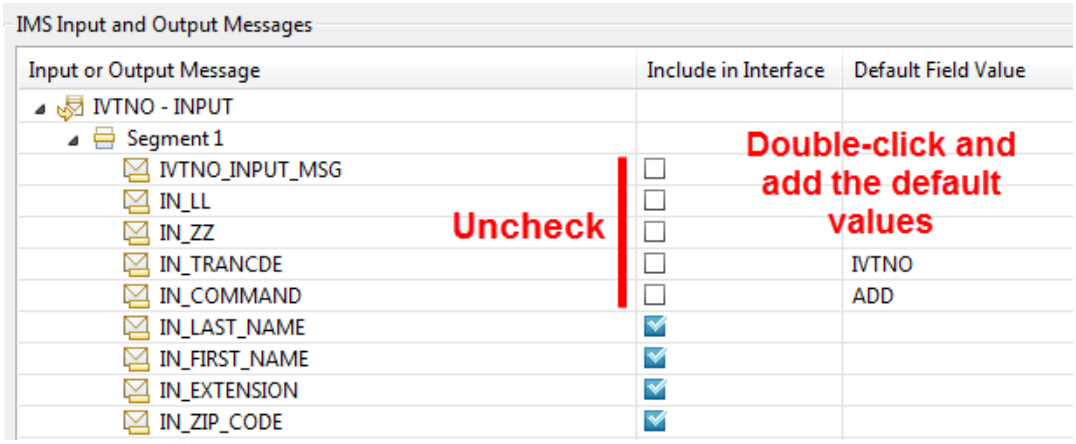


Click "OK".

- ☐ Next, select the "Intraction properties" to use by selecting from the down-down menu the one you created earlier:

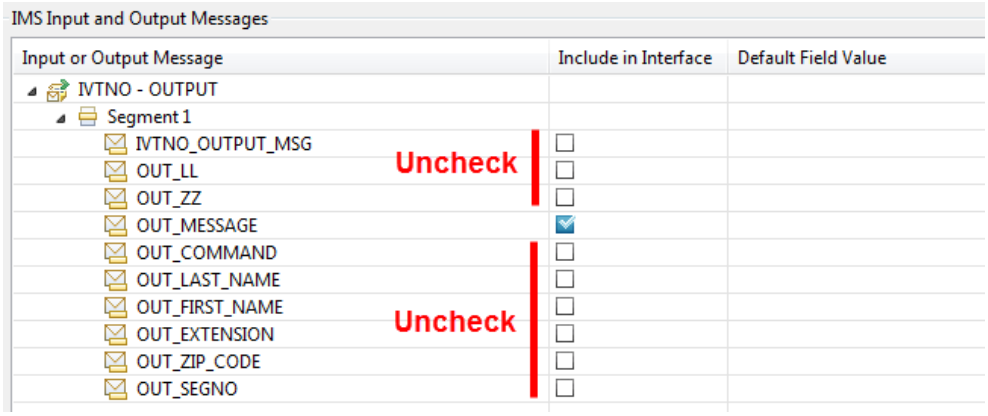


- ☐ Finally, take a look at the "Connection profiles" … you should have only one there (the one you created). If more than one appears, keep the one you want to use and remove the others.
- ☐ Click the "Next" button to expose the input and output field panel.
- ☐ For the "Input" fields, do the following:



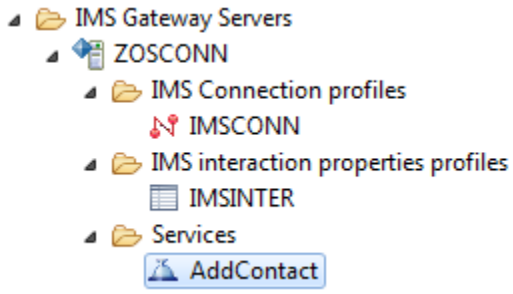
- Uncheck the the first five fields. This hides those fields from the REST client's view.
- Double-click on the `IN_TRANCDE` default field value and type **`IVTNO`**. This will supply the string "IVTNO" without the caller needing to supply it.
- Double-click on the `IN_COMMAND` default field value and type **`ADD`**. This will supply the string "ADD" without the caller needing to supply it.

☐ For the "Output" fields, uncheck all fields *except* `OUT_MESSAGE`:



☐ Click "Finish."

☐ You should see your new service in the "IMS Gateway Navigator" field:



Note: that definition has been synchronized up to your z/OS Connect EE V2.0 server. It put some XML into `ims-services.xml`, as well as some data into the Derby registry.

Test the service

There are two ways this can be done: using the IMS Explorer "run a test case" function, and using the simple REST client like we did before. We'll show both.

We'll start with the REST client. Do the following:

☐ Open a normal browser tab (not the REST client) and send the following URL:

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

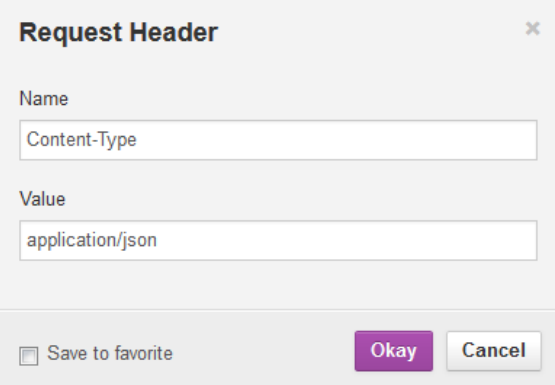
Where `<host>` is the host where your Liberty z/OS server is running, and `<port>` is the *secure* https port.

You should receive a certificate challenge because the server certificate is signed by a CA that is not known to the browser. Accept the challenge.

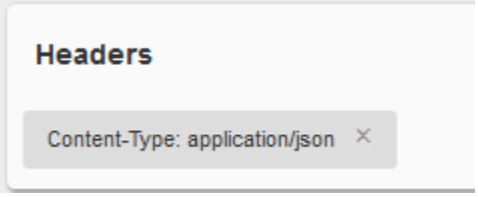
Note: This is done because most REST clients are not very good at handling encryption when the server certificate is self-signed, as is the case with your Liberty z/OS server at the moment.

You will then receive the basic authentication prompt. Supply the ID (**`Fred`**) and password (**`fredpwd`**). You should receive in return a difficult-to-read string of JSON that represents all the services that are understood by z/OS Connect EE V2.0.

☐ Next, open the REST client and set a "Custom Header" with a name of Content-Type and a value of application/json. For the Firefox REST client:



Click "Okay". You should see:



☐ Next, send in the following request:

HTTP verb:	**`PUT`**
URL: *(one line)*	`https//<host>:<port>/zosConnect/services /AddContact?action=invoke`
JSON body:	`{ "IVTNO_INPUT_MSG" : { "IN_EXTENSION" : "123", "IN_LAST_NAME" : "Client", "IN_FIRST_NAME" : "Fred", "IN_ZIP_CODE" : "11111" } }` **Note:** if copying from PDF, paste the JSON into a basic text editor and make sure the double-quotes are generic double-quotes and not some other character.

If you are prompted for an ID and password, enter `Fred` and `fredpwd`.

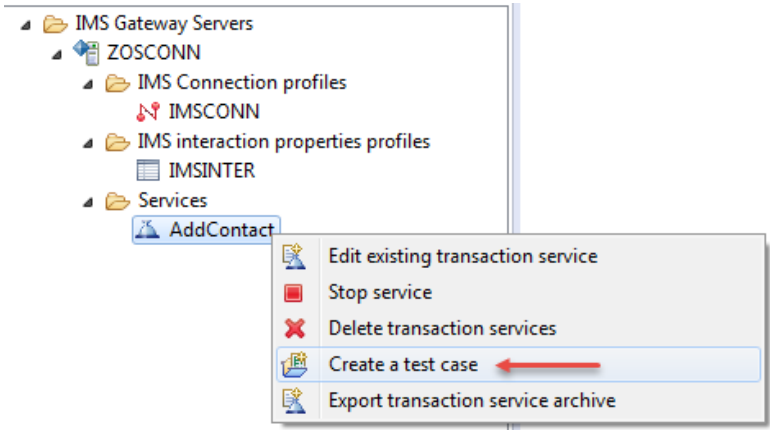
We are looking for a "200 OK" message and a formated JSON response of:

Note: if you receive `"OUT_MESSAGE": "ADDITION OF ENTRY HAS FAILED"` it is most likely because the last name already exists.

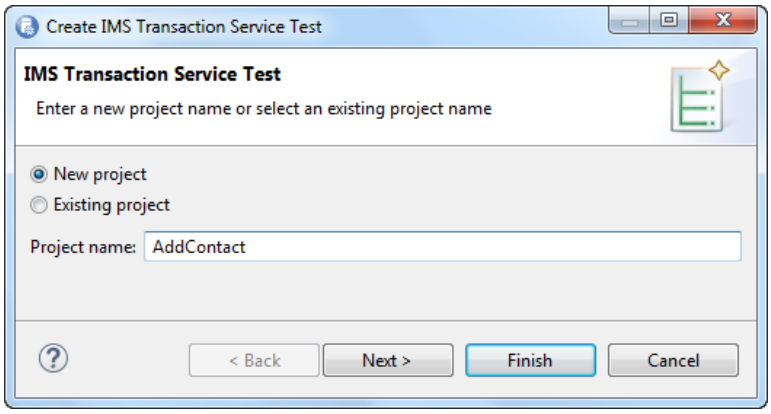
You have validated your service.

If you want to use the test case tool in IMS Explorer, then do the following:

- ☐ In the "IMS Gateway Navigator" (upper left), right-click on the new service, and then click on "Create a test case":

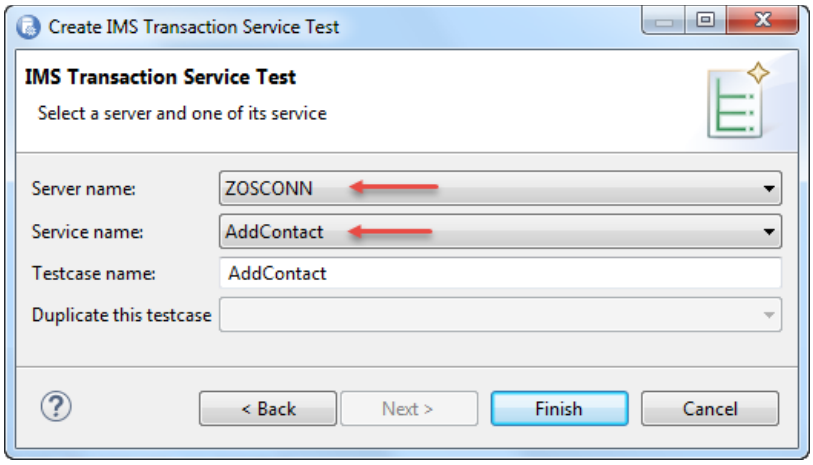


- ☐ Next, select the "New Project" radio button and add a name for this testcase, such as "AddContact".



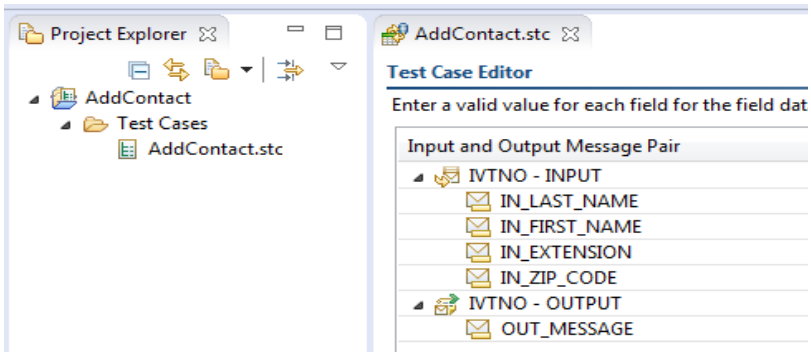
Then click "Next".

- ☐ Make sure the server name and service name map to the definitions you created earlier:



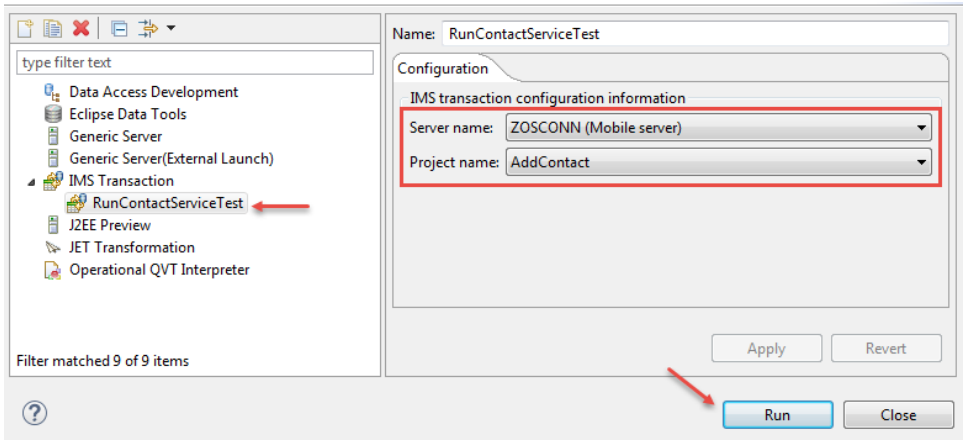
Then click "Finish"

- ☐ You should now see something like this:

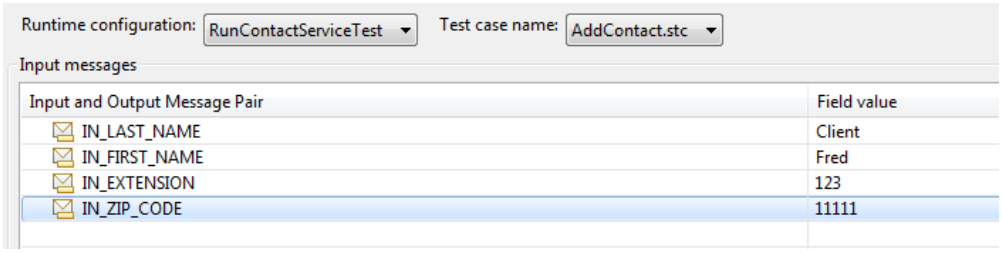


Note: a "Project Explorer" view has opened and come to the foreground. Your "IMS Gateway Navigator" view is still there, it's just behind the Project Explorer tab.

- ☐ From the menu bar, select *Run → Run Configurations* ...
- ☐ Then, make sure the "RunContactServiceTest" option is highlighted, the server is your z/OS Connect EE V2.0 server definition, and the "Project name" is your "AddContact" test project. Then click "Run":



- ☐ A rather large, full-screen window will come up. Double-click on each input field and provide a value, for example:

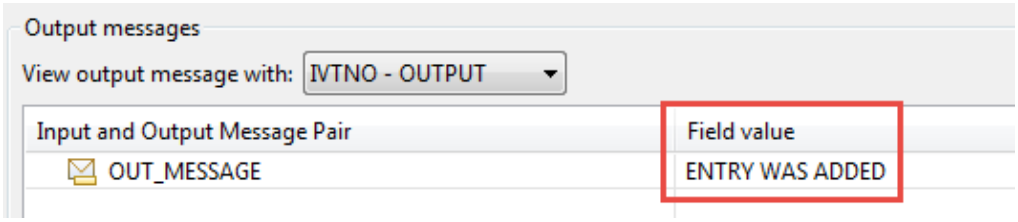


Note: remember, if the last name field is already in the database, you will get the message saying the add of the contact failed. Make sure it's a unique value.

- ☐ In the very middle of the screen you'll see a little green button:

Click that button to run the test case.

- ☐ Look in the output field. If successful, you will see:



You have validated your service.

- ☐ Close the test case window.

FYI – A discussion about z/OS Connect EE V2.0 APIs and IMS services

One of the significant functional enhancements to z/OS Connect EE V2.0 over z/OS Connect V1.0 was the addition of the API layer. The API layer provides a much better REST implementation: it provides the ability to use HTTP verbs to imply the action against a resource; it provides access to path and query parameters; and it provides the *ability to map, hide, or assign values to specific fields*.

We just saw that IMS Explorer also has the ability to hide and assign values to specific fields. Does that mean the z/OS Connect EE V2.0 API Editor brings little added value to interaction with IMS? On the contrary, z/OS Connect EE V2.0 and the API layer function *enhances* the flexibility of your design.

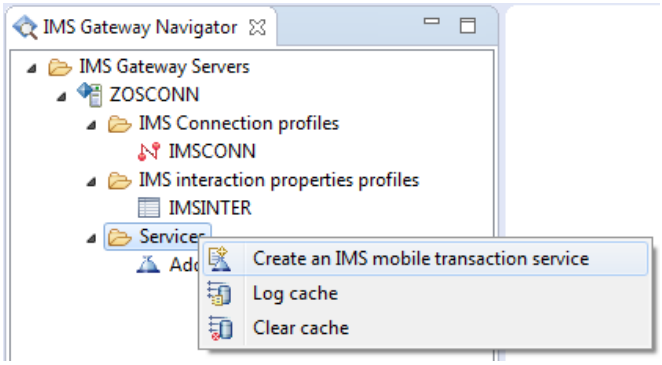
In the sections that follow we will guide you through a set of step-by-step instructions for creating a new service and an API that runs on top of that. If you're interested in the design considerations we used for this exercise, see "A discussion of why the IMS Phone Book API was designed as it was" on page 100. In that section we explain how z/OS Connect EE v2.0 *enhances* the flexibility of exposing the Phone Book sample.

But if you'd rather get to creating the new service and API, then continue on.

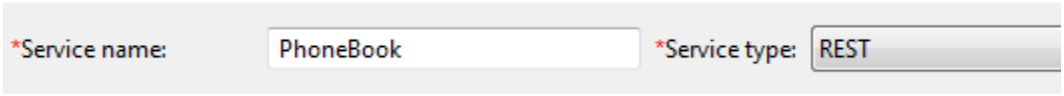
Create a more generic service using IMS Explorer

Do the following:

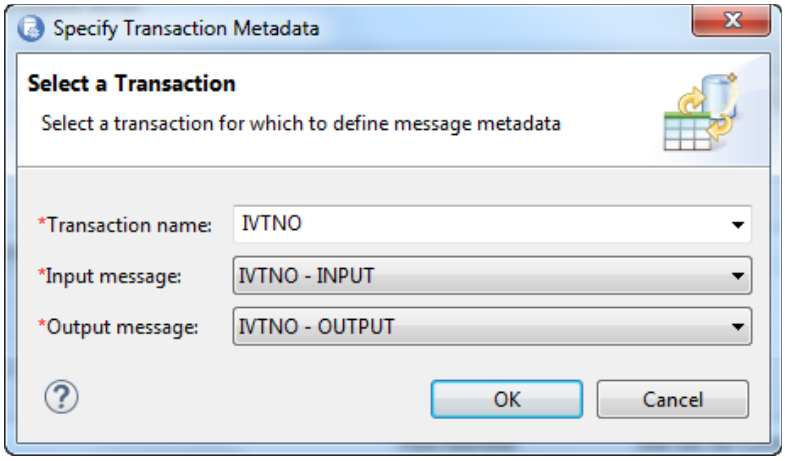
- ☐ In the IMS Explorer tool, go to the "IMS Gateway Navigator" view (upper left), right click on the "Services" folder, then select "Create an IMS mobile transaction service":



☐ Provide a "Service name" of "PhoneBook":



☐ Click on the "Browse" button for "Transaction Code," then select your transaction (`IVTNO`) and make sure the input and output messages are properly reflected:



Click "OK".

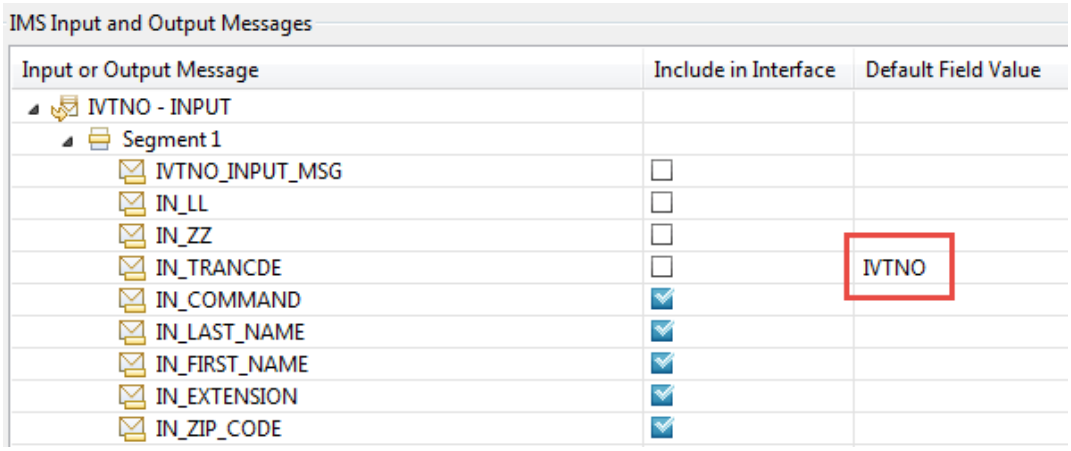
☐ Select your interaction properties definition from the drop-down menu:

☐ Look at the connection profiles. If there's only one and it's the one you created earlier, then continue. Otherwise, remove the ones you do *not* want, leaving the one you *do* want to use for this service.

☐ Click "Next."

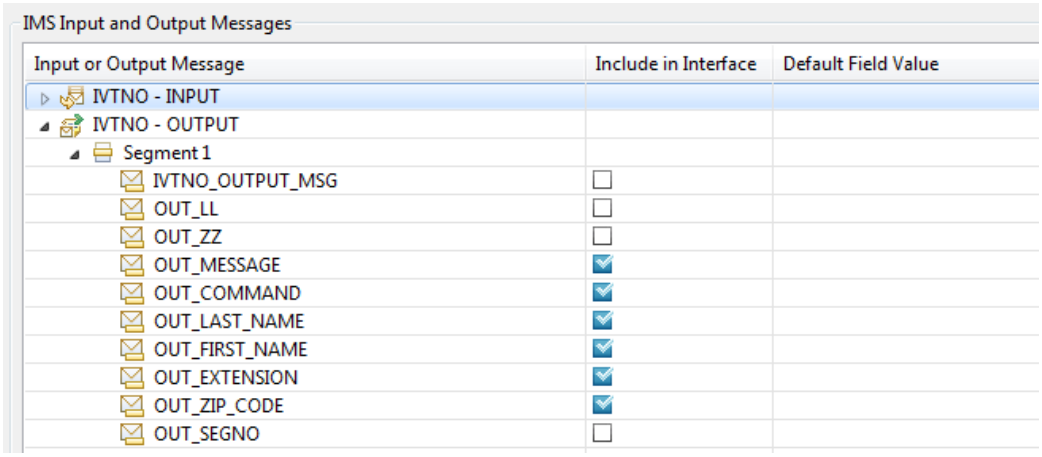
☐ Set the *input* mapping as shown here[53]:

53 This was discussed under ""Generic" Phone Book service" on page 101.

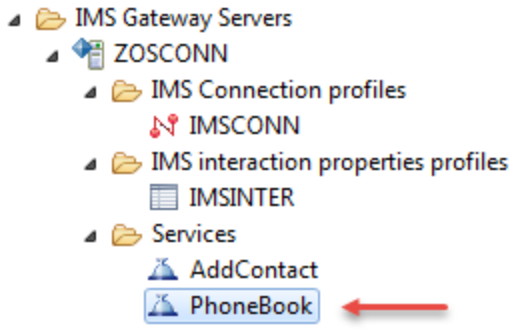


Note: to set the `IN_TRANCDE` value, double-click on the field and supply the value for the transaction you created earlier.

☐ Set the output mapping as shown here:

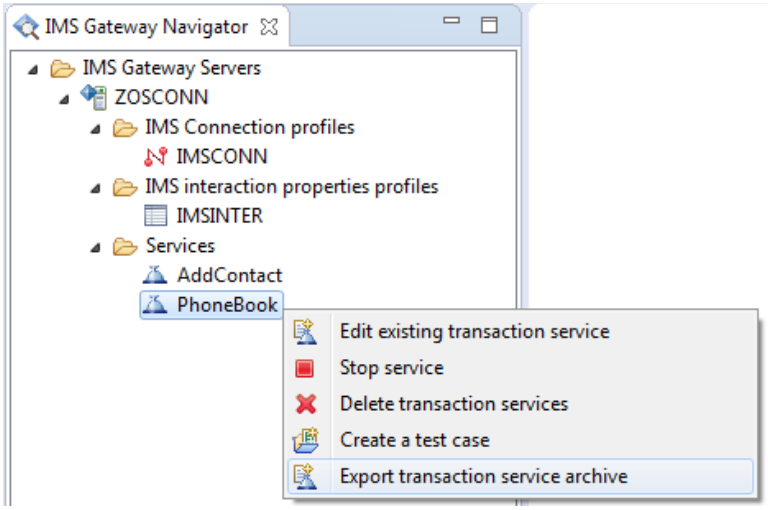


☐ Click "Finish." You should see your service in the navigator view:

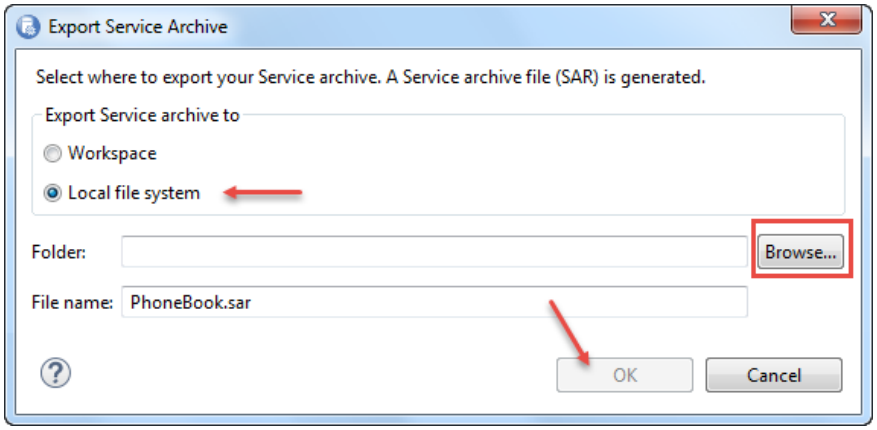


Note: You could, if you wished, create and run a test case for this like you did for the "AddContact" service. If you wish to do that, follow the instructions starting on page 70 and use "PhoneBook" for the name. When you run the test case you will need to supply the `IN-COMMAND` for the function you wish to perform: `ADD`, `DISPLAY`, `UPDATE` or `DELETE`.

☐ Right click on your new service and then select "Export transaction service archive":

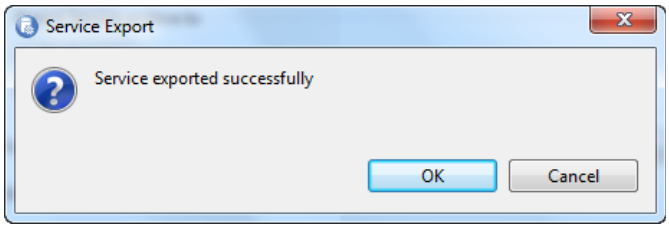


☐ Then select "Local file system[54]," browse for the folder you want to put the file in, and click "OK":



Note the location here:

☐ You get confirmation of export:



Click on "OK".

The SAR file is the key input file to the z/OS Connect EE V2.0 API Editor tool. That comes next.

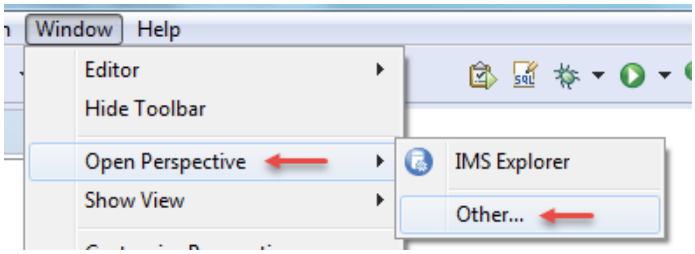
Compose API using z/OS Connect EE V2.0 API Editor

The process here is very much like what we illustrated for CICS back on page 34. The difference is the backend program is different and therefore the data fields are different.

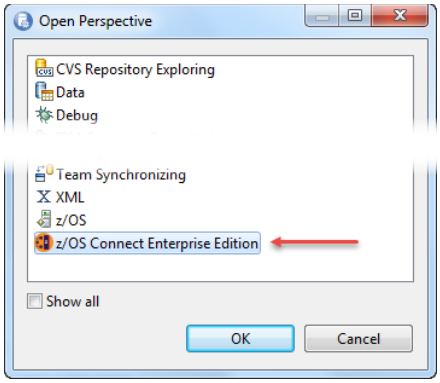
Do the following:

54 If you're familiar with Eclipse workspaces and you want to put the file there, then that's okay. The result is the SAR file is exported to a file system location. When you open the z/OS Connect EE V2.0 API Editor you will import this file, so you need to know where it went.

- ☐ Open the Eclipse tool (z/OS Explorer or IMS Explorer) in which you installed the z/OS Connect EE V2.0 API Editor plugin[55].
- ☐ From the menu bar, select *Window* → *Open Perspective* and select "Other":

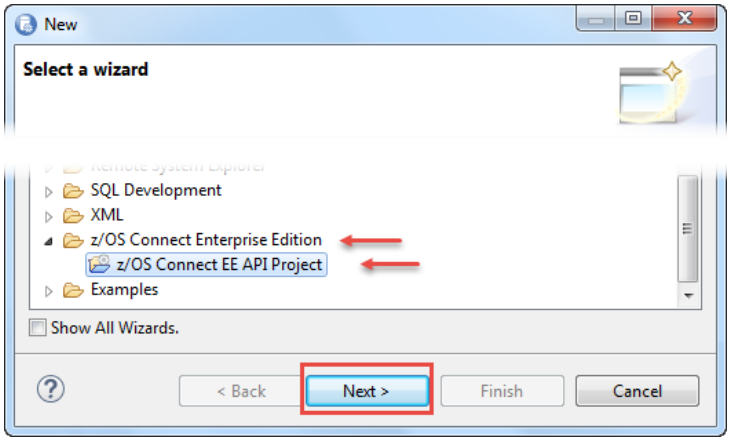


- ☐ Then in the list of perspectives, select "z/OS Connect Enterprise Edition":



Click "OK".

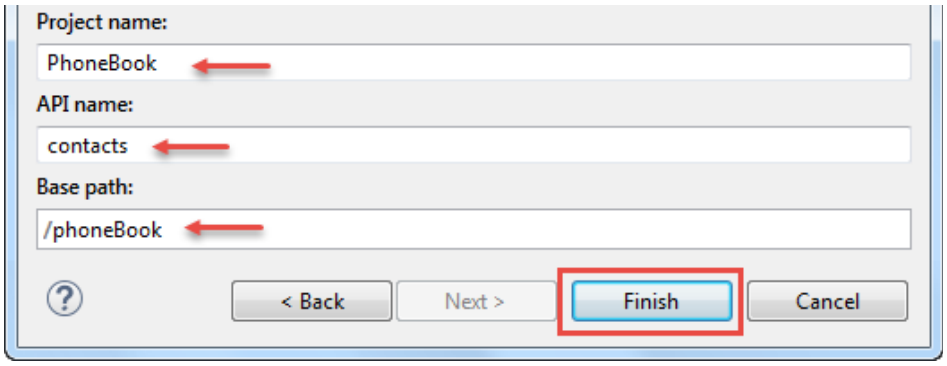
- ☐ Now from the menu bar select *File* → *New* and then "Other".
- ☐ Then scroll down and locate the "z/OS Connect Enterprise Edition" folder, open that and select "z/OS Connect EE API Project":



Then click "Next".

- ☐ For the project name values, provide the following, then click "Finish." See the notes that follow:

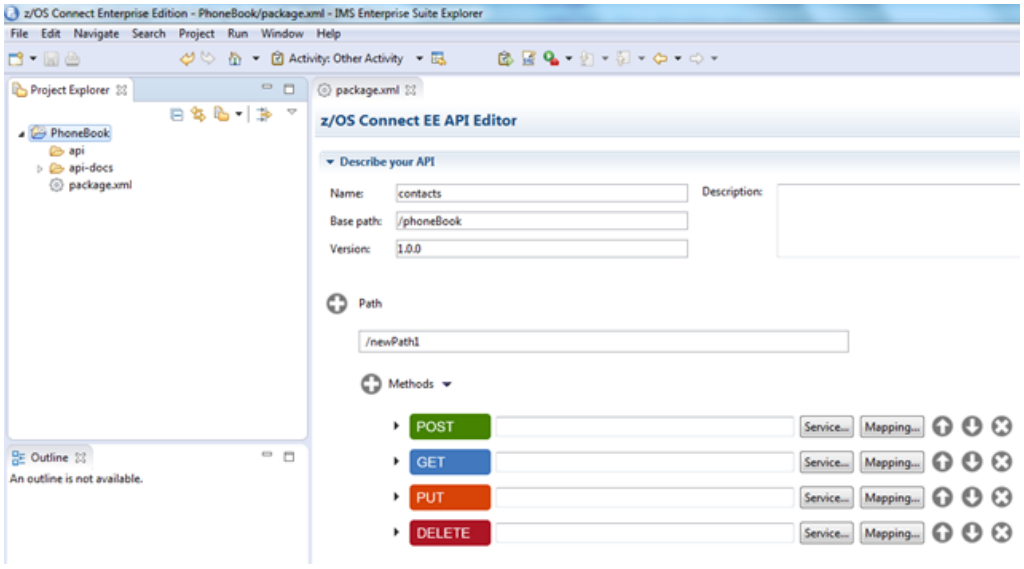
55 The z/OS Connect EE V2.0 API Editor is the same in both. If your primary backend system of record is IMS, then having the API Editor in IMS Explorer is very convenient.



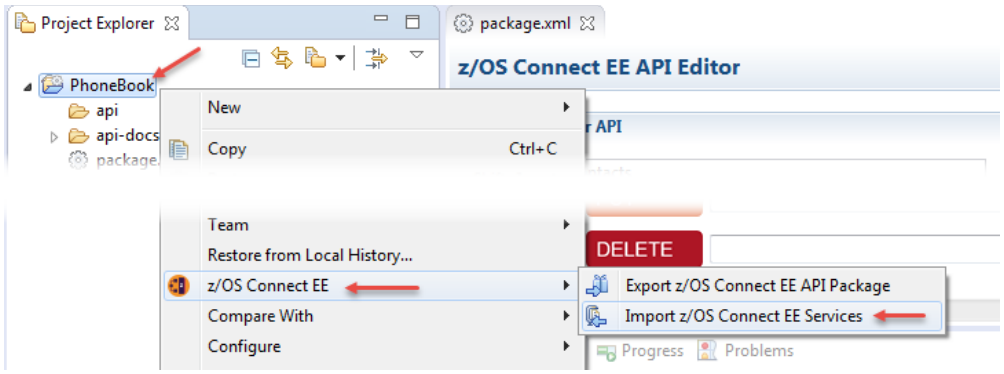
Notes:

- These values may be whatever you like, but they do have meaning.
- The "Project name" value is what Eclipse uses when it creates the project.
- The "API name" value the name z/OS Connect EE V2.0 will know this API.
- The "Base path" value is what will by REST clients in the URI they send in to invoke this API. This was the value planned under "Phone Book API design" starting on page 102.

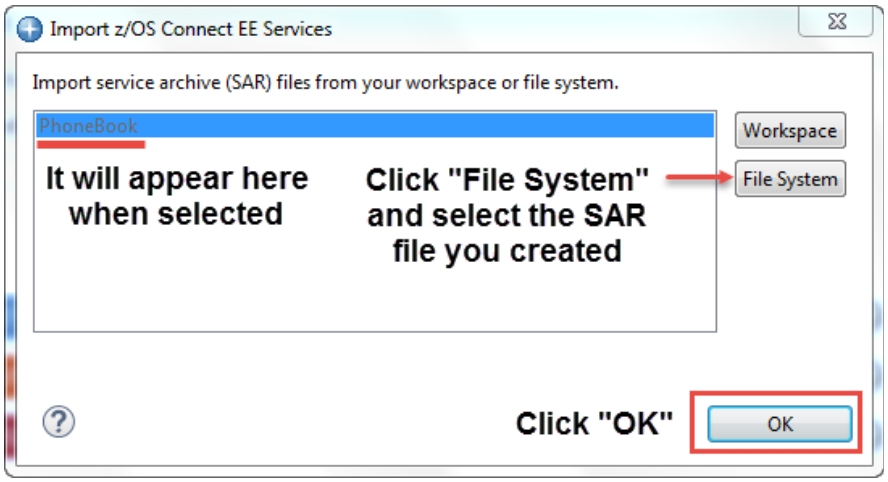
☐ You should now have a screen that looks something like this:



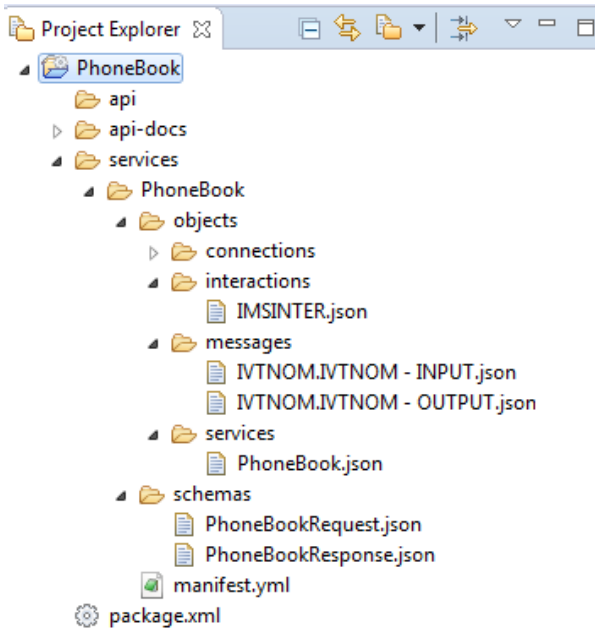
☐ In the "Project Explorer" tab, right click on the "PhoneBook" folder and then select "z/OS Connect EE" and then "Import z/OS Connect EE Services":



☐ Then, use the "File System" button[56] and navigate to where you stored the SAR file you exported earlier (we had you note the location on page 75). Select the SAR file and it will appear in the field on this panel. Then click "OK":



☐ In the "Project Explorer" window (upper left), expand the folders and you'll see something like the following:



Those artifacts were brought in with the SAR file. The SAR file, was produced by the IMS Explorer tool based on your creation of the service function for the Phone Book application.

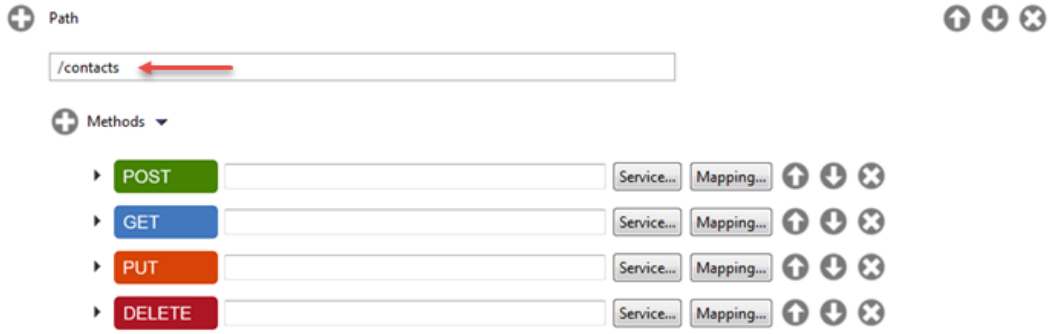
☐ Recall the plan we made for the API (from "Phone Book API design" on page 102):

Action	*Verb*	*URI (base path + API path)*
Add	`POST`	`/phoneBook/contacts`
Update	`PUT`	`/phoneBook/contacts/{lastName}`
Display	`GET`	`/phoneBook/contacts/{lastName}`
Delete	`DELETE`	`/phoneBook/contacts/{lastName}`

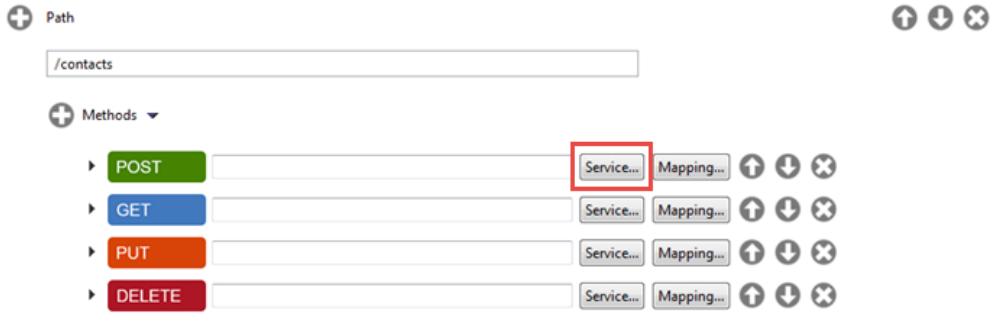
56 If you stored the SAR file in the workspace, use the "Workspace" button.

We are planning two paths here: one as just `/phoneBook/contacts` and the other as `/phoneBook/contacts/{lastName}`, where `{lastName}` is a path parameter. That will be the basis for the work in the API Editor, which comes next.

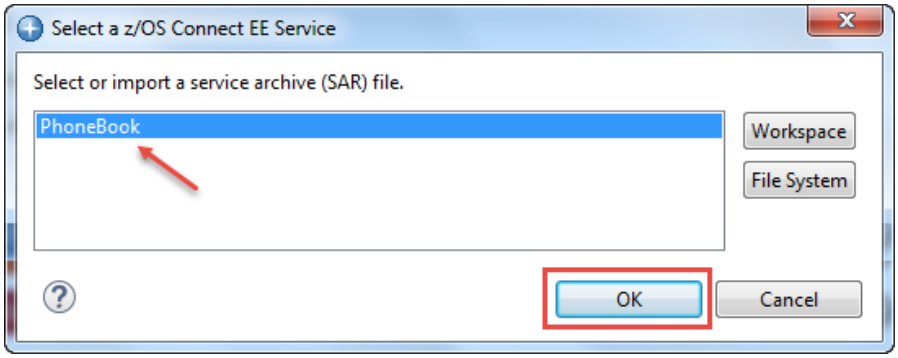
☐ We'll do the POST verb definition first. Set the "Path" value to **`/contacts`** as shown here:



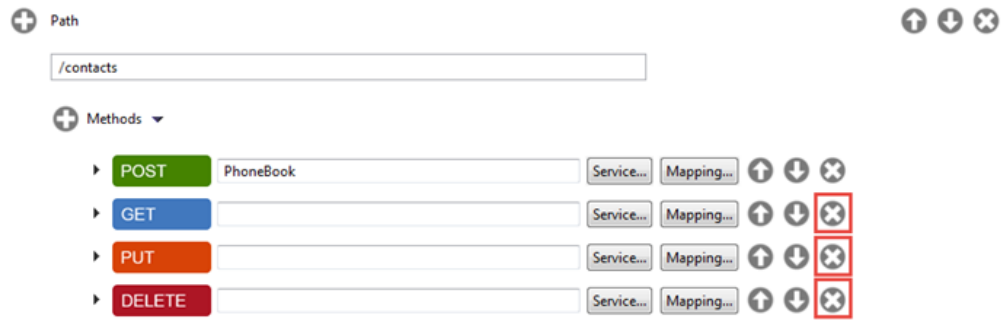
☐ Next, click on the "Service..." button that's on the POST row:



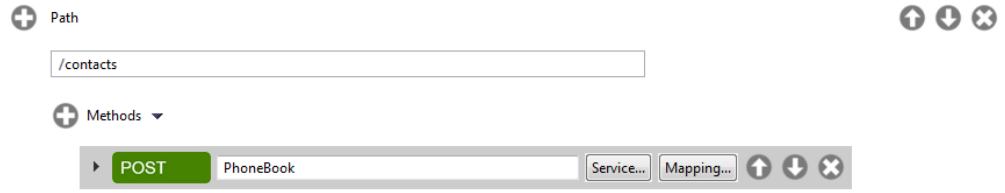
And then select the "PhoneBook" service (it should be the only one present), then click "OK":



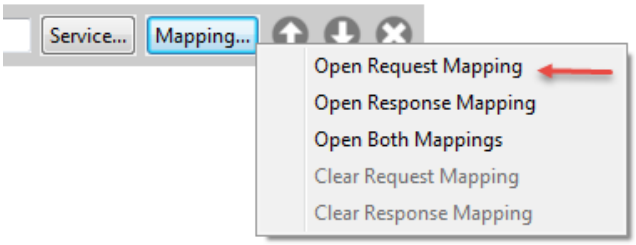
☐ For *this* API path (just `/contacts` with no path parameter) we will use only POST, so we can get rid of the GET, PUT and DELETE verbs. Click the "x" symbols to the right of each of those to remove them:



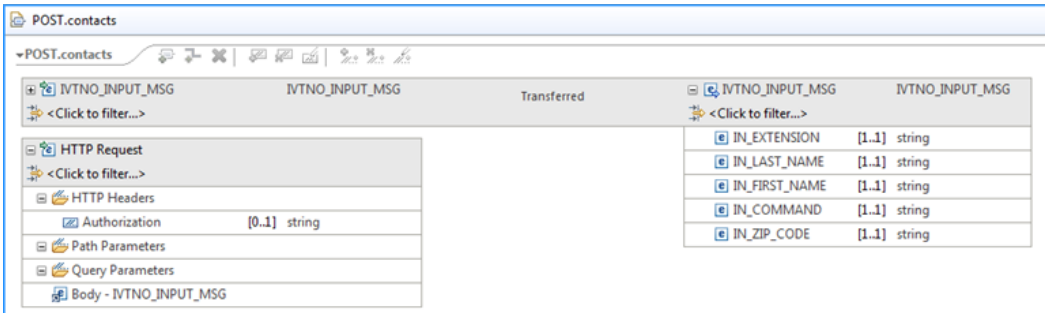
You should be left with:



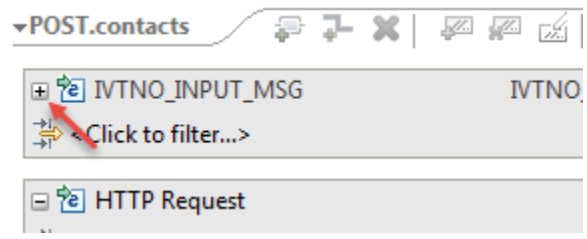
- ☐ Before you can do the field mappings you have to save your changes. From the menu bar, select *File* → *Save*[57].
- ☐ Now, click on the "Mapping..." button and select "Open Request Mapping":



You should then see:

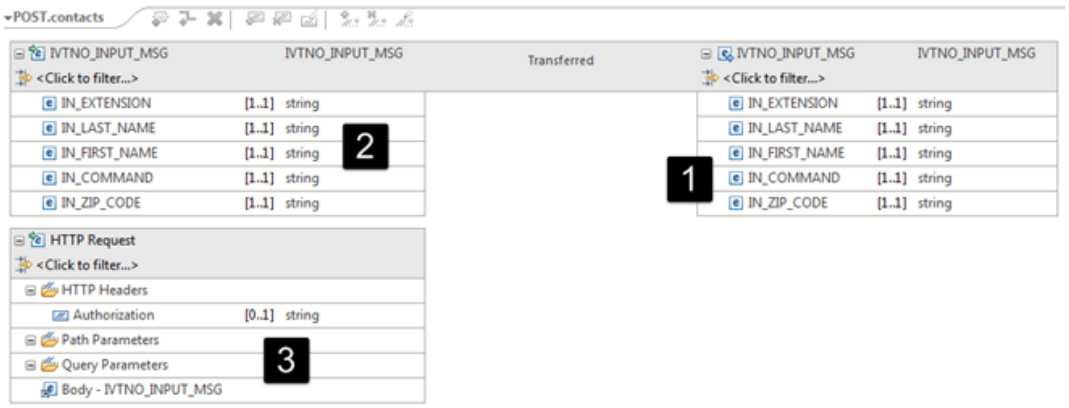


- ☐ In the upper-left, next to IVTNO_INPUT_MSG, click the little "+" sign symbol to expand the field:



You should now see:

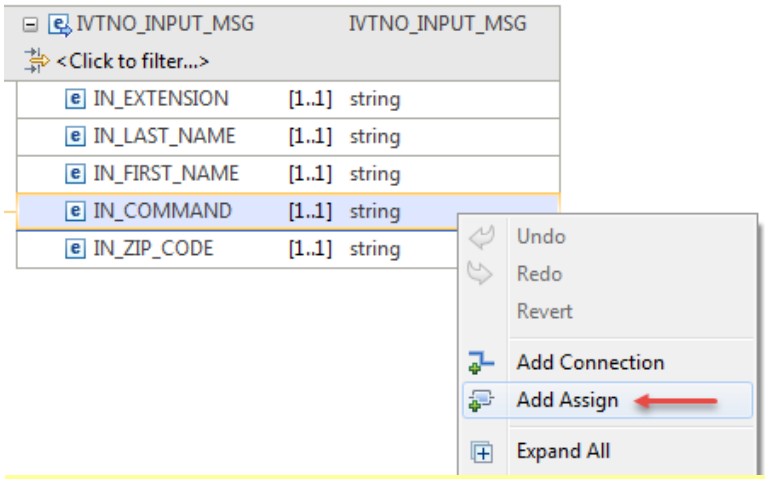
57 Or use Ctrl+s to save.



Notes:

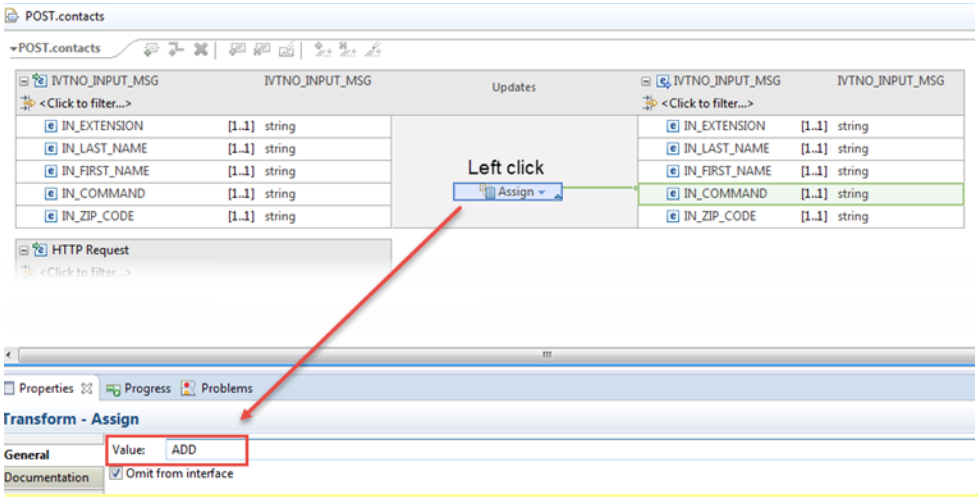
1. The right side represents the fields exposed by the service definition for the request.
2. The left side represents the fields that will be exposed to the REST client. Initially it's a one-for-one mapping of fields from right-to-left. You will soon change that by assigning a fixed value to `IN_COMMAND`, leaving *four* fields exposed to the REST client.
3. The "HTTP Request" section represents values from the HTTP request (such as path and query parameters) that can be mapped to the fields on the right side. For this POST action our path was just `/contacts`, so there is no path or query parameters.

☐ On the right side (block "1") in the picture above, right-click on the `IN_COMMAND` field and select "Add Assign":



Note: The `IN_COMMAND` field is one we do *not* wish to expose to the REST client. We know to add a contact the IMS transaction calls for a command of `ADD`. The POST verb is whe are using to "add a contact." Therefore, we can have z/OS Connect EE V2.0 assign the value `ADD` when this API's POST action is invoked.

☐ A little "Assign" box will appear to the left of the `IN_COMMAND` field. Left click on that and then type **`ADD`** in the properties tab field at the bottom of the screen:

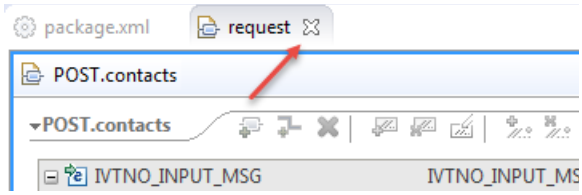


Note: The "Omit from interface" checkbox under the ADD value tells z/OS Connect EE V2.0 to hide the `IN_COMMAND` field from the REST client. Instead, a static value of `ADD` will be assigned to that field on each request for this API when the POST verb is used.

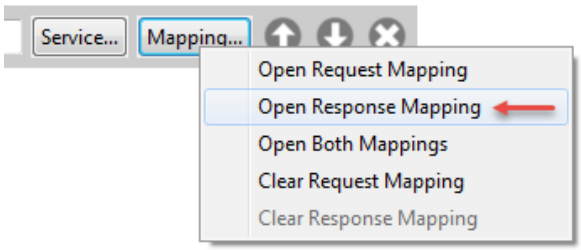
Four fields remain exposed to the REST client. Those fields represent the four data elements of the contact record: last name, first name, phone extension and zip code. Those fields will be carried in the JSON body for the request.

For the "add a contact" request mapping that is all that's needed.

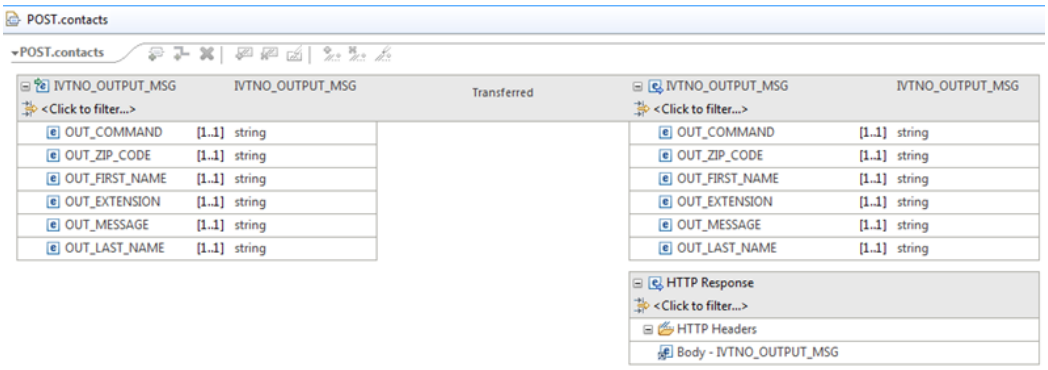
- ☐ Save the mapping: *File* → *Save* (or Ctrl+s).
- ☐ Close the request mapping tab:



- ☐ Now click on the "Mapping..." button again, but this time select "Open **Response** Mapping":

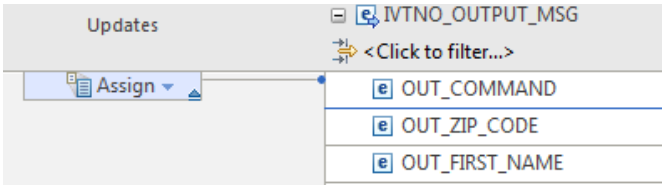


- ☐ You will see the fields that will be sent back to the REST client on the response:

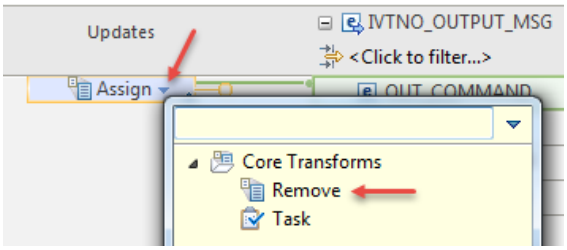


Six fields appear on the right side, but we don't want to send `OUT_COMMAND` back to the REST client[58]. So we will "remove" that field.

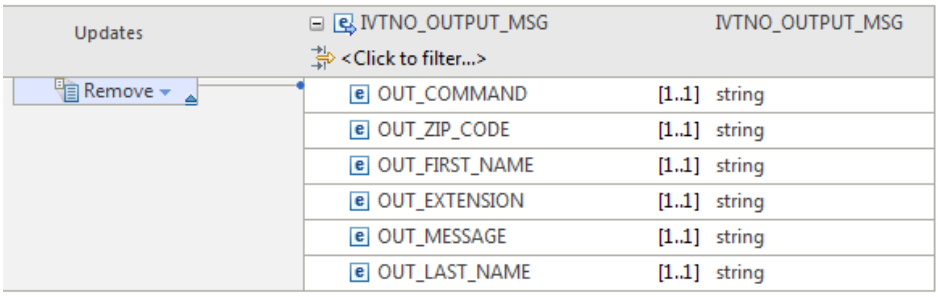
☐ On the right side of the response mapping display, right click on `OUT_COMMAND` and select "Add Assign". You will then see:



☐ Click on the little down arrow symbol in the "Assign" box, then select "Remove":



You should now see:



This means z/OS Connect EE V2.0 will hide `OUT_COMMAND` for the response, but will pass through (in a JSON object) the other five fields. `OUT_MESSAGE` will carry the message produced by the transaction (indicating success or failure of the "add" action). The other fields will provide confirmation of the data for the added contact record.

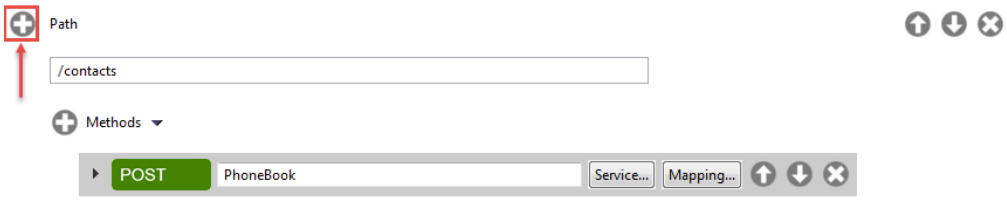
☐ Save the mapping: *File* → *Save* (or Ctrl+s).

58 This field provides the command sent in on the request. We assigned the static value of `ADD` to the request field. We *could* expose this field back to the REST client as confirmation of the command used on the request. Instead, we will hide it and let the `OUT_MESSAGE` (which will indicate success or failure of the "add" operation) carry the meaningful information about the "add a contact" action.

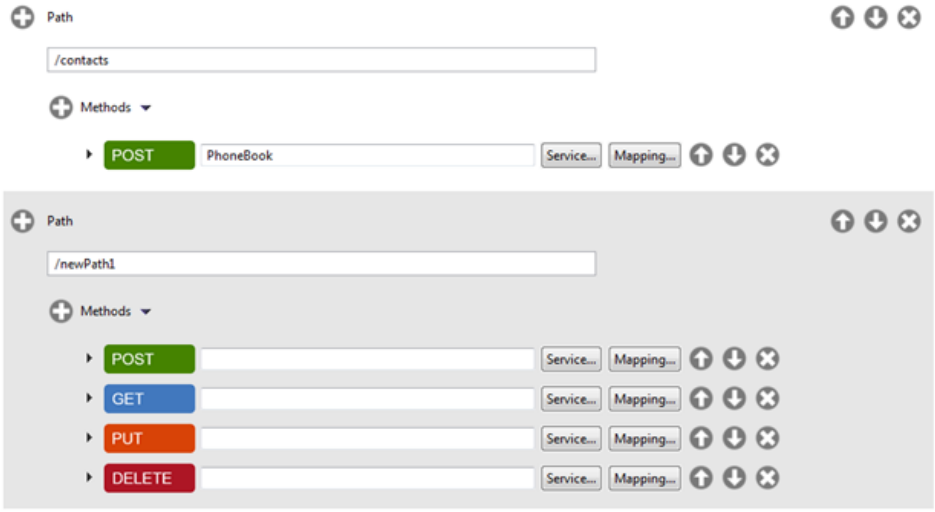
☐ Close the response mapping tab.

We have completed the POST verb "add a contact" portion of the API. Now let's create the other three. Do the following:

☐ Click on the "+" symbol next to "Path" as shown here:



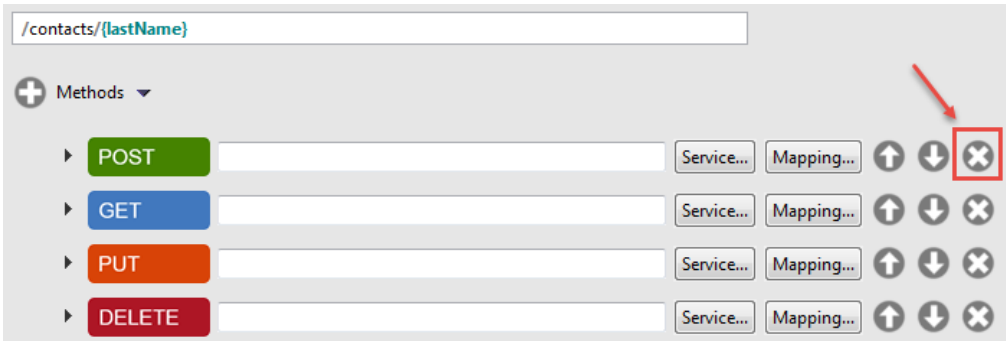
You should see a new "Path" section with a new set of HTTP verbs:



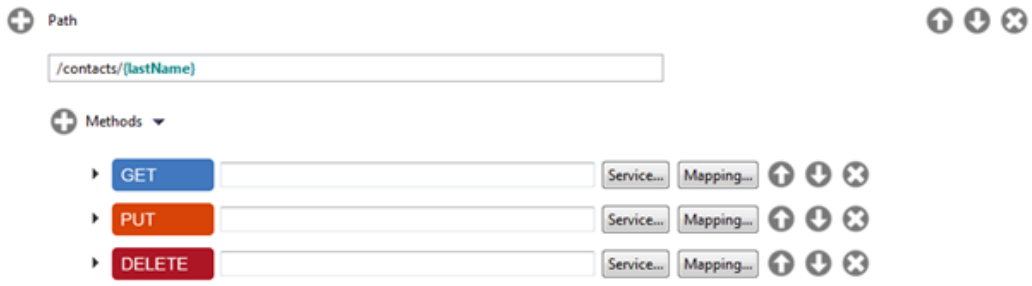
☐ For the "Path," provide the string **`/contacts/{lastName}`** as shown here:

The string `/{lastName}` represents a path parameter. The REST client will send the last name in on the GET, PUT, or DELETE action. In the request mapping editors for each of those actions you will map the path parameter to the `IN_LAST_NAME` field of the transaction.

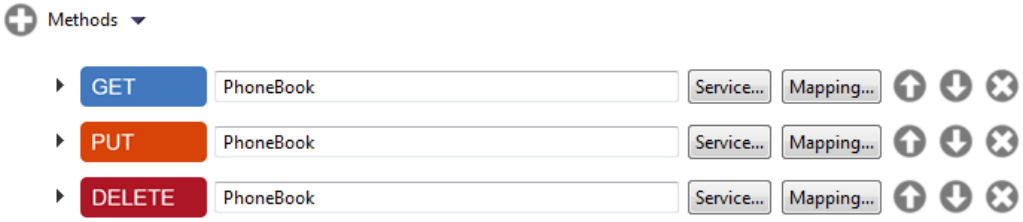
☐ We already defined the POST action, so we don't need it for this API Path. Remove it by clicking on the "x" symbol to the right:



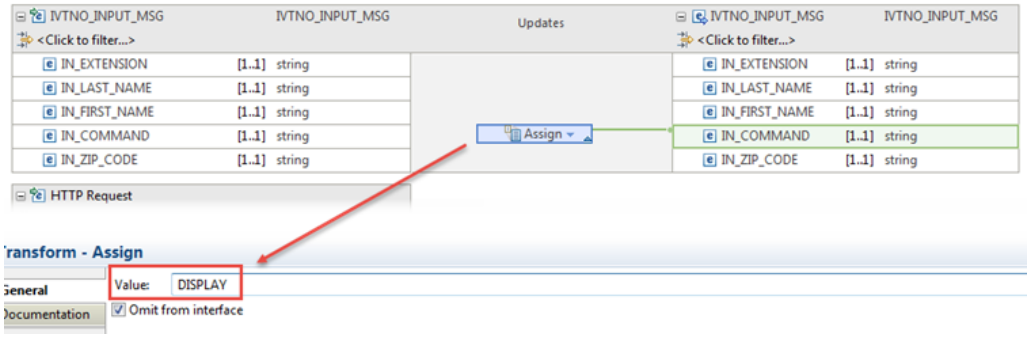
That leaves just GET, PUT, and DELETE:



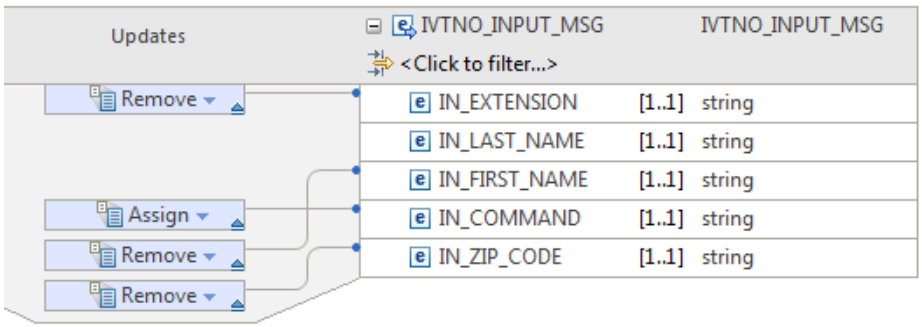
- ☐ For each (GET, PUT, and DELETE), click on the "Service..." button and select the "PhoneBook" service. The result should look like this:



- ☐ Save the changes: *File* → *Save* (or Ctrl+s).
- ☐ For the GET row, click on "Mapping..." and select "Open **Request** Mapping."
- ☐ Start by assigning the value **`DISPLAY`** to the `IN_COMMAND` field:



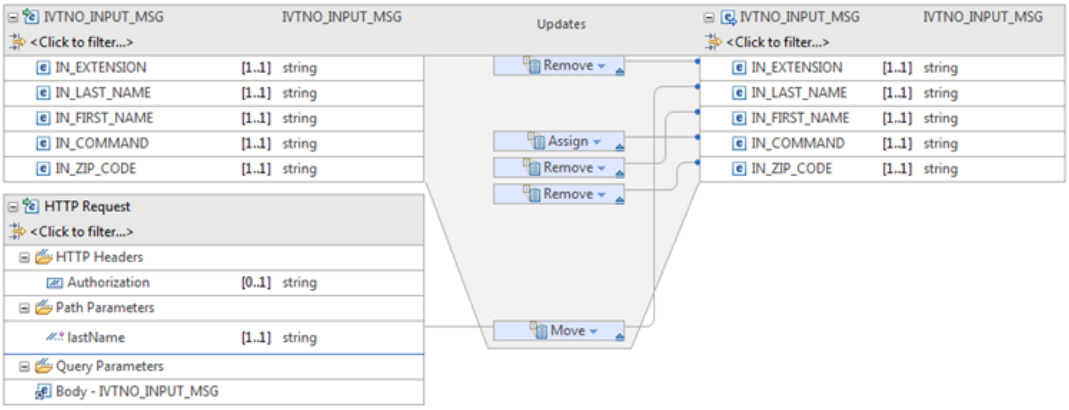
- ☐ Next, "Remove" the `IN_EXTENSION`, `IN_FIRST_NAME` and `IN_ZIP_CODE` fields:



Note: To display a contact record, all we need on the request is the last name value (which is the unique key for the Phone Book sample program database). We do *not* need to send in the first name, phone extension or zip code. Therefore, we remove those fields and they are hidden from the client for this request.

You may see the editor move the boxes so they're not exactly lined up with the field row. That's okay. Follow the connector lines and you will see the mapping applies to the fields you selected.

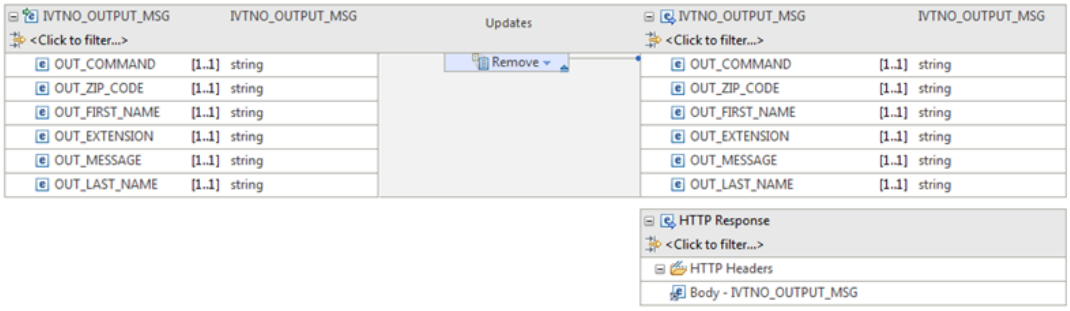
- ☐ Finally, we need to map the path parameter "lastName" from the HTTP Request section to the `IN_LAST_NAME` field on the right side. This is done by left-clicking on the path parameter field, then moving the cursor over to the field and dropping the line there. The result is this:



  And with that the request mapping for the "display a contact" action is complete: the `IN_COMMAND` field has the value **`DISPLAY`** assigned to it; the path parameter is "moved" to the `IN_LAST_NAME` field; and the first name, phone extension and zip code fields are removed.

  No JSON body is sent in with this request. All the information needed to display a contact record is carried in on the URI with the path parameter.

- ☐ Save the changes: *File* → *Save* (or Ctrl+s).
- ☐ Close the request mapping tab.
- ☐ Open the *response* mapping tab for the GET action.
- ☐ The mapping for the response is relatively simple. We want to "remove" the `OUT_COMMAND` field[59], but return to the REST client all the other fields. The mapping should look like this:



  The `OUT_MESSAGE` field is important to return to the client. If a "lastName" value is on the request and that last name is not found in the database, the `OUT_MESSAGE` will indicate this with a "contact not found" message. The other fields (last name, first name, extension and zip) would be empty.

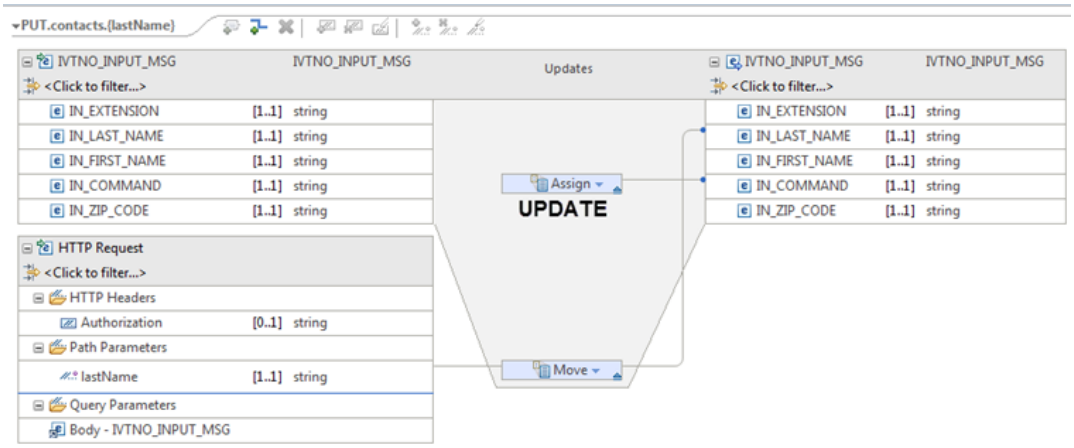
  If the last name is found, then the contact information is provided for last name, first name, phone extension and zip code.

- ☐ Save the changes: *File* → *Save* (or Ctrl+s).
- ☐ Close the response mapping tab.

59 We could hide this field on the service mapping and we would eliminate the need to "remove" it here. We left it exposed on the service mapping "just in case" an API designer wanted that field. In our case we don't want that field, so we remove it.

You've complete the GET action (display a contact) mapping. Now do the final two. Our instructions here will get less explicit. We will supply the desired mappings and commentary about why things are mapped the way they are.

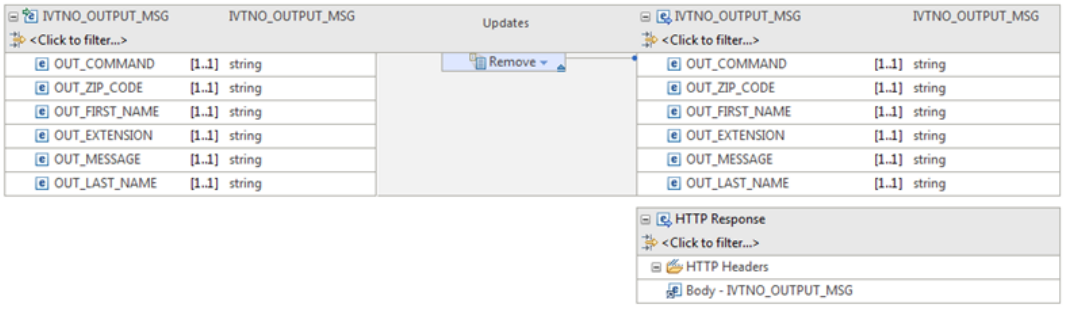
- ☐ Open the *request* mapping for PUT (update a contact). Using the techniques seen in the previous mappings, map this request as follows:



Notes:

- The command to update a record is **`UPDATE`**, so we assign that static value to the `IN_COMMAND` field.
- The `lastName` path parameter is the key used by the transaction to understand which record is being updated[60]. We move that value from the path parameter to the `IN_LAST_NAME` field.
- The leaves just `IN_FIRST_NAME`, `IN_EXTENSION` and `IN_ZIP_CODE` to be supplied in a JSON body for this request.
- The Phone Book sample program will update with the information it receives. If you sent in just the update for extension field, but left the first name and zip code fields blank, it would update the extension and make the first name and zip code fields blank.

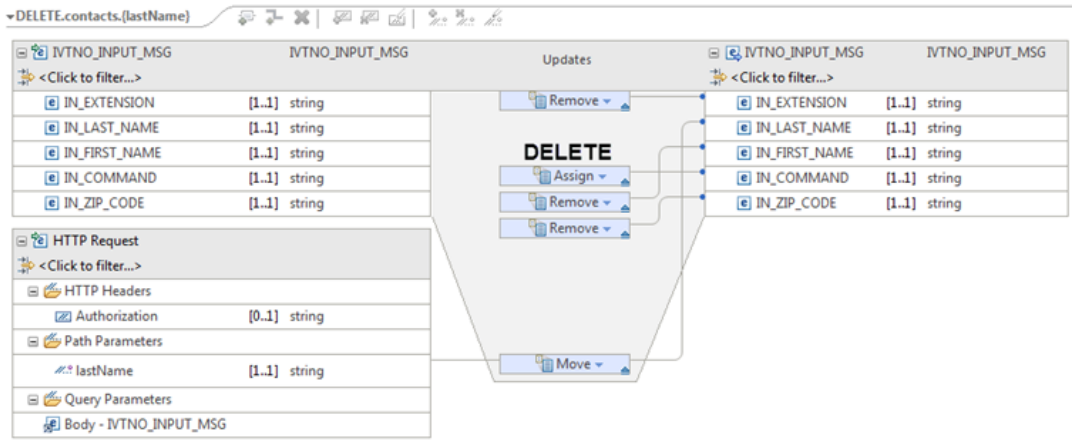
- ☐ Save the changes: *File* → *Save* (or Ctrl+s).
- ☐ Close the request mapping tab.
- ☐ Open the *response* mapping for PUT (update a contact). Remove the `OUT_COMMAND` field, then save and close the tab:



We return all the contact record fields along with the output message. This allows the client to see the updated values. The output message is important to provide because if the `lastName` path parameter value was not found, then the update action would not succeed. That would be reflected in the `OUT_MESSAGE`.

60 This program does not allow the last name to be updated.

☐ Open the *request* mapping for DELETE (delete a contact). Using the techniques seen in the previous mappings, map this request as follows:



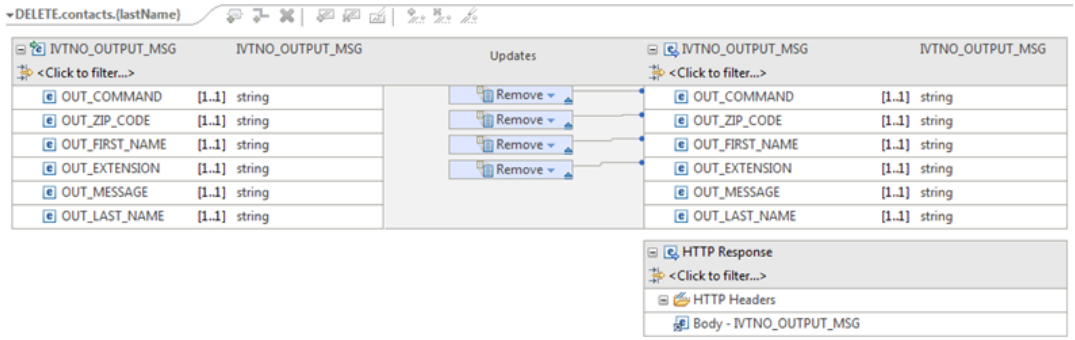
Notes:

- The command to update a record is **`DELETE`**, so we assign that static value to the `IN_COMMAND` field.
- The `lastName` path parameter is the key used by the transaction to understand which record is being deleted. We move that value from the path parameter to the `IN_LAST_NAME` field.
- We don't need to supply any other value for delete, so we remove `IN_FIRST_NAME`, `IN_EXTENSION` and `IN_ZIP_CODE` from the request.

☐ Save the changes: *File* → *Save* (or Ctrl+s).

☐ Close the request mapping tab.

☐ Open the *response* mapping for DELETE (delete a contact). Using the techniques seen in the previous mappings, map this request as follows:



Notes:

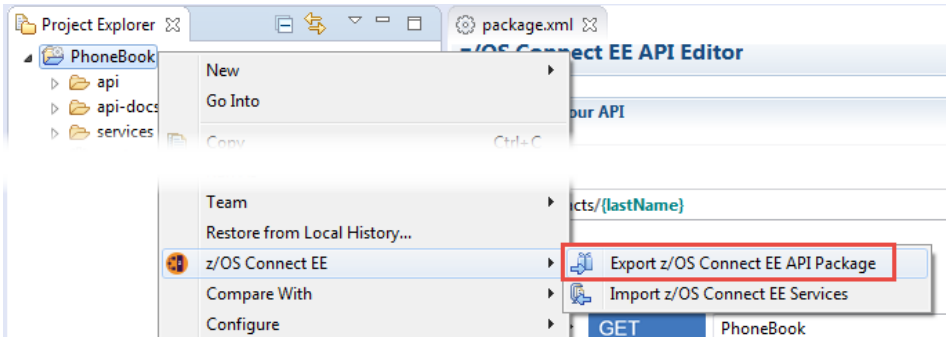
- When we delete a record, all we are interested in is whether the action succeeded (which will be indicated in `OUT_MESSAGE`) and the record that was deleted (as indicated by `OUT_LAST_NAME`).
- If the last name supplied on the path parameter is not found in the contact database, then that result will be indicated in `OUT_MESSAGE`.
- The other fields (`OUT_COMMAND`, `OUT_ZIP_CODE`, `OUT_FIRST_NAME` and `OUT_EXTENSION`) are not of interest on a delete action. So they are removed from the response.

☐ Save the changes: *File* → *Save* (or Ctrl+s).

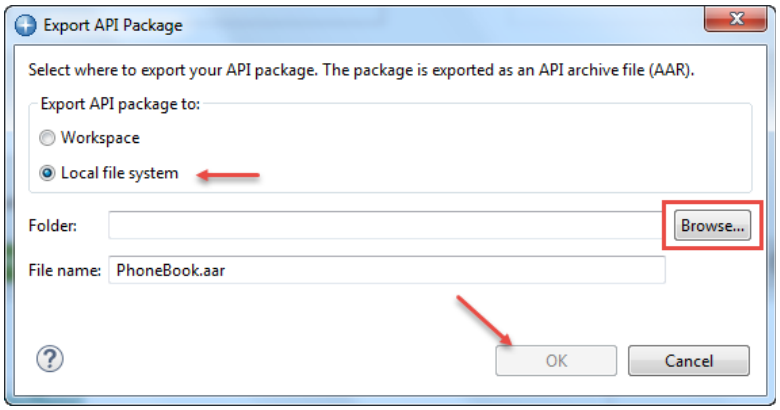
☐ Close the response mapping tab.

You're ready to export the API Archive file and deploy it to z/OS Connect EE V2.0. Do the following:

- ☐ In the Project Explorer, right click on the "PhoneBook" folder, then *z/OS Connect EE → Export z/OS Connect EE API Package*:



- ☐ Select "Local file system," then specify the output folder and click "OK":



Deploy API

To deploy the API into your z/OS Connect EE V2.0 server, do the following:

- ☐ Upload the `PhoneBook.aar` file *in binary* to some location on your z/OS system.
- ☐ Open a Telnet, SSH or OMVS window to your system.

> **Note:** The deployment process will create directories under the Liberty server's `/resources` directory, so the ID you log in with (or su to) must have authority to create directories there.

- ☐ The deployment process will create directories, so the ID you log in with (or su to) must have authority to create directories under the Liberty server's `/resources` directory.
- ☐ Change directories to the `/<install_path>/bin` directory for z/OS Connect EE V2.0.
- ☐ Make sure `JAVA_PATH` is set and points to your 64-bit Java:

```
export JAVA_HOME=path_to_your_64-bit_Java_SDK
```

- ☐ Issue the following command[61] [62] *as one line*:

```
./apideploy -deploy -a /<path>/PhoneBook.aar -p
      /var/zosconnect/servers/<server_name>/resources/zosconnect/apis
```

61 The Knowledge Center article on the apideploy utility is found here: http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.base.doc/topics/api_deploy_package.html

62 You can put this in a shell script as well. Be sure to set `JAVA_HOME` variable and provide path to `apideploy` location.

Where:

- *<path>* is where your uploaded `PhoneBook.aar` file exists
- *<server_name>* is your server name

Note: The target path directories will be created if they do not exist.

You should see:

```
BAQD0007I: The contacts API is created successfully in the server API
path /var/zosconnect/servers/<server>/resources/zosconnect/apis/contacts.
Restart the server to deploy the API.
```

☐ Check the ownership of the directories for the deployed API, starting at:

`/var/zosconnect/servers/<server_name>/resources/zosconnect/`

The ownership may be that of the ID under which the `apideploy` command was run. That may be okay provided your Liberty z/OS server ID has `READ` to those directories and files.

If you wish, you can set the ownership back to your Liberty z/OS server ID and group:

```
cd /var/zosconnect/servers/<server_name>/resources
chown -R <serverID>:<serverGroup> ./zosconnect
```

☐ Stop your z/OS Connect EE V2.0 server.

☐ Update the `server.xml` and add the following:

```
:  (XML before this point removed to save space in document)
 <authorization-roles id="zos.connect.access.roles">
     <security-role name="zosConnectAccess">
         <user name="Fred"/>
     </security-role>
 </authorization-roles>

 <zosconnect_zosConnectAPIs location="">
     <zosConnectAPI name="contacts" />
 </zosconnect_zosConnectAPIs>

:  (XML after this point removed to save space in document)
```

Notes:

- The `location=""` means the default location for APIs is used, which is under the server's `/resources/zosconnect/apis` directory. That's the same directory you deployed the API to using the `apideploy` utility.
- The `name=` attribute matches the value you supplied for "API Name" in the API Editor dialog box when you created the API. It also matches the directory name under `/resources/zosconnect/apis` that was created when this API was deployed.

☐ Save the `server.xml` file.

☐ Restart the z/OS Connect EE V2.0 Liberty server.

Validate API

The API is deployed and may now be accessed using any REST client.

We'll start with the REST client. Do the following:

☐ Open a normal browser tab (not the REST client) and send the following URL:

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

Where *<host>* is the host where your Liberty z/OS server is running, and *<port>* is the *secure* https port.

You should receive a certificate challenge because the server certificate is signed by a CA that is not known to the browser. Accept the challenge.

Note: This is done because most REST clients are not very good at handling encryption when the server certificate is self-signed, as is the case with your Liberty z/OS server at the moment.

You will then receive the basic authentication prompt. Supply the ID (**Fred**) and password (**fredpwd**). You should receive in return a difficult-to-read string of JSON that represents all the services that are understood by z/OS Connect EE V2.0.

☐ Open the REST client and set a custom header of "Content-Type" and "application/json". (See page 69 for an example of how you did that earlier.)

☐ Next, *create* a contact by sending in the following request:

HTTP verb:	**POST**
URL:	https//*<host>*:*<port>*/phoneBook/contacts

JSON body:	{
	"IVTNO_INPUT_MSG" : {
	"IN_EXTENSION" : "123",
	"IN_LAST_NAME" : "API",
	"IN_FIRST_NAME" : "Validate",
	"IN_ZIP_CODE" : "11111",
	}
	}
	Note: if copying from PDF, paste the JSON into a basic text editor and make sure the double-quotes are generic double-quotes and not some other character.

If you are prompted for an ID and password, enter `Fred` and `fredpwd`.

We are looking for a "200 OK" message and a formated JSON response of:

Note the OUT_MESSAGE value indicating the success (or failure) of the action.

☐ Send the exact same request (URI and JSON). We expect this to fail since the last name already exists in the database. You should see:

```
{
  "IVTNO_OUTPUT_MSG": {
    "OUT_ZIP_CODE": "",
    "OUT_FIRST_NAME": "",
    "OUT_EXTENSION": "",
    "OUT_MESSAGE": "ADDITION OF ENTRY HAS FAILED",
    "OUT_LAST_NAME": ""
  }
}
```

☐ *Display* the contents of the contact you created:

HTTP verb:	**GET**
URL:	https//*<host>*:*<port>*/phoneBook/contacts/**API** Where "API" in this case is the last name you created on the earlier POST operation.
JSON body:	*(None needed on the GET operation to display a record.)*

You should see:

```
{
  "IVTNO_OUTPUT_MSG": {
    "OUT_ZIP_CODE": "11111",
    "OUT_FIRST_NAME": "VALIDATE",
    "OUT_EXTENSION": "123",
    "OUT_MESSAGE": "ENTRY WAS DISPLAYED",
    "OUT_LAST_NAME": "API"
  }
}
```

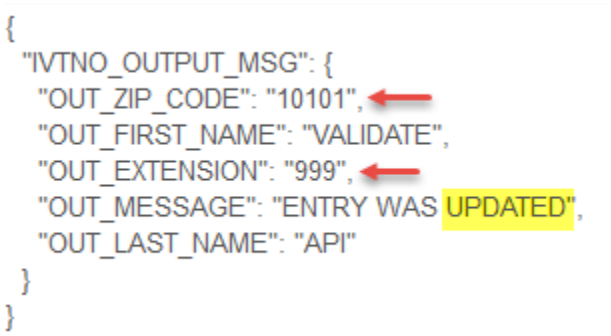
Again, note the OUT_MESSAGE value.

☐ *Update* the contents of the contact:

HTTP verb:	**PUT**
URL:	https//*<host>*:*<port>*/phoneBook/contacts/**API** Where "API" in this case is the last name you created on the earlier POST operation.

JSON body:	`{`
	`  "IVTNO_INPUT_MSG" : {`
	`  "IN_EXTENSION" : "999",`
	`  "IN_FIRST_NAME" : "Validate",`
	`  "IN_ZIP_CODE" : "10101",`
	`  }`
	`}`
	The extension and zip code values are being changed. Note how the last name is *not* included in the JSON. The last name is a path parameter, and in the API Editor we mapped that path parameter to the last name field.
	Note: if copying from PDF, paste the JSON into a basic text editor and make sure the double-quotes are generic double-quotes and not some other character.

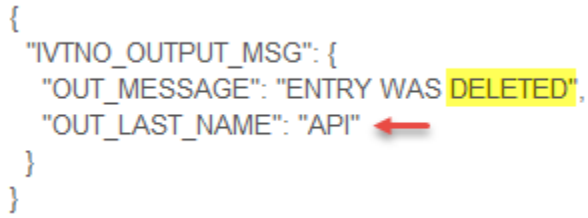
You should see:



☐ Finally, *delete* the contact you created:

HTTP verb:	**DELETE**
URL:	`https//<host>:<port>/phoneBook/contacts/`**API** Where "API" in this case is the last name you created on the earlier POST operation.
JSON body:	*(None needed on the DELETE operation to delete a record.)*

You should see:



Note how only the last name appears in the response. The response mapping for the delete action was planned to return only the last name since that was the key information needed: the record that was deleted.

Miscellaneous Topics

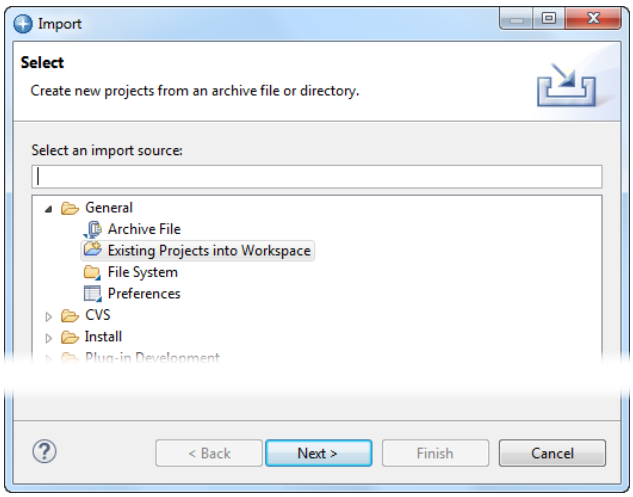
Importing the supplied API project into Eclipse

An API project can be downloaded from Techdocs and imported into your copy of Eclipse. This will provide you with the API for all three services. Do the following:

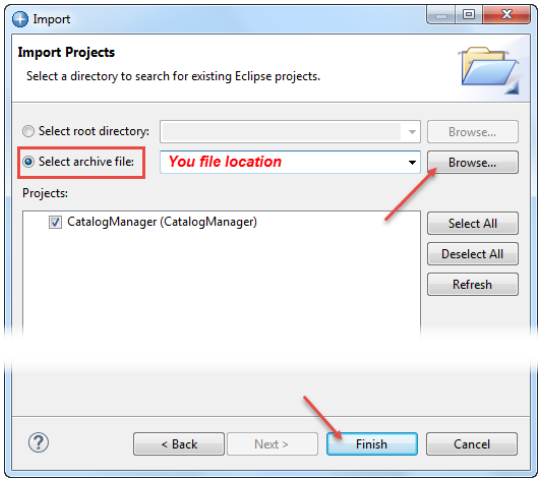
- ☐ Go to:

  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102604

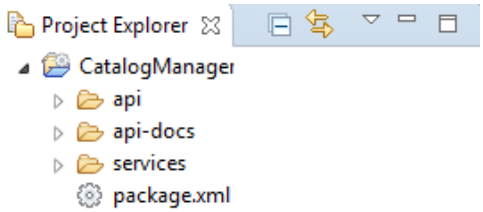
- ☐ Locate the "Getting Started" section of the Techdoc. You will find the project archive named "CatalogManager.zip" file there. Download it to your workstation.
- ☐ In Eclipse, go to *File* → *Import*, then select "Existing Projects into Workspace" from within the "General" folder and click "Next":



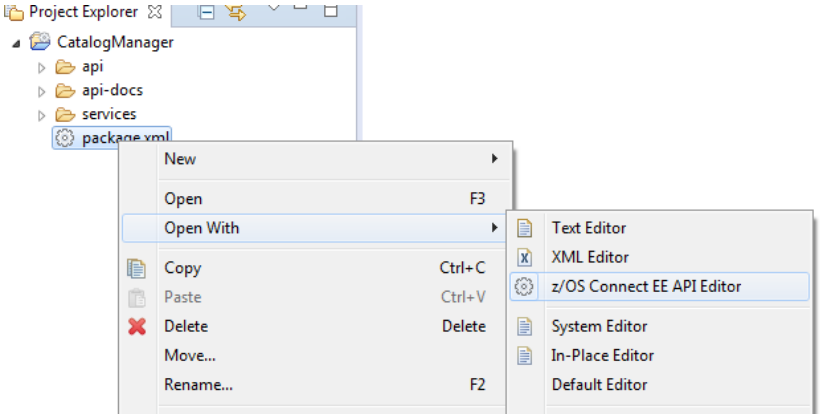
- ☐ Click on the "Select archive file" radio button, then "Browse" and navigate to file on your workstation. Select the file, then click on the "Finish" button:



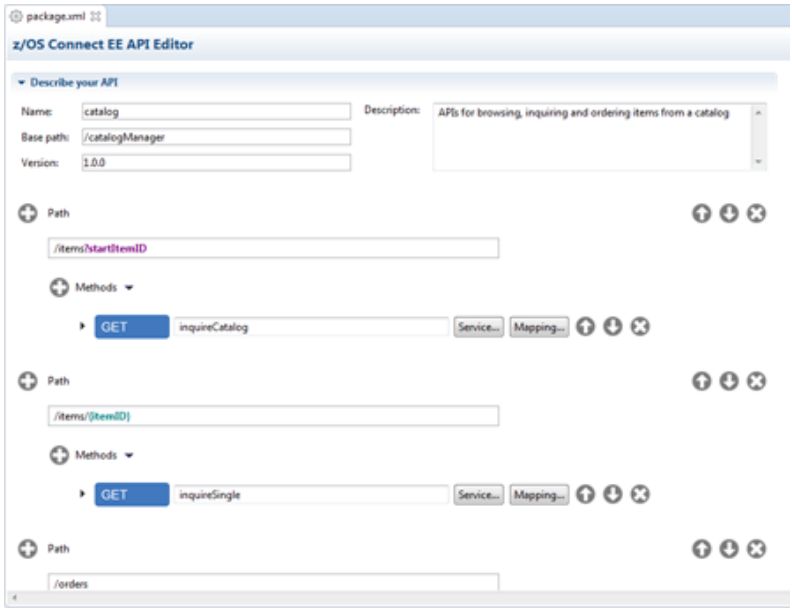
- ☐ In the Project Explorer view, you should then see:



☐ Right click on the "package.xml" file, select "Open with …" and select "z/OS Connect EE API Editor":



You should see:



You now have the complete API:

Path	*Method*	*Service*
`/catalogManager/items?`**`startItemID=`**`value`	GET	inquireCatalog
`/catalogManager/items/`**`itemID`**	GET	inquireSingle
`/catalogManager/orders`	POST	placeOrder

You can export this from the API Editor as an API Archive (AAR) file and deploy to your z/OS Connect EE V2.0 server. The process is the same as shown earlier in this document.

A discussion of why the Catalog Manager API was designed as it was

The Catalog Manager API examples shown in this document were created based on careful consideration of the requirements of the Catalog Manager program and good RESTful design practices. In this section we will discuss the planning that went into the API design.

This is based in part on the following Knowlege Center article:

http://www.ibm.com/support/knowledgecenter/SS4SVW_2.0.0/com.ibm.zosconnect.tool.doc/topics/api_design_intro.html

The way that you design APIs can have a significant impact on their adoption. When done right, APIs that are used inside your organization can enforce consistency and promote efficient reuse. Public APIs that are used outside your organization can expand the reach of your business, by allowing developers to extend the services that you provide. Ease-of-use for consumers is vital for the adoption of the API.

A good API has the following characteristics:

- It is quick to learn and easy to use
- It is focused on a 'resource' not a procedure or method
- The URI is intuitive
- It is stateless
- It uses the HTTP verbs explicitly:
  - POST to create a resource
  - GET to retrieve a resource
  - PUT to change the state of a resource or to update it
  - DELETE to remove or delete a resource

The API that we created for the catalog manager application is summarized in the table below:

API	HTTP verb	URI	JSON request	JSON response
List catalog	GET	`/catalogManager/items?startItemID=`*`value`* *(One line in practice; broken here to fit in column)*	(none)	Array of items First item reference Last item reference Item count Return code Response message
Get details of an item	GET	`/catalogManager/items/`*`itemID`*	(none)	Item details Return code Response message
Order item	POST	`/catalogManager/orders`	Item ref Number quantity	Return code Response message

Notes:

- ***List catalog***

  For listing the catalog we use a **`GET`** request to retrieve a collection of items. The URI consists of a basepath (`/catalogManager`), a resource (`/items`) and a query parameter (`?startItemID=<value>`). The query parameter is used as a filter to specify the item reference number for the start of the collection.

  The JSON response message contains the return code and response message, a collection of items (an array) and other information including the number of items returned.

- ***Get details of an item***

  We use a **`GET`** request to retrieve the details of an item. The URI consists of a basepath (`/catalogManager`), a resource (`/items`) and a path parameter (`{ItemID}`). The path parameter is used to specify the item reference number.

  The JSON response message contains the return code and response message, and the details of the item.

- ***Place order***

  We use a **POST** request to create a new order. The URI consists of a basepath (`/catalogManager`) and a resource (`/orders`).

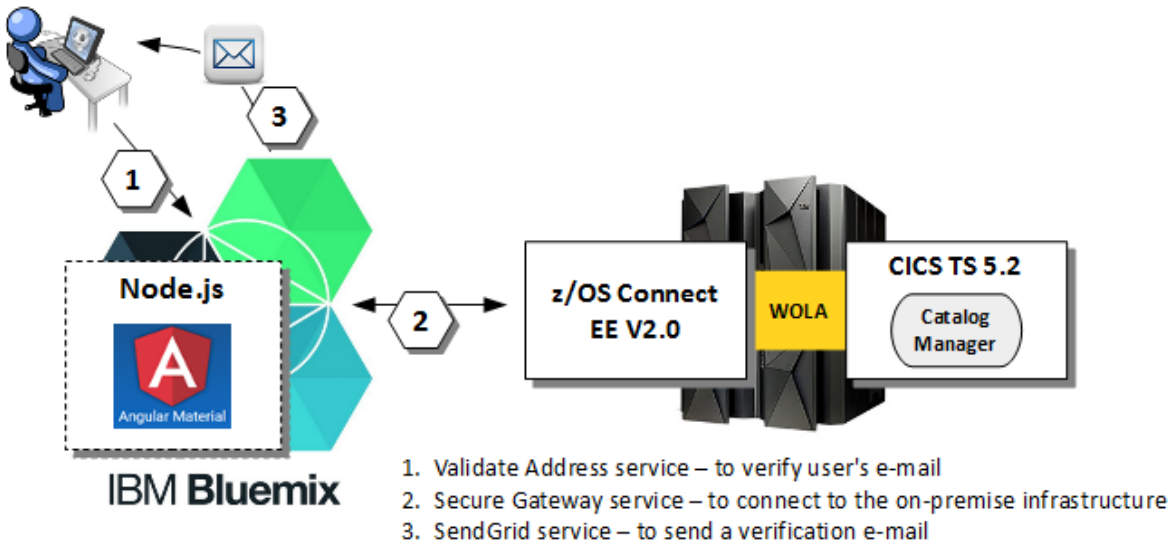
  The JSON request message contains the item reference number and the number of items to be ordered. The JSON response message contains the return code and response message.

An illustration of Bluemix as a client to z/OS Connect EE V2.0

Throughout this document we illustrated the use of a simple REST client browser plugin for interaction with z/OS Connect EE V2.0. That was to keep things simple for a "Getting Started" guide such as this.

But the fuller potential of z/OS Connect EE V2.0 is seen when a programmatic user interface is used, such as a mobile application, or an IBM Bluemix application.

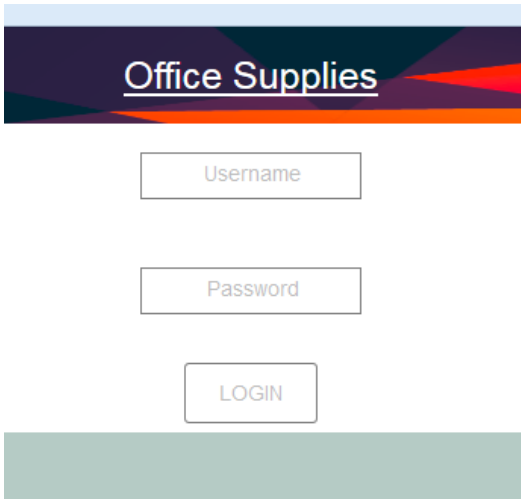
In this section we will guide you through a tour of an IBM Bluemix application using z/OS Connect EE 2.0 to access the CICS Catalog Manager program. The following picture illustrates the environment:



If you would like to experience this use of IBM Bluemix and IBM z/OS Connect EE V2.0, do the following:

☐ Using a browser, go to the following URL address:
`https://officesupplies.mybluemix.net`

☐ When you get to the logon screen:



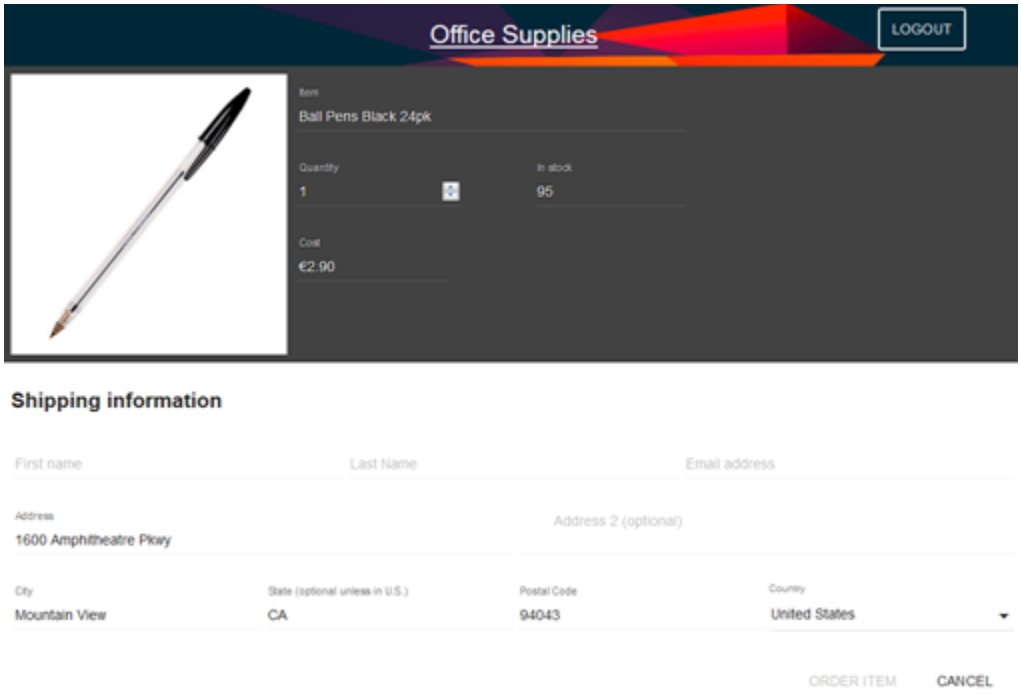
Use the following credentials:

Username	**JeanLeclerc**
Password	**jean**

☐ The items in the catalog manager are presented in graphical format:

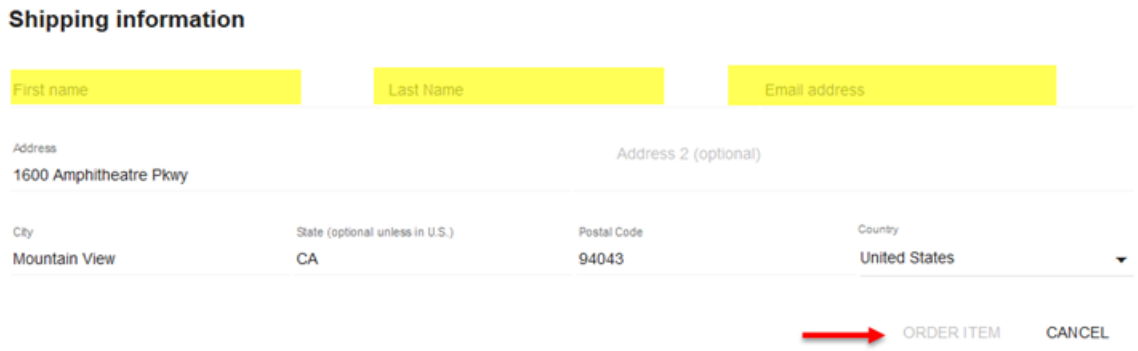
Note: The Node.js application is using the REST API in z/OS Connect EE V2.0 to get to the CICS region and the Catalog Manager sample program. Specifically, it used the "inquire catalog" service API (GET with `/items?startItemID=value`) to retrieve a JSON object from z/OS Connect EE V2.0 listing the items in the catalog. The Node.js application then retrieved images from a MongoDB database and formatted the page you see.

☐ If you click on an item – say the first item, which is the 24-pack of black ball point pens – you get the item details:



Note: Here again, the Node.js application uses the REST API in z/OS Connect EE V2.0. It sends a GET request with URI `/catalogManager/items/{itemID}` . The JSON object received is used to format the page seen above.

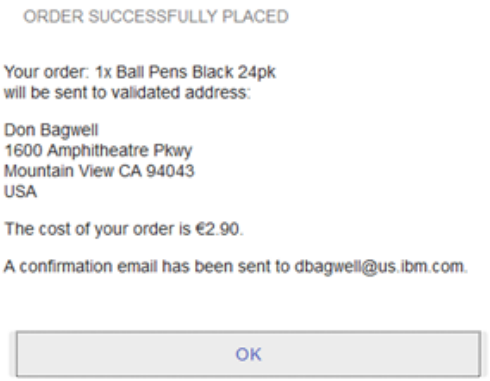
☐ That page has exposed fields for the user to fill in shipping information, including their name and e-mail address:



If you fill in the first name, last name and e-mail fields, the "Order Item" button will un-gray and you will be able to order the item.

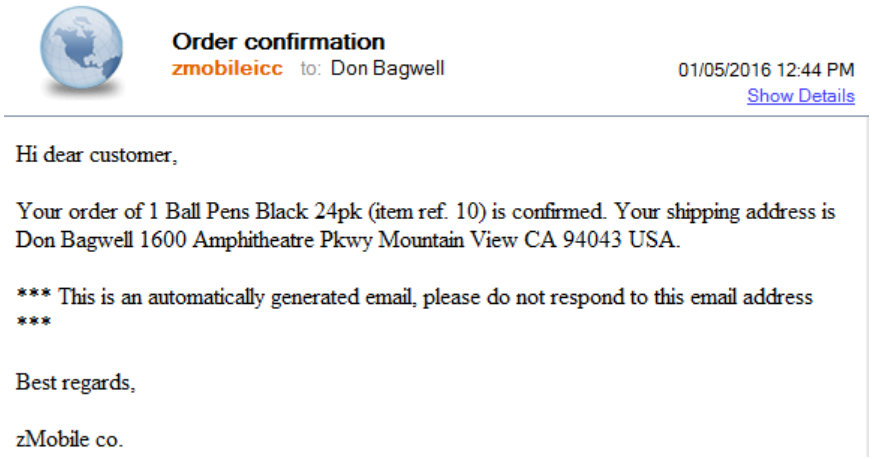
Note: Again the Node.js application uses the REST API in z/OS Connect EE V2.0. It sends a POST request with URI `/catalogManager/orders` and a JSON request message with the item reference number and quantity to be ordered.

- ☐ When you click the "Order Item" button and the order is placed, you receive an on-screen confirmation:



When you click "OK" you are returned to the initial page that shows all the products.

- ☐ In addition, an e-mail is sent to the e-mail address entered on the item detail screen:



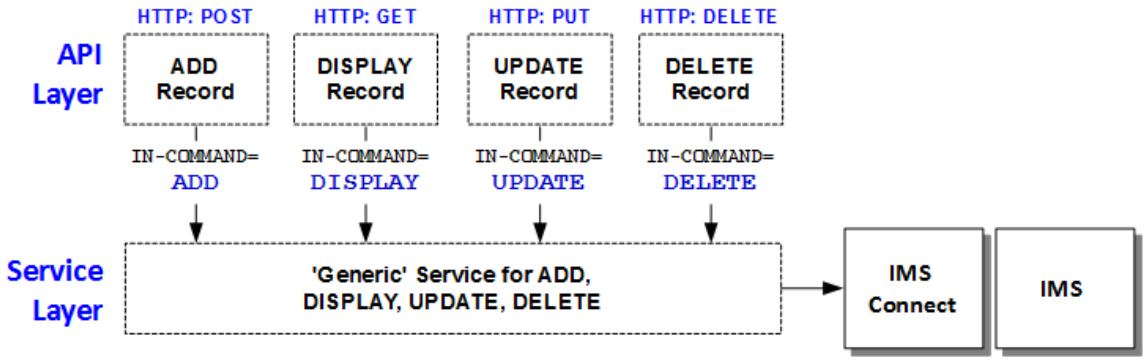
That comes from the e-mail ID of zmobileicc@gmail.com, so if your e-mail is set to filter out GMail, you may have to retrieve it from the filtered folder.

A discussion of why the IMS Phone Book API was designed as it was

The "Phone Book" sample was used in the section for IMS because it's a well-known sample transaction and every IMS installation has access to it. It's a relatively simple transaction with a database behind it. It has four basic functions: **add** a contact record, **display** a contact record, **update** a contact record, or **delete** a contract record.

In the earlier setup exercise ("IMS Explorer definitions, Part 2 (service definition)" starting on page 66) we had you create an "AddContact" service. That service exercised *one* of the four functions of the Phone Book sample. We had you hard-code the value for `IN-COMMAND` to `ADD`, and that's what told IMS to "add a contact record."

With z/OS Connect EE V2.0 we have the opportunity to create a more *generic* service layer for the Phone Book transaction and provide an API layer that exposes the four functions via separate HTTP verbs. So our basic design framework was this:



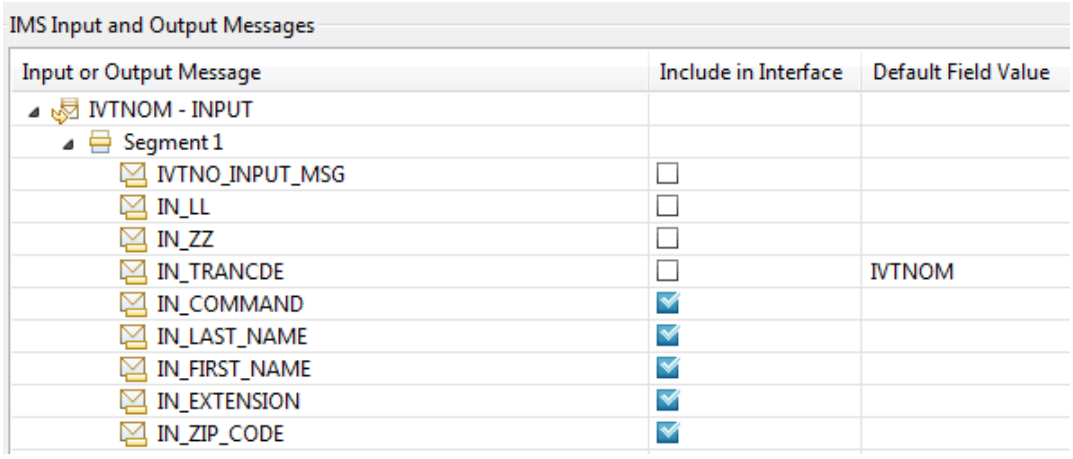
"Generic" Phone Book service

The *most* generic service would be one that exposed every field in the INPUT and OUTPUT data structure, but that's *too* generic. We have the opportunity to use the IMS Explorer service mapping function to tailor the service level to what we need for the APIs.

At the service level, the data field mapping plan for *input* was:

Input Field	*Action*	*Explanation*
IVTNO-INPUT-MSG	Hide	This is a "01" level element for the record. No need to expose this.
IN-LL	Hide	This field is internal to the IMS transaction and there is not meaningful to expose this to the API or to end-user clients.
IN-ZZ	Hide	This field is internal to the IMS transaction and there is not meaningful to expose this to the API or to end-user clients.
IN-TRANCDE	Assign	The value for this is always equal to the defined transaction definition (IVTNOM in our example). Therefore, we can assign this value at the service layer and not expose it to the API or end-user clients.
IN-COMMAND	Expose	The API will pass this value in based on the composition of the API in the z/OS Connect EE V2.0 API Editor tool.
IN-LAST-NAME	Expose	This is the unique key field for the database and is used in all four actions. Therefore, we expose it here and allow the API to pass it.
IN-FIRST-NAME	Expose	Needed for ADD and UPDATE. We expose it here to the API for those two functions. The API for DISPLAY and DELETE will hide this field field from the end-user client.
IN-EXTENSION	Expose	Same as for first name.
IN-ZIP-CODE	Expose	Same as for first name.

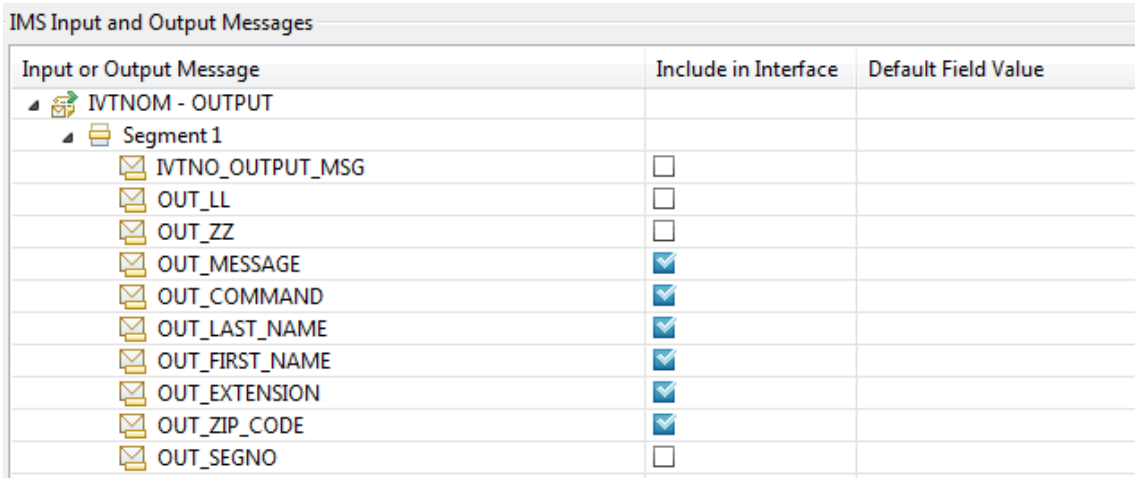
The IMS Explorer service mapping for *input* would look like this:



The plan for the *output* was:

Input Field	*Action*	*Explanation*
IVTNO-OUTPUT-MSG	Hide	This is a "01" level element for the record, so we hide it.
OUT-LL	Hide	This field is internal to the IMS transaction and there is not meaningful to expose this to the API or to end-user clients.
OUT-ZZ	Hide	This field is internal to the IMS transaction and there is not meaningful to expose this to the API or to end-user clients.
OUT-MESSAGE	Expose	This is the key message indicating the result of the action. This is exposed to the API so it can expose it to the end-user client.
OUT-COMMAND	Expose	This field will be made available to the API for exposing to the end-user client if the API designer wishes. A potential use for this field is a simple verification of the command used to achieve the result. This can be hidden if you don't see a use-case for it.
OUT-LAST-NAME	Expose	This field will be made available to the API for exposing to the end-user client if the API designer wishes. For all four actions there is value in displaying this value back to the end-user client.
OUT-FIRST-NAME	Expose	This field will be made available to the API for exposing to the end-user client if the API designer wishes. For ADD, UPDATE and DISPLAY this is useful to the end-user for validation purposes. For DELETE this has less value and can be hidden by the API.
OUT-EXTENSION	Expose	Same for first name.
OUT-ZIP-CODE	Expose	Same for first name.
OUT-SEGNO	Hide	This field is internal to the IMS transaction and there is not meaningful to expose this to the API or to end-user clients.

The IMS Explorer service mapping for *output* would look like this:



The step-by-step for creating this service is provided under "Create a more generic service using IMS Explorer" on page 72.

Phone Book API design

With the "generic" service in place, we can compose the API that operates on top the service. First, we start by asking the question: what "resources" are acted upon by this transaction? The answer is it has only one resource: "contact records," which are rows in the database.

Note: Contrast this with the CICS Catalog Manager sample application. There we had *two* "resources": *items* (things for sale in the catalog), and *orders* (created when a customer placed an order).

We know this transaction provides four actions we can process against a resource (contact record): we can **add** a contact, we can **update** a contact, we can **display** a contact, and we can **delete** a contact:

Action	*IN-COMMAND value*	*RESTful HTTP Verb*
Add a contact	**ADD**	**POST**
Update a contact	**UPDATE**	**PUT**
Display a contact	**DISPLAY**	**GET**
Delete a contact	**DELETE**	**DELETE**

The next step is to plan the URI paths, the use of path or query parameters, and input JSON requirements:

Action	*Verb*	*Path*[63] *and notes*
Add a contact	**POST**	**/phoneBook/contacts** + JSON body The HTTP verb `POST` implies the action ("add"), so in the API mapping we will assign the value `ADD` to the `IN-COMMAND` field. The JSON will carry in the four contact record values: last name, first name, extension and zip code.
Update a contact	**PUT**	**/phoneBook/contacts/{lastName}** + JSON body The HTTP verb `PUT` implies the action ("update"), so in the API mapping we will assign the value `UPDATE` to the `IN-COMMAND` field. The resource to act upon is specified as a path parameter, which will get mapped to the LAST-NAME field. The JSON will carry in the three other contact record values: first name, extension and zip code[64].
Display a contact	**GET**	**/phoneBook/contacts/{lastName}** The HTTP verb `GET` implies the action ("retrieve" or "display"), so in the API mapping we will assign the value `DISPLAY` to the `IN-COMMAND` field. The transaction supports the display of a single contact record, not a range. Therefore, we use a path parameter. The API will map this path parameter value to the last name field. **Note:** *if* the transaction supported a range of contact records for display, then we may have considered using a query parameter string such as: `/phoneBook/contact?startName=<value>&numRecs=<value>` But it supports only a single contact, so we used a path parameter.

63 The value `/phoneBook` is an arbitrary value that is assigned when the API is first being created in the API Editor. This is known as the "base path." The value `/contact` is another arbitrary value that relates to the resource being acted up. This is the "API path."

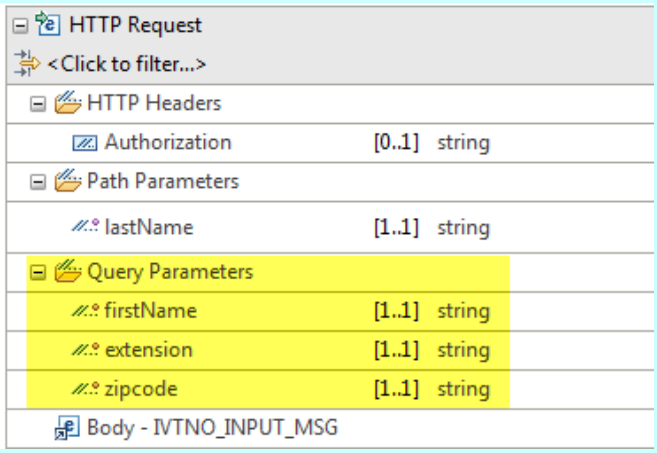
64 The Phone Book will assume a blank or null update field implies clearing the field. So if you want to update just the extension number for a contact, you supply the other three *unchanged* values along with the *udpated* value for the extension.

Delete a contact	**DELETE**	**/phoneBook/contacts/{lastName}** The HTTP verb DELETE implies the action ("delete"), so in the API mapping we will assign the value DELETE to the IN-COMMAND field. The transaction supports the delete of a single contact record, not a range. Therefore, we use a path parameter. The API will map this path parameter value to the last name field.

Note: REST API design patterns may vary. For example, this application has a limited number of user input values (just four), so you *could* pass those values in as a combination of a path parameter and query parameters:

PUT /phoneBook/contacts/{lastName}?firstName&extension&zipcode

In the API editor, the path parameter and query parameters would be available for mapping:



In that case the REST client would *not* need to construct a JSON body. The URI would be:

/phoneBook/contacts/Smith?firstname=John&extension=2222&zipcode=12345

It's a style choice. We show this example to let you know passing query parameters is possible, and that z/OS Connect EE V2.0 provides the flexibility to designing the API according to your preferred pattern.

What about the output plan for each action? Again, this depends on the action:

Action	*Fields returned to the end-user REST client*
Add a contact	OUT-MESSAGE – to indicate success or failure of the action. LAST-NAME, FIRST-NAME, EXTENSION, ZIP-CODE – this provides validation of the information that supplied on the input. We will hide the OUT-COMMAND value. We could have exposed it to show what command was entered, but the OUT-MESSAGE will provide this information.
Update a contact	The same as "add a contact."
Display a contact	The same as "add a contact."
Delete a contact	OUT-MESSAGE – to indicate success or failure of the action. LAST-NAME – to provide validation of the contact deleted. We will hide OUT-COMMAND and the first name, extension and zip code information. The key information supplied back to the REST client will be the message indicating success or failure and the last name of the contact record.

The step-by-step instructions for composing this API are provided under "Compose API using z/OS Connect EE V2.0 API Editor" on page 75.

Summary: z/OS Connect EE V2.0 allows us to simplify the service layer for the Phone Book to just one service. We can then use well-designed RESTful APIs on top the service to use the four functions of the transaction.

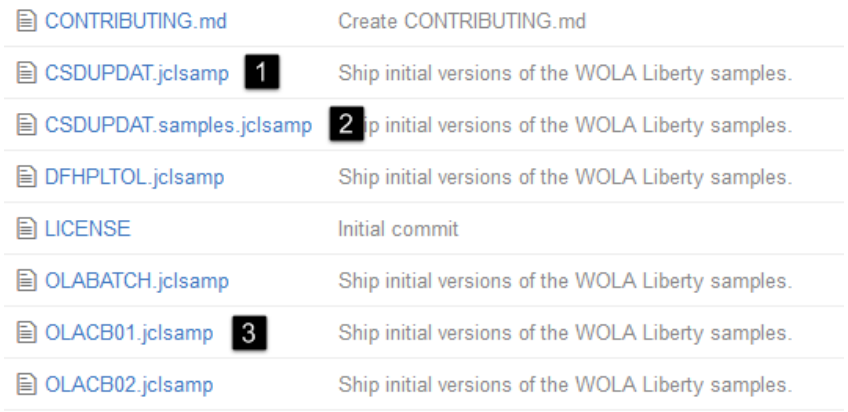
If we need to modify the API – for example, we get a requirement to show the first name, extension and zip on a delete action – we can update the API and redeploy the AAR file without having to change a thing at the service level.

OLACB01 sample

The OLACB01 sample comes with the WOLA samples and provides a very simple echo program for validating access to CICS using WOLA.

The sample (as well as a sample CSD update for all the WOLA samples) can be found here:

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



Notes:

1. The CSD update sample used earlier for updating the CICS region to support WOLA.
2. The CSD update sample to include the WOLA samples in the CSD for the CICS region.
3. The COBOL source for the OLACB01 sample program

If you want to use OLACB01 in your CICS region, then:

- ☐ Download the sample CSD update, customize and submit.
- ☐ Download the OLACB01 sample, customize the compile options and submit. The program itself requires no customization.
- ☐ Make sure whatever module library you compiled this into is on the DFHRPL DD for the CICS region.

Here a brief tour of the sample program:

```
 01  DFHCOMMAREA.
     05  MESSAGE-DATA  PIC X(80).     ← A single 80-character request/response field
*
 PROCEDURE DIVISION.
*
     IF EIBCALEN = 0      ← You must send some data; zero-length results in ECB1 abend
     THEN
       EXEC CICS
           ABEND
           ABCODE('ECB1')
       END-EXEC
     END-IF.

     DISPLAY MESSAGE-DATA.   ← It echoes back whatever you send.
```

```
    EXEC CICS RETURN END-EXEC.

    GOBACK.
```

WOLA and CICS TS 5.3 or higher

APAR PI56615 provides an update that allows the WOLA Task Related User Exit (TRUE) code to operate in a CICS TS 5.3 region[65]. Without this fix in place you experience a generic BBOX abend in the BBOATRUE program that operates in the CICS region. Specifically, you will see something like this:

```
DFHDU0203I 11/19/2015 10:02:23 CICWKS68 A transaction dump was taken for
dumpcode: BBOX, Dumpid: 1/0001, Tranid: BBOC, Tranum: 00000045, Program:
BBOATRUE.
DFHAC2236 11/19/2015 10:02:23 CICWKS68 Transaction BBOC abend BBOX in program
BBOATRUE term 0075. Updates to local recoverable resources will be backed
out.
```

After the fix is applied, do the following four things:

1. Recopy the WOLA modules (see page 23) and insure the updated modules are what is concatenated to the CICS region DFHRPL.
2. Update your Angel process so it is using the updated level of Liberty z/OS.

   Note: This is not *strictly* required, but it is encouraged.

   If your Angel process is operating from an install of Liberty z/OS *different* from that provided with z/OS Connect EE V2.0 (for example, your Angel process is associated with z/OSMF 2.1), then update the Angel's Liberty install to fixpack 8.5.5.9 or higher. This implies a stop and restart of the Angel process, which implies a stop and restart of all the Liberty z/OS server instances on the LPAR.
3. Restart your z/OS Connect EE V2.0 server and verify the new level by inspecting the messages.log file for this message:

   ```
   BAQR0000I: z/OS Connect Version 2.0.0.1
   ```
4. Restart your CICS region and the WOLA TRUE and WOLA Link Server task.

WOLA and long running task

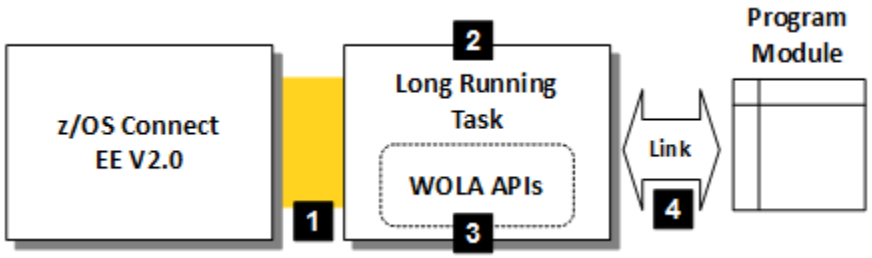
This document is focused primarily on z/OS Connect EE V2.0 and CICS or IMS. But those are not the only backend systems that can be used with z/OS Connect EE V2.0. Another is using WOLA to connect to a long-running task that use the WOLA APIs to "host a service." The common use-case for that model is to link to COBOL modules and provide them as a service through z/OS Connect EE V2.0.

Note: See: http://www-03.ibm.com/software/products/en/zos-connect-enterprise-edition

That page has a link to the announcement letter for z/OS Connect EE V2.0. The announcement letter has a section titled "Statement of general direction." There you will find information on z/OS Connect EE V2.0 and DB2 and MQ.

The following picture provides a high-level illustration of the z/OS Connect EE V2.0 using WOLA to connect to a long-running task:

65 APAR PI56615 provides several other updates as well. See: http://www-01.ibm.com/support/docview.wss?crawler=1&uid=swg1PI56615



Notes:

1. WOLA registration between the z/OS Connect EE V2.0 Liberty server instance and the address space for the long running task.
2. A task started and running a program (COBOL, C/C++, PL/I or High-Level Assembler) that use the WOLA APIs to "host a service."
3. The "host a service" APIs are `BBOA1SRV` (or the more advanced `BBOA1RCA`, `BBOA1RCS`). In summary, those APIs can be made to hold program control until a message is received from z/OS Connect EE V2.0.
4. From the program you can link to another program module and get the results from that module. That may then form the return to z/OS Connect EE V2.0, which then converts to JSON and returns to the requesting client.

If you're interested in the WOLA APIs that can accomplish this, see:

`http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490`

Locate the section for "Native API Primer":

Native API Primer
Ready to start coding to the native APIs? This primer offers a guided walkthrough of the APIs, from simple to increasingly sophisticated.

WP101490 - The WOLA Native APIs ... a COBOL Primer.pdf

WP101490 Primer.zip

That document provides a comprehensive review of the WOLA APIs and how they are used. Specifically, focus on the "outbound" sample `EXER3B`. That illustrates the use of `BBOA1SRV`.

Beyond the simple server.xml security elements

Encryption and Authentication in SAF

Earlier we illustrated the use of a "basic" security model, with the userid and password registry coded in the `server.xml`, and the transport layer security (encryption) key and trust store the default generated by Liberty itself. It was easy to set up, it worked, and it allowed us to get to the composing of APIs and deployment of APIs, which was the core objective of the document.

In reality, however, you will likely wish to make use of SAF[66] for elements of your security design. This is possible. It is based on the security functions Liberty z/OS provides, and in that sense it uses the same approach used by z/OS Connect Version 1.0.

If you go to this URL:

`http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102439`

and locate the "Quick Start Guide":

66 LDAP is also possible.

Quick Start Guide and Supporting Files

The following document provides a step-by-step set of instructions for installing, configuring and validating the operations of z/OS Connect:

WP102439 - WAS zOS Connect Quick Start Guide.pdf

The following ZIP file contains files referenced in the Quick Start Guide:

WP102439 Files.zip

you will find that "Section 4" provides information on using SAF for the registry and the key and trust store for encryption:

The instructions there will guide you through removing the "basic" security definitions and making use of SAF.

Client certificates

For this we will point you again to the z/OS Connect Version 1.0 Quick Start guide at the WP102439 Techdoc location. In "Section 4" of that guide you find instructions on using client certificates to authenticate:

Turning off SSL and Authentication

By default, z/OS Connect EE V2.0 will require both transport security (commonly referred to as "SSL," but more precisely called "TLS," or Transport Layer Security) and authentication.

Earlier in this document we saw that requirement surface: the instructions had you accept the security challenge caused by the self-signed server certificate, and then supply the userid and password.

But you may have certain services or APIs on which you do not wish to enforce transport security or authentication. z/OS Connect EE V2.0 provides a way to turn off either or both.

No Security? Yes, there are use-cases where transport security or authentication is not needed:

When z/OS Connect EE V2.0 is inside the network secure zone, behind firewalls and authentication devices. In that case you may decide the overhead of transport encryption is not needed. And you may decide that authentication at a mid-tier device is sufficient and z/OS Connect EE V2.0 can trust the traffic that flows back from there.

The service being exposed is of such low importance that encryption or authentication is not required. An example of this is a service that provides the day's menu in the office cafeteria.

If you deem encryption and/or authentication unnecessary, you can turn it off at the API or service level.

Turning off at the API level

In the CICS section of this document we illustrated the deployment of the catalog API, with the API defined in the `server.xml` with this:

```
<zosconnect_zosConnectAPIs location="">
    <zosConnectAPI name="catalog" />
</zosconnect_zosConnectAPIs>
```

In that case the API would require, *by default*, both transport security and authentication.

You could turn both off with:

```
<zosconnect_zosConnectAPIs location="">
    <zosConnectAPI name="catalog"
     requireAuth="false"
     requireSecure="false" />
</zosconnect_zosConnectAPIs>
```

Where `requireAuth` controls authentication, and `requireSecure` controls transport layer encryption. Coding `"false"` turns off the requirement for the API.

Clients may then access this API without authenticating and without going through the handshake protocol to establish encryption. This is true even if the underlying service definition still requires both authentication and encryption.

Turning off at the service level

Alternatively, you can turn off the authentication and transport security at the service:

```
<zosconnect_zosConnectService id="inquireSingleService"
        requireAuth="false"
        requireSecure="false"
        invokeURI="/inquireSingle"
        serviceName="inquireSingle"
        dataXformRef="xformJSON2Byte"
        serviceDescription="Inquire on an item in the catalog"
        serviceRef="wolaCICS" />
```

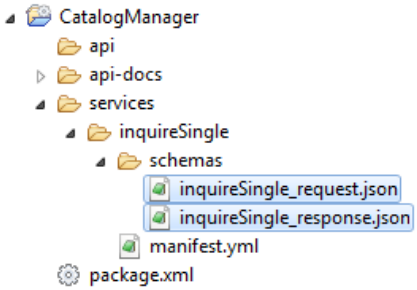
Any API at the higher-level that uses this service inherits this definition. For example, the catalog API we illustrated earlier makes use of the inquireSingle service. If you code authentication and transport security *off* for the service, the API no longer requires either as well.

Workaround to the problem of EBCDIC JSON schema in SAR files

With the initial release of z/OS Connect EE V2.0, a problem existed where the SAR files generated by the `BAQLS2JS` utility contained JSON schema files in EBCDIC. The Eclipse-based API editor requires those schema files in ASCII.

APAR PI53740[67] fixes the problem. But if you do not yet have that APAR on your system, you can work around this problem by deleting the schema files from the project, then importing ASCII copies of those files. Do the following:

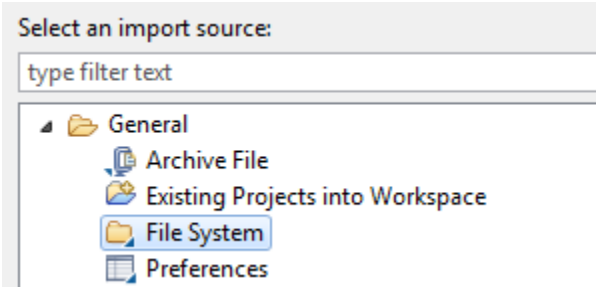
- ☐ On your z/OS system, locate the JSON schema files generated by the `baqls2js` utility. These will be in the USS file system location you specified in the parameters for that job.
- ☐ Downloaded those JSON files *in ASCII* on your workstation.
- ☐ Select the two schema files for the service:



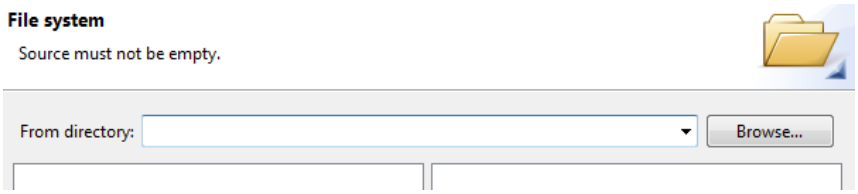
Then right-click and select "Delete."

67 http://www-01.ibm.com/support/docview.wss?crawler=1&uid=isg1PI53740

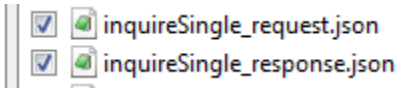
☐ Right click on "schemas" and select "Import," then select "File System":



☐ Use the browse button to navigate to the folder where the ASCII schema files reside:

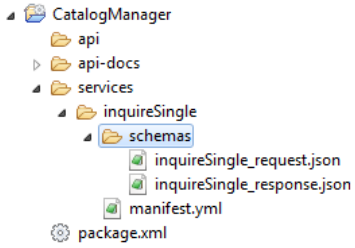


Then select the files you want to import:

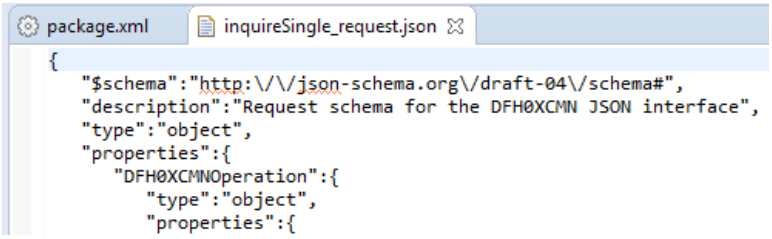


Then click "Finish."

☐ The files should appear in your Project Explorer view:



☐ Right click on one of the files and select *Open With → Text Editor*. If you see something like this:



Then you have worked around the problem. You may proceed with the API composition.

Common Problems … Symptoms and Causes

In this section we will provide a catalog of common problems we have seen and provide information to identify the problem and correct.

Note: This list is *not* exhaustive. We will add things to this as we come across them.

WOLA-related

"Angel process not compatible with local communication service"

If you see this error:

```
CWWKB0307E: The angel process on this system is not compatible with the
local communication service. The current angel version is 2, but the
required angel version is 3
```

It is because the Angel process started task in use by your Liberty z/OS server is running at a code level below what's required for z/OS Connect EE V2.0.

This issue can arise when you have an existing Angel process – for example, used by z/OSMF with z/OS 2.1 or higher – and you intend to re-use that Angel for your z/OS Connect EE V2.0 instances. However, if that existing Angel is operating at a lower code level, you will see the message illustrated above.

The corective action is to stop the Angel, update the Angel's JCL start procedure and point the `SET ROOT` variable to the installation path for z/OS Connect EE V2.0 (or any Liberty z/OS installation at 8.5.5.8 or above), and restart the Angel:

```
//BBGZANGL PROC PARMS='',COLD=Y
//*--------------------------------------------------------------------
// SET ROOT='<path to Liberty installation>'
//*--------------------------------------------------------------------
//* Start the Liberty angel process
//*--------------------------------------------------------------------
//STEP1   EXEC PGM=BPXBATA2,REGION=0M,
//  PARM='PGM &ROOT./lib/native/zos/s390x/bbgzangl COLD=&COLD &PARMS'
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
```

Note: Stopping the Angel on an LPAR implies all Liberty z/OS instances on the LPAR must come down. Schedule this update during a maintenance window.

Abend S138 - WOLA three-part name not unique on the system

To use WOLA, the `server.xml` must include the "three-part name" used when external address spaces WOLA-register into the Liberty z/OS server.

When the Liberty z/OS server starts, that three-part name is checked against a list of other three-part names in use on the system. (The list is maintained by the Angel process.)

The three-part name must be unique on the LPAR. If it is not, the server will not start and you will experience an S138 abend:

```
CEE3250C The system or user abend S138  R=02340404 was issued.

From entry point ntv_advertiseWolaServer at compile unit offset
+0000000020DF12E6 at entry offset +0000000000039D76 at address
0000000020DF12E6.
```

The `messages.log` file has very little other information about this error, other than the *lack* of the following message:

```
CWWKB0501I: The WebSphere Optimized Local Adapter channel registered with
the Liberty profile server using the following name: <three-part name>
```

The corrective action is to use a three-part name that is *unique* on the LPAR. Unfortunately, there is no easy way to check for what three-part names are currently in use. You have to know what values are coded in the `server.xml` files that are part of started Liberty instances.

Error: Could not open WOLA message catalog wola_cics_messages.cat

This message will appear in the `BBOMSG` output of a CICS region when a WOLA transaction is invoked and WOLA runtime support is not able to locate the WOLA CICS runtime message catalog file, e.g. `wola_cics_messages.cat`. This message can be eliminated by identifying the full path and message catalog file name in environment variable `NLSPATH`[68].

Language Environment (LE) provides a way to add environment variable `NLSPATH` to any WOLA enabled CICS by using a customized LE options module (`CEEROPT`). A WOLA version of `CEEROPT` is created by using the `ENVVAR` parameter of the `CEEXOPT` macro to add a `NLSPATH` environment variable for WOLA. Just assemble the macro and place the linked module into a data set in the CICS region's `DFHRPL` concatenation list. For more information on this topic see manual *Language Environment Customization, SA38-0685*.

Below is an example of the `CEEXOPT` macro setting environment variable `NLSPATH` to the WOLA CICS runtime message catalog:

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7..
        CEEXOPT ENVAR=(('NLSPATH=/usr/lpp/IBM/zosconnect/v2r0/wlp/lib/nX
              ative/zos/s390x/nls/%N.cat'),OVR)
```

Note: Standard Assembler Language continuation syntax rules apply. A character indicating the line is continued on the next line must appear in column 72. The directory path name must extend up to and include column 71 and the next character in the directory path name must start in column 16.

Below is a sample of the JCL required to assemble `CEEXOPT` and link edit module `CEEROPT`:

```
//STEP1 EXEC PGM=ASMA90,PARM='DECK,NOOBJECT'
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&TEMPOBJ(CEEROPT),DISP=(,PASS),UNIT=SYSDA,
// SPACE=(TRK,(1,1,1)),DCB=(BLKSIZE=3120,LRECL=80,DSORG=PO)
//SYSLIB DD DISP=SHR,DSN=SYS1.SCEEMAC
//       DD DISP=SHR,DSN=SYS1.MACLIB
//SYSIN  DD *
CEEROPT  CSECT
CEEROPT  AMODE ANY
CEEROPT  RMODE ANY
         CEEXOPT ENVAR=(('NLSPATH=/usr/lpp/IBM/zosconnect/v2r0/wlp/lib/nX
               ative/zos/s390x/nls/%N.cat'),OVR)
         END
//STEP2 EXEC PGM=IEWL,
// PARM='NCAL,RENT,LIST,XREF,LET,MAP,SIZE=(9999K,96K)'
//SYSPRINT DD SYSOUT=*
//SYSUT1  DD UNIT=SYSDA,SPACE=(TRK,(5,5))
//SYSLMOD DD DISP=SHR,DSN=USER1.ZCONN2.WOLA.LOADLIB
//SYSLIB  DD DSN=&&TEMPOBJ,DISP=(OLD,PASS)
//SYSLIN  DD *
```

68 Thanks to Mitch Johnson of the North America Advanced Technical Skills organization for this information.

```
INCLUDE SYSLIB(CEEROPT)
ENTRY CEEROPT
ORDER CEEROPT
NAME CEEROPT(R)
```

Document Change History

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

January 13, 2016	Original document at time of Techdoc creation
January 21, 2016	Added a note about CICS TS 5.3 and WOLA. In short: to use WOLA and CICS TS 5.3 requires, at this time, an iFix. CICS TS 5.2 and below works without an iFix.
February 3, 2016	Added commentary to the request and response mapping illustration to explain why the actions are being taken and the expected results of those actions.
February 16, 2016	Added the section for IMS.
March 9, 2016	Updated to indicate z/OS 1.13 is also a supported level of z/OS, along with z/OS 2.1 Updated to highlight APAR PI56615, which (among other things) provides a way to run WOLA in a CICS TS 5.3 region.
April 1, 2016	Updated to more accurately reflect the BAQSTRT JCL start proc for the server.
May 28, 2016	Updated to include a "Recommended Maintenance" table in the Overview section. Updated to illustrate message "*CWWKB0307E: The angel process on this system is not compatible with the local communication service. The current angel version is 2, but the required angel version is 3*," which means the use of an existing Angel process will require that Angel to be updated by pointing its JCL start procedure to the z/OS Connect EE V2.0 installation location.
July 22, 2016	Added section "Common Problems … Symptoms and Causes" to catalog error symptoms and corrective actions commonly seen during initial setup. Updated the IMS section to more accurately reflect where to get the data structure definition for the "Phone Book" application, and to correct the transaction code for the sample program (the document had IVTNOM before, now it is IVTNO).

End of WP102604