



WebSphere Application Server for z/OS Version 8.5

# Java Batch Runtime Quick Start Guide

**A step-by-step guide to setting up and using Java Batch**

*Version Date:* June 1, 2013

See "Document Change History" on page 44 for a description  
of the changes in this version of the document

IBM Advanced Technical Skills  
**Gaithersburg, MD**

**WP101783** at

[ibm.com/support/techdocs](http://ibm.com/support/techdocs)

© IBM Corporation 2013

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

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

## Table of Contents

<b>Overview .....</b>	<b>4</b>
Focus on z/OS.....	4
Just WAS z/OS V8.5? Or Compute Grid in WAS V7 or V8 as well?.....	4
WAS V8.5 Installation Manager install options.....	4
What about developing Java batch applications?.....	4
<b>Getting Started - Initial Enablement and Validation.....</b>	<b>5</b>
Overview of the process.....	5
Assumed starting WAS z/OS configuration.....	5
Create the LRS and LREE database tables in DB2.....	5
Create the JDBC provider and data sources.....	6
Create the EJBROLE definitions to secure the Job Management Console (JMC).....	12
Turn on application security and validate secure access to JMC.....	13
Configure the Job Dispatching function in a server.....	13
Deploy sample application into a server and validate endpoint configuration.....	15
Prepare xJCL, submit, and validate successful execution.....	17
Storing xJCL in the Job Repository and submitting from there.....	20
Running the XDCGIVT sample.....	21
XDCGIVT with DB2.....	23
<b>Enabling the WSGRID Interfaces.....</b>	<b>25</b>
Conceptual overview.....	25
On z/OS, two approaches.....	26
Product InfoCenter articles on each approach.....	26
WSGRID and MQ.....	26
Overview.....	26
Create queues in MQ.....	27
Install MDB interface into dispatcher server.....	27
Create the WSGRID load module.....	30
Configure JCL and validate.....	31
Summary of native WSGRID on z/OS.....	32
WSGRID and Service Integration Bus.....	33
Overview.....	33
Run wsgridConfig.py script to configure default messaging.....	33
Run WSGrid.sh shell script to submit job.....	36
Summary of native WSGRID and default messaging.....	37
<b>When Compute Grid 8 on WAS V7 or V8 is the Environment.....</b>	<b>38</b>
Configuration similar ... function identical.....	38
Creating the runtime with CG is different between V7 and V8.....	38
Configuring the Compute Grid Function.....	39
Create the LRS and LREE database tables in DB2.....	39
Create the JDBC provider and data sources.....	39
Create the EJBROLE definitions to secure the Job Management Console (JMC).....	39
Turn on application security and validate secure access to JMC.....	39
Configure the Job Dispatching function in a server.....	39
Deploy sample application into a server and validate endpoint configuration.....	39
Prepare xJCL, submit, and validate successful execution.....	40
Storing xJCL in the Job Repository and submitting from there.....	40
Running the XDCGIVT sample.....	40
Configuring the WSGRID interfaces.....	40
<b>Miscellaneous Information.....</b>	<b>41</b>
How to create and customize a WAS z/OS runtime environment.....	41
wsgridConfig.py command example.....	42
Example of IM used to install WAS 8 and CG 8 into same install image.....	42
<b>Documentation References.....</b>	<b>43</b>
<b>Document Change History .....</b>	<b>44</b>

## Overview

"WebSphere Java Batch" (often referred to as "Compute Grid") is a software function that allows batch programs to run inside the Java runtime environment of WebSphere Application Server<sup>1</sup>. It is available on all operating system platforms supported by WebSphere Application Server itself.

Prior to WAS V8.5 it was offered as a separately licensed product called "Compute Grid". To use the function the product was "augmented" (added) to an existing V7 or V8 runtime configuration. With V8.5 no augmentation is needed; the function is simply part of the application server foundation.

**Note:** If you're wondering which is better -- the separate Compute Grid product or what is included with WAS V8.5 -- the answer is they are *both the same*. The function is identical between Compute Grid V8 and what is included with WAS V8.5<sup>2</sup>.

Saying this function provides batch processing services is one thing; showing it in action is another.

**Quick Start** This document is titled a "Quick Start Guide" because it is designed to help you get the function up and running as quickly as possible with the least amount of struggle. This document will guide you, step by step, through initial setup, validation and finally using the batch function.

### Focus on z/OS

We mentioned earlier WebSphere Java Batch was available on all operating systems supported by WAS itself. However, *this* document will focus on WAS z/OS. Some of the setup and usage concepts are common across the platforms, but WAS z/OS is just different enough in some key areas (such as DB2 setup, or security configuration) to warrant a little extra focus.

### Just WAS z/OS V8.5? Or Compute Grid in WAS V7 or V8 as well?

The main focus of this document will be WAS z/OS V8.5<sup>3</sup>. The setup and usage steps are very similar between V8.5 and Compute Grid V8 on WAS V7 or WAS V8. However, they are not identical. To assist with these differences, see "When Compute Grid 8 on WAS V7 or V8 is the Environment" starting on page 38.

### WAS V8.5 Installation Manager install options

WAS z/OS requires the use of IBM Installation Manager (IM) to do the install of the product code<sup>4</sup>. IM takes as input install options that control what gets installed. The core feature of WAS z/OS is installed with the option `core.feature`. But we recommend you also specify `ejbdeploy`, `liberty`, `thinclient`, `embeddablecontainer`, and `samples` as well.

The option `ejbdeploy` is particularly important as one of the sample batch applications requires that option be installed to do the application deployment.

### What about developing Java batch applications?

Eventually you will start developing your own batch applications that deploy and run using the WebSphere Java Batch function. The supplied samples are useful for validation and to show key concepts, but you don't run your business on sample programs.

This document will not go into that topic. Instead, we will point you to a few good starter references for developing WebSphere Java Batch programs. See page 43.

Our mission here is to get your runtime operational and validated as quickly and easily as possible. With that accomplished then you will have a good environment for your own Java batch programs to be run.

- 1 The WP101783 Techdoc has overview material on what WebSphere Java Batch (also known as "Compute Grid") does and how it operates. This document will focus on getting it up and running.
- 2 If you're wondering why anyone would use the separate Compute Grid product when the function is included with WAS V8.5 ... the answer to that question has to do with what level of WAS you are at presently, and whether you are interested in moving to V8.5 to get the batch function. Some have just recently moved to WAS V7 or WAS V8 do not wish to introduce a new level of WAS into their environment just yet. For them the answer is to use Compute Grid and augment the function onto their existing V7 or V8 runtimes.
- 3 Version 8.5, Fixpack 2 to be precise.
- 4 See <http://www.ibm.com/support/techdocs/atsmasr.nsf/WebIndex/WP102014>

## Getting Started - Initial Enablement and Validation

The objective of this section will be to enable the batch runtime environment, deploy one of the sample batch applications and have it run successfully.

### Overview of the process

In a simple bullet list the process looks like this:

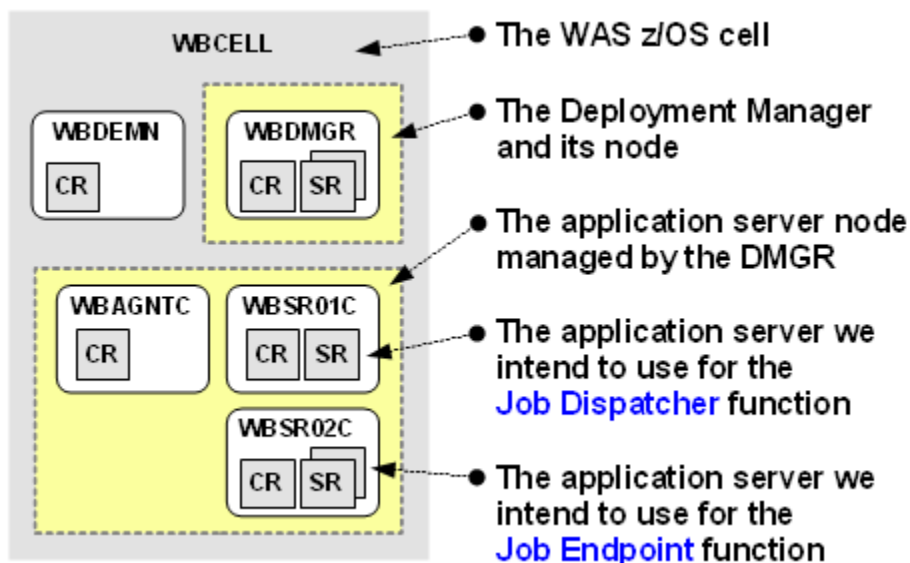
- Create the database tables
- Create the DB2 JDBC provider and data source definitions
- Create the EJBROLE definitions for securing the Job Management Console
- Turn on application security so the Job Manage Console will be secured
- Configure the Job Dispatching function in a server and validate access
- Deploy a sample batch application into a server and validate the endpoint configuration
- Submit the job and watch it run to completion

There's detail behind all that of course. That's what this document will provide.

### Assumed starting WAS z/OS configuration

This document will *not* go through all the steps to create and customize a WAS z/OS runtime<sup>5</sup>.

The runtime topology used for the development of this document was the following<sup>6</sup>:



In summary, a relatively simple Network Deployment (ND) configuration on one LPAR with two servers. We felt this was the proper level for illustration: not too simple; not too complex.

**Note:** The WebSphere Java Batch function can work in a configuration as simple as a single Standalone server, or something as sophisticated as a multi-LPAR cluster environment built on top a Parallel Sysplex. What we will show in this document applies to those as well<sup>7</sup>.

### Create the LRS and LREE database tables in DB2

Underlying the WAS z/OS V8.5 batch runtime environment is a set of relational tables used to control the batch processing operations. Using z/OS DB2 is not a requirement. You could use a file-based relational system such as Derby, or you could use Oracle<sup>®</sup><sup>8</sup>. For this document we

<sup>5</sup> We provide an overview of the process along with some pointers to key documentation under "How to create and customize a WAS z/OS runtime environment" on page 41.

<sup>6</sup> Why "WB" for the cell prefix? Answer: "WebSphere Batch". ☺

<sup>7</sup> What changes are the deployment targets. If you have a clustered environment then rather than configuring the Job Dispatcher to run in a server, you configure it to run in the cluster. Similarly, rather than deploy a batch application to a server, you deploy it to the cluster.

<sup>8</sup> Oracle is a registered trademark of the Oracle Corporation

will illustrate the use of IBM DB2 for z/OS as the database engine.

**Overview:** WAS z/OS V8.5 comes with a file that contains the database and table definitions. What you do is customize a few things in the supplied definitions, then work with your database administrator to have them implemented.

Do the following:

- Locate the home for your application server node. Typically that's something like `/wasv85config/wbcell/wbnodec/AppServer`.
- Now drill down under that home and locate the `/util/Batch` directory. Note the file `SPFLRS` located there. That's the file that contains the DB2 z/OS definitions.
- Copy that file to a FB 80 data set. What follows is an example of the command to copy it to a PDS. The command may be issued from ISPF Option 6:

```
OGGET '/<path>/SPFLRS' 'hlq.dataset(SPFLRS)'
```

- Browse the contents of that file. The first several lines look like the following. Note how variables are established that permit global changes to customize the definitions:

```
-- Tailor these values for your DB2 environment.
--Variables ... Use global change on these variable names
-- LRSCHSG      Store group
-- LRSCHED      Database
-- <volume_name> Volume for the store group, or '*' for SMS-managed.
-- <vcat>       VSAM catalog
-- LRSSPACE     Tablespace
-- LRSSHEMA     Schema name
-- PUBLIC       Access ID

DROP STOGROUP LRSCHSG;
CREATE STOGROUP LRSCHSG VOLUMES(<volume_name>) VCAT <vcat>;
:
```

- Review the file with your DBA. Make the global changes to customize the values for the variables seen at the top of that file.
- Use either JCL or SPUFI to submit the updates and create the database and tables.

**Note:** You will likely find your overall job ends with a `RC=8` since some of the statements in that file will fail with `SQLCODE=-204`. That's because the file as supplied performs `DROP` operations on `STOGROUP` and `DATABASE` before those are defined. You can avoid that by commenting out those `DROP` lines on your *first* invocation of these definitions<sup>9</sup>.

- Review the output and make sure all the *other* DB2 SQL statements returned successful.
- Make note of the schema name you used. You'll need that later.

Your schema value:	
--------------------	--

### Create the JDBC provider and data sources

Both the Job Dispatcher function as well as the batch job endpoint function access the relational tables you just built. That means some JDBC definitions need to be in place. Even if you have existing JDBC definitions to the same DB2 subsystem instance, create new ones for WebSphere Java Batch to use. That will give you isolation between definitions.

Do the following:

- Log on to your cell's Administrative Console

<sup>9</sup> If you're looking to drop and recreate the tables to start fresh, then leave the `DROP` lines in effect.

- Go to *Resources* ⇒ *JDBC* ⇒ *Data Sources* and set the "Scope" to "All Scopes"

What data sources do you see listed? If you see `jdbc/lrsched` or `jdbc/lree` then they need to be deleted.

**Why?** If you see data sources of `jdbc/lrsched` or `jdbc/lree` it implies artifacts from a previous configuration are present.  
Our preference is you delete any existing `jdbc/lrsched` or `jdbc/lree` data sources (which are likely related to Derby) to avoid any confusion or ambiguity.

Delete `jdbc/lrsched` and `jdbc/lree` and data sources at this time<sup>10</sup>.

Save and synchronize the changes.

Next we will create a JDBC provider and data sources at the cell level. We do it at the cell level because the Deployment Manager also needs access to the database as well as the application servers.

The JDBC "type" we will create will be Type 4 (XA). Normally on z/OS we prefer Type 2 for the advantages it provides, but for the WebSphere Java Batch connections into DB2 Type 4 at the cell level provides the most flexibility. This is particularly true if the cell topology spans LPARs in a Parallel Sysplex and all the servers and nodes need access to the same JDBC definitions.

- In the Admin Console go to *Resources* ⇒ *JDBC* ⇒ *JDBC providers*.
- Set the scope to "Cell"
- Click on the "New" button to create a new provider definition.
- Set the Database type, Provider type and Implementation type as shown:

Scope

 ← **Note "cell" scope**

\* Database type

\* Provider type

\* Implementation type

\* Name

The name value may be whatever you wish provided it is unique at the scope you set. Click the "Next" button.

<sup>10</sup> Only if you are certain doing so won't harm others. If the cell you're working with is your own test cell then you can feel relatively safe. If the cell is a cell shared with others then exercise caution before deleting.

- ❑ Set the driver path and native library path<sup>11</sup> according to your DB2 installation values. Leave the "PureQuery" path empty.

Directory location for "db2jcc4.jar, db2jcc\_license\_cisuz.jar" which is  
 \${DB2\_JCC\_DRIVER\_PATH}

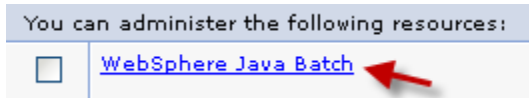
Directory location for "pdq.jar, pdqmgmt.jar" which is saved as Web  
 \${PUREQUERY\_PATH}

Native library path

Directory location which is saved as WebSphere variable \${DB2\_JCC

Click the "Next" button.

- ❑ At the summary panel, click the "Finish" button.
- ❑ Now click on the link that represents your new JDBC provider:



- ❑ Then over on the right, click the "Data sources" link:



- ❑ Click the "New" button.
- ❑ Provide the "Data source name" and "JNDI name" as lrsched and jdbc/lrsched as shown here:

JDBC provider name

\* Data source name

\* JNDI name

Click the "Next" button.

- ❑ Provide the specific properties for access to the DB2 system:

Name	Value
* Driver type	4
* Database name	WSCDBP0
* Server name	wsc3.washington.ibm.com
* Port number	446

Where:

- "Driver type" is 4 as shown
- "Database name" is the DB2 location name for your DB2 subsystem

<sup>11</sup> JDBC Type 4 (XA) should not need the native library path. We show it just for completeness.



- "Server name" is the host value on which your DB2 is listening
- "Port number" is the port on which your DB2 is listening

Click the "Next" button.

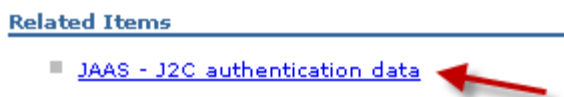
- The next panel is the "Security alias" panel. For now click the "Next" button. Later you'll come back and update the alias value.

**Why?** Your Type 4 connector will need an alias<sup>12</sup>. However, unless you had an alias created earlier, you would have found the drop-down list on that panel empty. There's no good way to create an alias at that point in the process without losing the updates made to that point. It's easier to complete the data source, create the alias separately, then come back into the data source and update it.

- On the "Summary" panel click "Finish."
- Save and synchronize the changes made to this point.
- Click on the link that represents your new data source:



- On the right side of the resulting page you'll find this:



Click on that link.

- Click the "New" button.
- Create your alias:

**General Properties**

\* Alias

\* User ID

\* Password

Description

Apply OK Reset Cancel

Where:

- "Alias" is any name you wish to use to identify this alias
- "User ID" is the ID you wish WAS z/OS to use when asserting into DB2

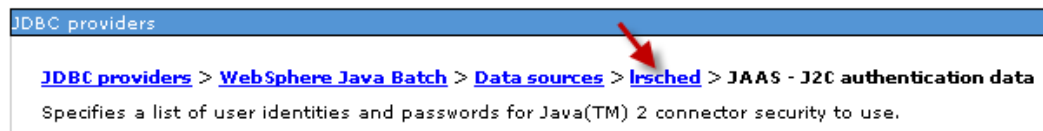
**Note:** Any valid RACF ID along with its password will work. This is possible because the tables you created earlier had, by default, GRANT values of PUBLIC. If your DBA changed the GRANT values to something *other* than PUBLIC, then work with your DBA to come up with an ID that will work with the GRANT values defined.

- "Password" is the RACF password associated with that ID

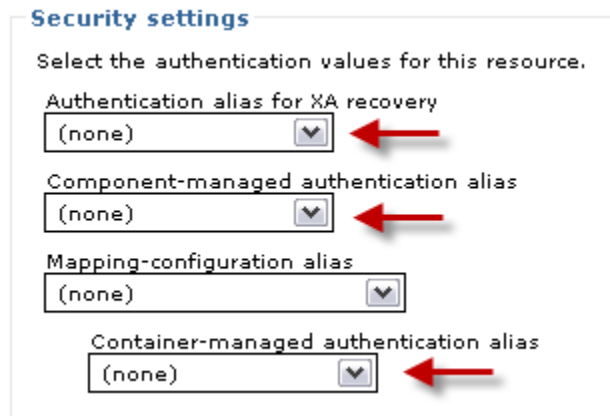
Then click on the "OK" button.

<sup>12</sup> Without an alias defined, an attempt to establish a connection with that data source would result in a Error Code=-4,461 message.

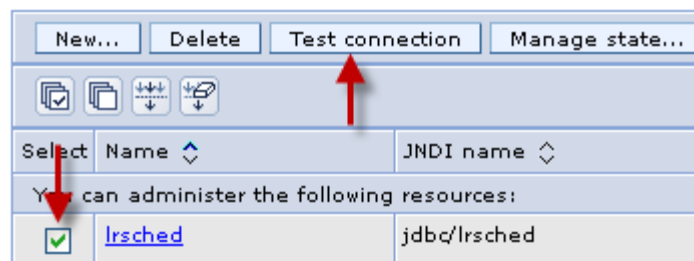
- At the top of the screen you should now see a navigation string that will allow you to go back to your data source definition. Click on the "Irsched" link:



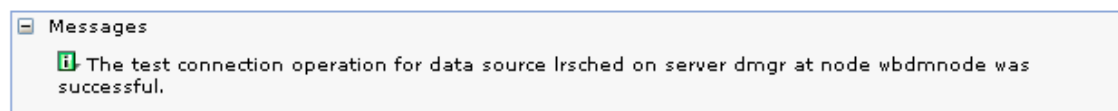
- On the page that comes up, scroll down and locate the "Security settings" section. Use the drop-down lists to assign your new alias for the fields shown here:



- Click on the "OK" button.
- Save and synchronize the changes made to this point.
- Test your connection by selecting your new data source and clicking on the "Test connection" button:



You should see something like this:



If you do *not* see that success message, then check the following as part of debugging the problem:

- Make sure the target DB2 is actually up and ready to receive connections.
- Check the spelling of the path to the DB2 driver path. At this point the value you provided will be in an environment variable. Go to *Environment* ⇒ *WebSphere variables*, set the scope to "Cell" and look for the DB2\_JCC\_DRIVER\_PATH variable. The value is your driver path. Check for spelling errors and problems with case mis-match. Correct if required.
- Check the host and port values you provided on the data source. Those must be what your DB2 "DIST" address space is configured to listen on.
- Check the alias you provided and make sure the ID and password as provided is correct.

- Now create a *second* data source with name `lree` and JNDI name of `jdbc/lree`. The process for creating this second data source is the same as for the first<sup>13</sup>.
- Save and synchronize the changes.
- Use the "Test connection" button to test this new data source.
- Click on the `lree` data source link to open up the general properties of the data source.
- On the right side of the panel, click on the "Custom properties" link:

Additional Properties

- [Connection pool properties](#)
- [WebSphere Application Server data source properties](#)
- [Custom properties](#) 

- Scroll down and locate the `currentSchema` custom property:

<input type="checkbox"/>	<a href="#">currentSchema</a>	Identifies the default schema name used to qualify unqualified database object references where applicable in dynamically prepared SQL statements. Unless <code>currentSchema</code> is used, the default schema name is the authorization id of the current session user.	false
--------------------------	-------------------------------	--	-------


It will have no value associated with it initially.

- Click on the `currentSchema` property link
- Update the "Value" field with the schema value you defined when you created the database tables. We had you capture this value back on page 6.

Name

Value

Then click the "OK" button.

- Now go to *Environment* ⇒ *WebSphere variables*.
- Set the "Scope" to "Cell."
- Locate the `GRID_ENDPOINT_DATASOURCE` variable. It's default value will be `jdbc/pgc`. Click on the link for that variable and change the value to `jdbc/lree` to match the data source created earlier. Then click "OK".
- Click the "New" button and create a new variable<sup>14</sup>:

Name:	GRID_ENDPOINT_BACKENDID
Value:	DB2UDBOS390_V9_1

- Click the "OK" button
- Save and synchronize all changes made to this point.
- Stop and restart your DMGR and servers to pick up those environment variable changes.

<sup>13</sup> You'll find the alias value you created earlier now available in the drop-down lists when creating this second data source.

<sup>14</sup> Even if your DB2 is at V10, set the value of this to what's shown.

## Create the EJBROLE definitions to secure the Job Management Console (JMC)

The Job Management Console<sup>15</sup> will be secured so only permitted IDs may access it. This involves creating a set of EJBROLE definitions and granting your WAS Admin ID access to the roles<sup>16</sup>.

Do the following:

- In the Admin Console, go to *Security* ⇒ *Global security*.
- On the lower right of the resulting panel, locate and click on the "Custom properties" link:

- [Security domains](#)
- [External authorization providers](#)
- [Programmatic session cookie configuration](#)
- [Custom properties](#) 
- [z/OS security options](#)

- Locate the `com.ibm.security.SAF.profilePrefix` property:

<input type="checkbox"/>	<a href="#">com.ibm.security.SAF.profilePrefix</a>	WBCELL
--------------------------	--	--------

Record the value (if a prefix is defined<sup>17</sup>) here:

Your SAF prefix value:	<input type="text"/>
------------------------	----------------------

- Work with your security administrator to get the following EJBROLE definitions created:

```
RDEFINE EJBROLE <SAF_prefix>.lradmin UACC(NONE)
RDEFINE EJBROLE <SAF_prefix>.lrsubmitter UACC(NONE)
RDEFINE EJBROLE <SAF_prefix>.lrmonitor UACC(NONE)
```

**Notes:** EJBROLE names are **case-sensitive**. Create them just as shown here.

The `lradmin` role has full authority to submit jobs, cancel or stop jobs, and purge jobs.

The `lrsubmitter` role has authority to submit jobs, but nothing else.

The `lrmonitor` role has authority to view jobs, but can take no action against them.

- Then grant the WAS Admin ID<sup>18</sup> READ to those profiles<sup>19</sup>:

```
PERMIT <SAF_prefix>.lradmin CLASS(EJBROLE) ID(<admin_id>) ACC(READ)
PERMIT <SAF_prefix>.lrsubmitter CLASS(EJBROLE) ID(<admin_id>) ACC(READ)
PERMIT <SAF_prefix>.lrmonitor CLASS(EJBROLE) ID(<admin_id>) ACC(READ)
```

- Finally, make sure to refresh the EJBROLE class:

```
SETROPTS RACLIST(EJBROLE) REFRESH
```

<sup>15</sup> This is the web interface to the "Scheduler" function, which in this document we refer to as the "Job Dispatcher." We call it the "Dispatcher" rather than the "Scheduler" to avoid confusion with enterprise scheduler products such as Tivoli Workload Scheduler, or similar products from other vendors. We show to integrate with these enterprise schedulers starting on page 25.

<sup>16</sup> Or an ID other than the WAS Admin ID. Here we'll show the WAS Admin ID simply because it's an ID you already have created and are familiar with.

<sup>17</sup> A SAF prefix is not required. It may be when your cell was created no SAF prefix was specified. If so that's okay. If one was specified, then you will use it with the EJBROLE definitions that you will create. If no SAF prefix is defined for the cell, check if the EJBROLE profiles shown here are already defined. If so, then you do not need to create the roles. Assigning the ID READ to the roles is still needed.

<sup>18</sup> Or whatever valid SAF ID you wish.

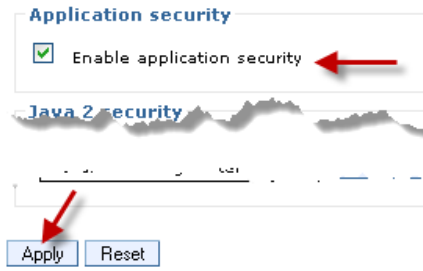
<sup>19</sup> Granting the same ID access to all three roles is not necessary. The `lradmin` role has full authority. We show all three to illustrate the commands for all three roles. Normally you would grant different IDs access to the different roles so those IDs would have different authorities within the JMC.

**Turn on application security and validate secure access to JMC**

For the EJBROLE definitions to have effect, application security must be enabled for the cell.

Do the following:

- In the Admin Console, go to *Security* ⇒ *Global security*.
- Click the checkbox for "Application Security," then click "Apply":



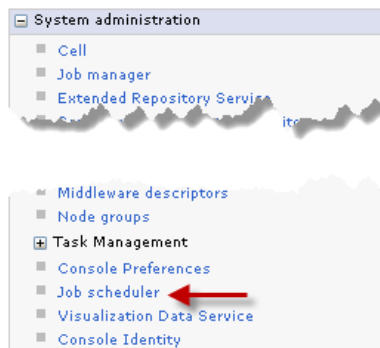
- Save and synchronize the changes.

**Configure the Job Dispatching function in a server**

This is known in the product documentation as the "Job scheduler." But as mentioned before, we prefer to call this the "Dispatcher" so there's no confusion about this function compared to the function served by enterprise scheduler products such as Tivoli Workload Scheduler.

Do the following:

- Go to *System administration* ⇒ *Job scheduler*:



- On the panel that you see, do the following:

**General Properties**

Scheduler hosted by

Database schema name

Data source JNDI name

Endpoint job log location

Record usage data in scheduler database

Record usage data in SMF (z/OS only)

- Set the "Scheduler hosted by" to the server you wish
- Make sure the "Database schema name" matches your tables (see page 6)
- Click the "OK" button.

- Save and synchronize all changes to this point.
- Stop the server<sup>20</sup> you selected to host the Scheduler (i.e., "Dispatcher") function.
- Start the server.
- When the server completes initialization, open a browser that's *different* from the WAS Admin Console browser you are using. For example, if the WAS Admin Console browser is IE, the open Firefox for the JMC.

**Why?** If you simply opened a new tab in the same browser, the cookie for the WAS Admin ID used to log onto the Admin Console would be detected and used and the Job Management Console would *not* prompt for a userid and password.

But we wish to show logging onto the JMC. You really have two choices:

- 1) Close the Admin Console browser and all windows of that browser type, or
- 2) Use a different browser

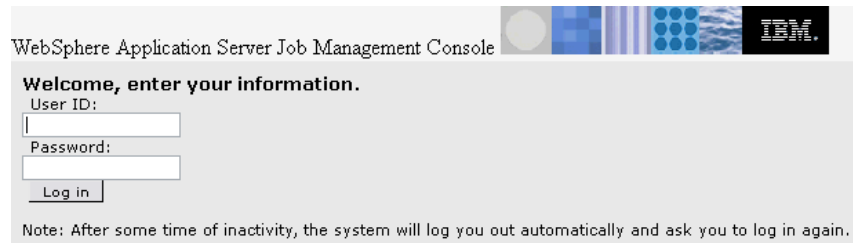
Opening a different browser is often the easiest.

- From that different browser, enter the URL:

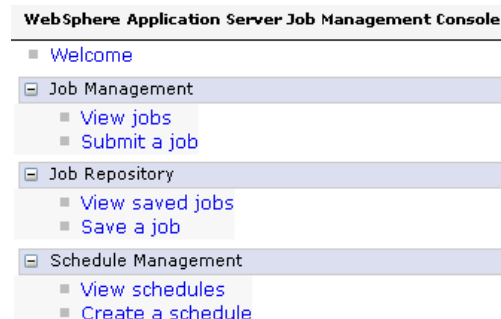
`http://<host>:<port>/jmc`

Where <host> is the TCP host for the server, and <port> is the normal HTTP port.

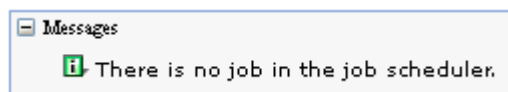
- If the browser issues a challenge because the site's certificate is unknown, accept the challenge and proceed to the logon panel.
- You should see something like this:



- Enter the ID and password for the ID granted READ to the EJBROLE definitions earlier.
- You should then see something like this:



- Click on the "View jobs" link under "Job Management".
- You should see no jobs listed, and this message:



Which makes sense ... you haven't submitted any yet. But this little test validated the ability of the Dispatcher function to query the tables and determine there were no jobs to list. That's a good sign.

<sup>20</sup> If currently running. The server did not need to be up for you to make the changes you made.

## Deploy sample application into a server and validate endpoint configuration

Deploying a batch application into a server tells the Dispatcher function that the server is a *batch endpoint*. In this section you will deploy the "SimpleCI" sample application. That application simply runs for the specified period of time, then ends. It has no input or output requirements. It is not very good for illustrating the function of WebSphere Java Batch, but it is very good for simple validation.

Do the following:

- Go to the following URL to begin the process of downloading the sample application:

<http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp>

- In the navigation tree on the left, locate "Samples, Version 8.5" and click on it:



- Click on the "Downloads" tab in the panel that appears in the right frame of the page:

### WebSphere Application Server Version 8.5 samples

View the latest WebSphere® Application (single server) Samples that apply to one or more of the following c/z/OS®. This information applies to Version 8.5 and to all subsequent releases and modifications until otherwi



- Scroll down and locate the "Java Batch" section of samples:



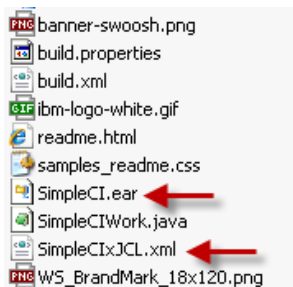
- Locate the "Java Batch SimpleCI sample" and use one of the two download methods (FTP or HTTP) to download the ZIP file.

#### Java Batch SimpleCI sample

Install, run, and monitor a simple application that performs compute-intensive calculations for a specified amount of time using the simpleCI sample.

Download sample: [FTP](#) [HTTP](#) | [More information](#)

- When downloaded, extract the contents of the ZIP file into a folder on your workstation. Several files are included, but only two are of interest to us right now:



SimpleCI.ear is what is deployed into the server, and SimpleCIxJCL.xml is the job declaration file that is used to submit the job and have it execute.

- In the Admin Console, go to *Environment* ⇒ *WebSphere variables*, and set the scope to "Cell." The verify that the following two variables are set properly:

<a href="#">GRID_ENDPOINT_BACKENDID</a>	DB2UDBOS390_V9_1
<a href="#">GRID_ENDPOINT_DATASOURCE</a>	jdbc/lree

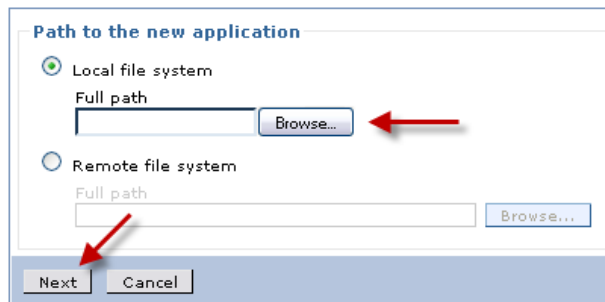
GRID\_ENDPOINT\_BACKENDID should be set to what's shown in the picture.

GRID\_ENDPOINT\_DATASOURCE should be set to your "lree" data source JNDI.

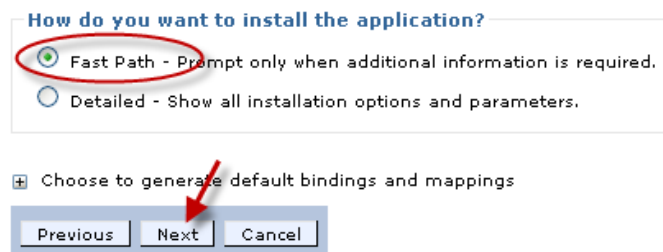
- In the Admin Console, go to *Applications* ⇒ *New applications*, then click on *New Enterprise Application*<sup>21</sup>.

**Important** The SimpleCI application you are about to install requires "EJBDeploy" to take place. That means the copy of WAS z/OS V8.5 must have been installed using Installation Manager with the `ejbdeploy` option specified along with `core.feature`, `liberty` and any other options you need.

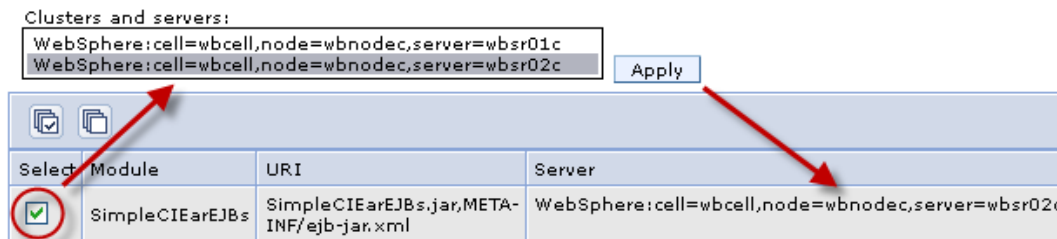
- Use the "Local file system" option to browse for and select the `SimpleCI.ear` file you unzipped from the downloaded file, then click "Next":



- Take the "fast path" and then click "Next":



- For "Step 1" just click "Next."
- For "Step 2" make sure the one application module gets mapped to the server you intend to use as the batch end point. Here we're illustrating how we mapped it to our second server, "wbsr02c":



Then click "Next."

- On the "Summary" page click "Finish."

<sup>21</sup> What we are about to illustrate is the standard application deploy process for WebSphere Application Server. It's a batch application, but the deployment process is no different from any other EAR file.




- The deploy will take a few moments as the "EJBDeploy" processing takes place. When that's done, click the "save" link:

Application SimpleCIEar installed successfully.

To start the application, first save changes to the master configuration.

Changes have been made to your local configuration. You can:

- [Save](#) directly to the master configuration. 
- [Review](#) changes before saving or discarding.

- Make sure the changes synchronize to the nodes.
- Now go to *System administration* ⇒ *Job scheduler*, and click on the link for "WebSphere grid endpoints":

**Additional Properties**

- [Classification rules](#)
- [Custom properties](#)
- [Job classes](#)
- [Security role to user/group mapping](#)
- [WebSphere grid endpoints](#) 

You should see the server you deployed SimpleCI into listed as a grid endpoint, and you should see the data source JNDI name as equal to what was set for the variable GRID\_ENDPOINT\_DATASOURCE<sup>22</sup>:

Name	Datasource JNDI name
<a href="#">WebSphere:cell=wbcell,node=wbnodec,server=wbsr02c</a>	jdbc/lree
Total 1	

- Start the application server you deployed SimpleCI into (it not already started); or if started, then go to *Applications* ⇒ *All Applications* and start the SimpleCI app.

Before you can run that sample application you need to prepare the SimpleCI "xJCL" (the job declaration file) for submission. That's next.

**Prepare xJCL, submit, and validate successful execution**

Earlier (page 15) we had you unzip the SimpleCI download file into a directory. One artifact was the `SimpleCI.ear` file, which you deployed in the previous section. Another artifact is a file called `SimpleCIxJCL.xml`, which is the xJCL used to run the job.

<sup>22</sup> This is why we had you change the default value of `jdbc/pgc` to your JNDI value, and why we had you verify that just before deploying the application.

Do the following:

- Use an editor to open the file. The contents will look like this:

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

<!--
...

<job name="SimpleCIEar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jndi-name>ejb/com/ibm/ws/ci/SimpleCIEarCIController</jndi-name>
  <substitution-props>
    <prop name="calctime" value="30" />
    <prop name="outfile" value="/path/simpleciout.txt" />
  </substitution-props>

  <!-- ### STEP 1 ### -->

  <job-step name="Step1">
    <classname>com.ibm.websphere.ci.samples.SimpleCIWork</classname>
    <props>
      <prop name="calculationTimeInSecs" value="${calctime}" />
      <prop name="outputFileName" value="${outfile}" />
    </props>
  </job-step>
</job>
```

**Note 1**

**Note 2**

**Note 3**

#### Notes:

1. This xJCL has two "substitution properties" defined -- one for the length of time the batch job runs, and the second is for the output path and file name. Here the values are being set to a preliminary default of 30 seconds and the path and file you see above. We are going to override those defaults at time of submission.
2. This job is very simple: one step, and the class name it invokes you can see spelled out clearly in the xJCL.
3. This is where the substitution props are fed into the batch job. The default values were set higher in the xJCL. But as mentioned, we will override those defaults at the time of submission.

- Log onto the Job Management Console:

<http://<host>:<port>/jmc>

Where <host> is the TCP host for the server, and <port> is the normal HTTP port.

- Click on the "Submit a job" link:



- Use the "Browse" button under "Local file system" to navigate to and select the SimpleCIxJCL.xml file:



- On the same browser screen, check the "Update substitution properties" checkbox and then click on "Submit":

Update substitution properties  
 Delay submission  
 \* Start date (yyyy-MM-dd)  
 2013 - 05 - 24  
 \* Start time (HH:mm:ss)  
 13 : 20 : 11

- You will see the following. Note how the values fields are populated with the default values coded in the xJCL. Here is where you may override the defaults:

Property	Value
calctime	30
outfile	/path/simpleciout.txt

Change the value of 30 to 10, and change the path to one you know the server ID will have write permissions (such as /tmp or whatever is appropriate for your location).

Then click the "OK" button.

- You should then see a message like this:

Messages  
 Successfully submitted the job definition SimpleCIxJCL.xml with job ID SimpleCIEar:00000.

- Now click on the "View jobs" link. You should see something like this:

Select	Job ID	Submitter	Last Update	State	Node	Application Server
<input type="checkbox"/>	<a href="#">SimpleCIEar:00000</a>	wbadmin	2013-05-24 13:12:53.395	Ended	wbnodec	wbsr02c

Filtered: 1 Total: 1

Depending on how quickly you click on "View jobs" the "State" field will either be "Submitted" (xJCL accepted, but job not yet dispatched); "Executing" (job running); or "Ended" (job complete, as shown here).

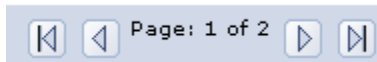
- Click on the link that represents the job:

Select	Job ID
<input type="checkbox"/>	<a href="#">SimpleCIEar:00000</a>

Filtered: 1 Total: 1

- The job log should be displayed. You will notice the original xJCL as submitted, then a second section with the xJCL as resolved after substitution properties are factored in.

- ❑ Scroll to the bottom and you'll see icons to move forward or backwards in the pages of the job log:



Click the "right arrow" to advance to the second page.

The second page will show you the output of this job's execution.

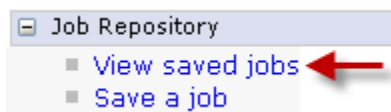
- ❑ *Optional:* submit the job again with a longer execution time and observe the "states" the job goes through -- Submitted, Executing, and Ended.

### Storing xJCL in the Job Repository and submitting from there

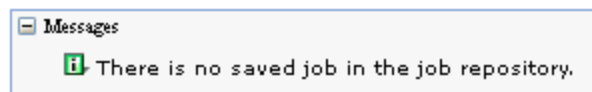
Up to this point you have submitted the SimpleCI job with xJCL from your workstation. Now you will save a copy of the xJCL in one of the relational tables<sup>23</sup> and submit the job from there.

Do the following:

- ❑ In the Job Management Console (JMC), click on the "View saved jobs" link:



You should see an empty list with the following message:



- ❑ Now click on the "Save a job" link:



- ❑ You should see the following:

\* Job name:

\* xJCL path:

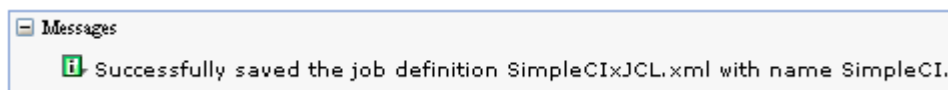
Audit String:

Replace the job if the specified job name exists

Then:

- In the "Job name" field type in SimpleCI
- In the "xJCL path" field, use the "browse" button to select the SimpleCIxJCL.xml file you extracted from the downloaded sample ZIP file.
- Click on "Save"

- ❑ You should see this message:



<sup>23</sup> In the table <schema>.JOBREPOSITORY.

- Once again, click on the "View saved jobs" link. You should see your saved job:

Select	Name
<input type="checkbox"/>	<a href="#">SimpleCI</a>

Filtered: 1 Total: 1

- Click on the link that represents your save job. You will see the xJCL that is saved:


This panel shows the definition of batch job SimpleCI.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--(C) Copyright IBM Corp. 2005 - All Rights Reserved. DISCLAIMER: The following source
<job name="SimpleCIJar" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jndi-name>ejb/com/ibm/was/ci/SimpleCIJarCIController</jndi-name>
  <substitution-props>
    <prop name="calctime" value="30"></prop>
    <prop name="outfile" value="/path/simpleciout.txt"></prop>
  </substitution-props>
  <!--### STEP 1 ###-->
  <job-step name="Step1">
    <classname>com.ibm.websphere.ci.samples.SimpleCIWork</classname>
    <props>
      <prop name="calculationTimeInSecs" value="${calctime}"></prop>
      <prop name="outputFileName" value="${outfile}"></prop>
    </props>
  </job-step>
</job>
```

- Click on the "Submit a job" link, then click the "Job repository" radio button:


**Job Definition**

Local file system  
 \* Specify path to xJCL


Job repository   
 \* Specify job name

Update substitution properties

Click the "Browse" button to see and select your saved xJCL:

Select	Name
<input checked="" type="radio"/> 	SimpleCI

Filtered: 1 Total: 1



- You will see your selected job in the "Specify job name" field. If you click the "Update substitution properties" checkbox you are then able to submit it just as you did before.

Job repository  
 \* Specify job name

Update substitution properties

Delay submission

### Running the XDCGIVT sample

The SimpleCI sample was good to use as the initial verification sample because it required no input or output operations. The xJCL was relatively short with only two substitution variables.

However, SimpleCI did *not* exercise many of the features of WebSphere Java Batch. For example, it did not use the "Batch Data Stream Framework" support for input and output stream management, and it did not make use of the checkpoint processing support.

Another supplied sample will show these things. The sample is called "XDCGIVT."

Do the following:

- ❑ Go back to the samples URL (see page 15), click on "Samples, V8.5" and then the "Downloads" tab just as you did for SimpleCI.

- ❑ Scroll down and locate the "Java Batch" section, then locate the following:

**Java Batch IVT sample**

Install, run, and monitor a batch application that reads and writes files using the Java Batch Installation Verification Test (IVT) sample.

Download sample: [FTP](#) [HTTP](#) | [More information](#)

- ❑ Download that ZIP file just as you did the SimpleCI sample, and unzip the contents into a new folder for XDCGIVT.
- ❑ Using the Admin Console, deploy the XDCGIVT.ear file as you would any EAR file. There are no special requirements ... take the fast path, map the application to the same server you used for SimpleCI, then save and synchronize.
- ❑ In the Admin Console, go to *Applications* ⇒ *All applications*. You should see XDCGIVT installed. Select the checkbox next to XDCGIVT and then start the application.

You can administer the following resources:					
<input type="checkbox"/>	<a href="#">SimpleCIEar</a>	Base edition	Active	Java 2 Platform, Enterprise Edition	
<input type="checkbox"/>	<a href="#">XDCGIVT</a>	Base edition	Active	Java 2 Platform, Enterprise Edition	

- ❑ In the Job Management Console, click on "Submit a job," then on the radio button for "Local file system," then use the "Browse" button to navigate to and select the following xJCL file from the folder in which you unzipped the sample:



**Note:** That xJCL -- XDCGIVTtxt2txtxJCL.xml -- uses UNIX files as its input and output data sources. It's the simplest to start with.

The XDCGIVTtxt2db2txtxJCL.xml file uses a relational table. That requires the creation of a DB2 table. The sample supplies the DDL to do that. We will show how to use DB2 with this sample application after we do the text file operation.

- ❑ Click on the "Update substitution properties" checkbox, then click "Submit."

- The xJCL has 9 substitution properties defined<sup>24</sup>. Here's what you should see:

Property	Value
checkPoint	10
debugEnabled	false
fileEncoding	8859_1
inputDataStream	/path/input-text.txt
numberRecords	100
outputDataStream	/path/output-text.txt
perfEnabled	true
supportclassIn	com.ibm.websphere.batch.devframework.datas
supportclassOut	com.ibm.websphere.batch.devframework.datas

**Note:** The properties of interest at this point are:

**inputDataStream** -- this defines the UNIX path and file that will be used as the input file. The sample will create the input records in this file.

**outputDataStream** -- this defines the UNIX path and file that will be used as the output file.

**numberRecords** -- this defines how many records will be created and processed.

**checkPoint** -- this defines how frequently a checkpoint will be taken. This is what exercises the checkpoint support function of WebSphere Java Batch.

- Change the /path portion of **inputDataStream** to be a path you know to be valid and one the application server ID has write access to.
- Change the /path portion of **outputDataStream** to be the same as you used for the *input* data stream property.
- Click the "Submit" button.
- Click on the "View jobs" link to see the job execute and end.
- Click on the link that represents the finished job and go to page two of the held output. There you will see evidence of the checkpoint processing taking place every 10 records.

## XDCGIVT with DB2

To use XDCGIVT with DB2 requires you create a database table. The DDL for that is supplied in the `CreateIVTTablesDB2zOS` file supplied with the sample. The xJCL then provides two additional substitution properties:

Property	Value
DB2info	IVTSCHEMA.IVTTABLE
DB2jndi	jdbc/IVTdbxa

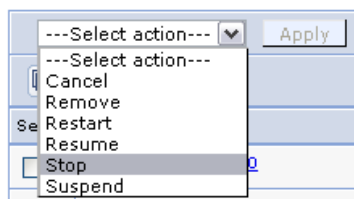
The first allows you to specify the schema and table name, the second the JDBC data source JNDI name to use to access DB2.

Do the following:

- Locate the `CreateIVTTablesDB2zOS` file in the folder where you unzipped the XDCGIVT sample zip file.

<sup>24</sup> If you see 12, including "DB2info," then you've selected the DB2 copy of the xJCL by mistake. Click the "Cancel" button and select the "txt2txt" copy of the xJCL.

- Upload that file to **in binary** to your z/OS system<sup>25</sup>. Upload to either a UNIX file or an MVS data set.
- Work with your DBA to update the variables in the DDL and create the table<sup>26</sup>.
- Determine what JDBC data source you will use to access the new XDCGIVT table.
  - If the table is in the same DB2 instance as the LRS tables, then you can use the data sources you created for that (`jdbc/lrsched` or `jdbc/lree`)
  - If the table is in the same DB2 instance but you wish to use a different data source (such as one defined as Type 2 using RRS rather than cell-scoped XA as the LRS and LREE data sources are defined), then create the provider and data source and note the JNDI name of the data source.
  - If the table is in a different DB2 instance from the LRS/LREE tables, then you can't use the LRS/LREE data sources to access. Define a new provider and data source.
- In the Job Management Console, click on "Submit a job."
- Using "Local file system," select the `XDCGIVTtxt2db2txtxJCL.xml` file and click on "Update substitution properties."
- Update the following properties:
  - `DB2info` Set equal to the schema and table name for the XDCGIVT table created for this sample application
  - `DB2jndi` Set equal to the JNDI name of the JDBC data source you will use to access DB2 and the table.
  - `inputDataStream` Update the `/path` information to be a path you know to be valid and one the application server ID has write access to.
  - `outputDataStream` Update the `/path` information to be a path you know to be valid and one the application server ID has write access to.
- Click the "Submit" button.
- Click on the "View jobs" link to see the job execute and end.
- Click on the link that represents the finished job and go to page two of the held output. There you will see evidence of the checkpoint processing taking place every 10 records.
- Optional:* browse the contents of the DB2 table. You should see 100 records ... one each for the value of the `numberRecords` substitution property.
- Optional:* set the `numberRecords` substitution property to some much larger number -- 10,000 or so -- and submit. Then go to "View jobs". You should see your job in an "Executing" state. Select the checkbox next to your executing job, and from the pulldown list select "Stop" then click "Apply":



Your job should go into a "Restartable state. If you look at the job log you'll see it will have stopped on a checkpoint interval. Then select "Restart" and "Apply." You should see your job picks back up from the last saved checkpoint interval<sup>27</sup>.

<sup>25</sup> The file is encoded in EBCDIC in the ZIP file. You want it to be EBCDIC in z/OS. Upload in binary to accomplish that.  
<sup>26</sup> The DDL calls for the creation of a `STOGROUP`, `DATABASE` and a `TABLESPACE`, along with a `TABLE` and an `INDEX`. For the sake of testing it may be easiest to create this table in the same `STOGROUP`, `DATABASE` and `TABLESPACE` as the LRS/LREE tables you created back on page 5. Again, work with your DBA on this.  
<sup>27</sup> The stop/restart process would work when XDCGIVT is operating against UNIX files; DB2 is not required for this stop/restart processing.

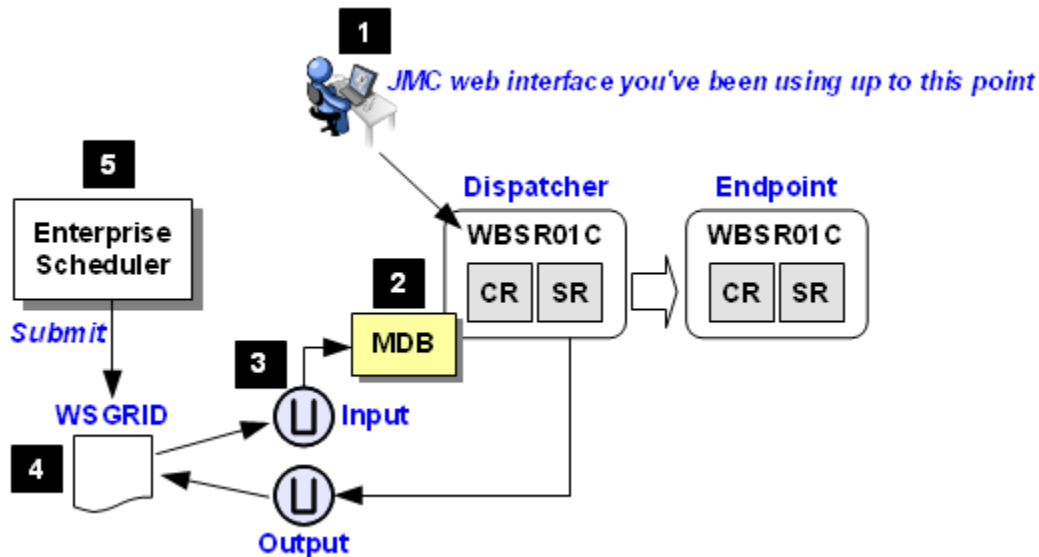


## Enabling the WSGRID Interfaces

The purpose of "WSGRID" is to allow you to integrate existing enterprise scheduler functions<sup>28</sup> with WebSphere Java Batch. The objective is to have WebSphere Java Batch be viewed as *part* of the overall enterprise batch process, *not* an island off to the side.

### Conceptual overview

The integration is performed using a supplied Message Drive Bean (MDB) application to the Job Dispatcher function:



### Notes:

1. Up to this point in the exercises we have used the Job Management Console (JMC), which is the *browser interface* of the Dispatcher function.
2. In this section we will add a Message Driven Bean interface<sup>29</sup>. The MDB will listen on an input queue<sup>30</sup> for job submission requests. It will take the submission request off the queue and submit it.
3. Two queues are used: an input queue and an output queue. The input queue is used to submit jobs; the output queue is used to feed job log output back.
4. In this picture "WSGRID" represents a supplied utility program<sup>31</sup> that puts the job submission request on the input queue, and gets back the job log output messages.
5. The enterprise scheduler function submits<sup>32</sup> (or invokes) the WSGRID utility program. **The WSGRID program stays active for the life of the Java Batch program running in WebSphere Java Batch.**

The enterprise scheduler sees WSGRID running and believes it to be the batch job, but in reality the *real* batch job is running in WebSphere Java Batch. WSGRID mirrors the activity of the Java batch job: it is active while the Java batch job is active; it ends when the Java batch job ends; if the Java batch job fails then WSGRID fails with the same return code; the output of the Java batch job is fed back to WSGRID's standard output for archival purposes.

<sup>28</sup> For example, IBM Tivoli Workload Scheduler. See [http://en.wikipedia.org/wiki/List\\_of\\_job\\_scheduler\\_software](http://en.wikipedia.org/wiki/List_of_job_scheduler_software) for a list of other vendors that provide job scheduler software.

<sup>29</sup> A supplied shell script will be used to install the MDB application and configure the interface. It's all fairly well automated.

<sup>30</sup> As you will see, the queue may exist either in WebSphere MQ or in the WAS Service Integration Bus (SIBus) function.

<sup>31</sup> On z/OS this takes one of two forms: a native z/OS program that uses MQ BINDINGS mode to work with MQ directly, or a UNIX shell script that puts messages on the SIBus queues. The native z/OS program is WAS z/OS only; the shell script is common to all platforms.

<sup>32</sup> If your enterprise scheduler software can submit JCL, invoke a shell script, or run a Windows BAT file, it can work with WSGRID.

**On z/OS, two approaches**

As mentioned, for WAS z/OS two approaches are available:

1. A z/OS native program implementation of WSGRID that uses MQ BINDINGS mode to PUT and GET messages. The MDB interface in WAS z/OS is also configured to listen on the MQ queue using BINDINGS mode. This is a z/OS exclusive.

**Note:** And because it is BINDINGS mode, it's *very* fast.

2. An implementation that uses a supplied shell script to invoke WSGRID as a Java program. That interacts with the Service Integration Bus (SIBus) of WAS to put and get messages. This option is available on every supported platform.

The native z/OS program must be run on z/OS, and further the WSGRID program must run on the same LPAR as the MQ Queue Manager to which it connects. It uses BINDINGS mode, and BINDINGS mode must be on the same LPAR<sup>33</sup>. The WSGRID invoked with a shell script operates across the network to put and get messages from the queues; it may be located local or remote to the WebSphere Java Batch dispatcher function server.

**Product InfoCenter articles on each approach**

The InfoCenter<sup>34</sup> provides very good articles on how to set up each implementation approach:

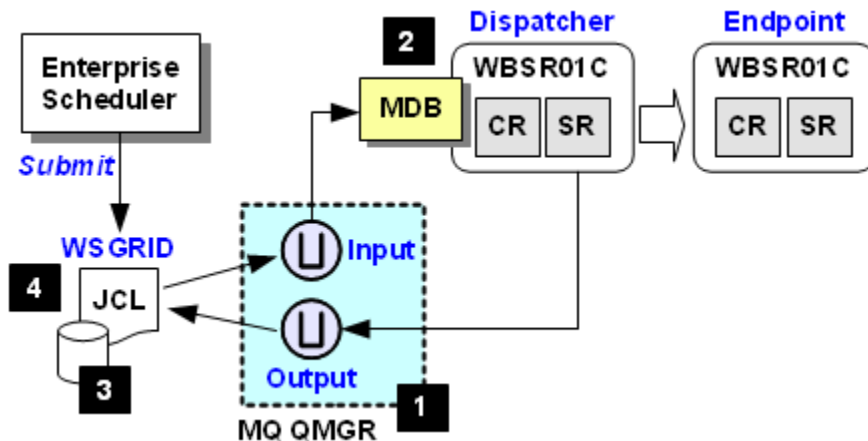
z/OS Native and MQ	WAS z/OS V8.5 InfoCenter search string: <b>tgrid_zoscgnative</b>
Java-based and SIBus	WAS V8.5 InfoCenter search string: <b>tgrid_cgexternalconfig</b>

The step-by-step actions spelled out in this document are based on the InfoCenter articles. We will show WSGRID and MQ next, and WSGRID with the SIBus starting on page 33.

**WSGRID and MQ**

**Overview**

The following picture provides an overview of the process you will follow:



Notes:

1. Configure the input and output queue in the MQ Queue Manager
2. Install the MDB interface code into the Job Dispatcher server
3. Unpack and create the WSGRID program load module
4. Configure the WSGRID JCL to submit a job to WebSphere Java Batch

<sup>33</sup> That does not mean the WebSphere Java Batch Dispatcher server must be on the same LPAR. You could configure the queue local to WSGRID as a transmit queue, with MQ then responsible for delivering the message to a queue on a QMGR local to the dispatching server.

<sup>34</sup> For WAS V8.5: <http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/index.jsp>

## Create queues in MQ

Do the following:

- Identify the MQ queue manager in which the queues will exist:

Queue Manager Name:	
---------------------	--

- Identify the path location where MQ is installed<sup>35</sup>:

MQ install path:	
------------------	--

- Determine the high-level qualifier for your MQ installation's `SCSQAUTH` and `SCSQANLE` data sets. Determine if those are in `LINKLIST` or if your server needs to `STEPLIB` to them to accomplish `BINDINGS` mode.

High-level qualifier:	
STEPLIB Needed?	<input type="checkbox"/> Yes <input type="checkbox"/> No

- Work with your MQ administrator to define two queues in the queue manager. The names may be whatever you wish<sup>36</sup>. They must be defined as shared mode<sup>37</sup>. Capture the queue names here:

Input Queue:	
Output Queue:	

## Install MDB interface into dispatcher server

Do the following:

- Make sure your Deployment Manager is up and running.
- Establish a Telnet session<sup>38</sup> to your z/OS system.
- Log on as the WAS Admin ID, or switch users to this ID.

**Note:** We have you do this so the WADMIN client can establish the SSL connection to the running Deployment Manager. The WAS Admin ID typically by default has what it takes to establish the SSL -- that is, access to the keyring that holds the CA certificate that signed the DMGR's server certificate.

- Change directories to your Deployment Manager's `/bin` directory.
- Compose the following command (**as one line**) in Notepad<sup>39</sup> (or other editor):  

```
./wsadmin.sh -lang jython -conntype SOAP -host aaaaa -port bbbbbb
               -user ccccc -password ddddd -f installWSGridMQ.py -install
               -node eeeee -server fffff -qmgr gggggg -inqueue hhhhh -outqueue jjjjj
```

Where:

- **aaaaa** = the TCP host on which your DMGR is listening
- **bbbbbb** = the SOAP port of the DMGR server
- **ccccc** = the WAS Admin ID<sup>40</sup>
- **dddddd** = the WAS Admin ID password

<sup>35</sup> For example: `/usr/lpp/mqm/V7R1M0`

<sup>36</sup> For example: `WBCELL.INPUT.QUEUE` and `WBCELL.OUTPUT.QUEUE`. That's what we used when testing this function for this document.

<sup>37</sup> This is key -- the queues will be accessed by two separate processes: (1) the function in the Dispatcher server, and (2) the WSGRID native program. If not shared then it's a race to see who accesses first, then the other is locked out and this solution doesn't work.

<sup>38</sup> Or ssh, using whatever terminal program you prefer (PuTTY, TeraTerm). OMVS tends not to work well because the input field for OMVS is limited and the command to be entered can be quite long.

<sup>39</sup> It's much easier to compose the command separately and paste than it is to type it into the terminal session.

<sup>40</sup> Using the WAS Admin ID and password assures the authority needed to perform the tasks this script performs.

- **eeeeee** = the node *long* name of the node in which the dispatcher server runs
- **fffff** = the server *long* name of the dispatcher server
- **ggggg** = the MQ queue manager name
- **hhhhh** = the input queue name
- **jjjjj** = the output queue name

- Copy that from your editor session, paste it into your Telnet session and submit.
- The process will produce many lines of output. Look for the following as an indication of success:

```
saving config...
Done saving..
installWSGridMQ.py INFO: Configuration was saved and synchronized to
the active nodes
Install complete.
```

- In the Admin Console, go to *Environment* ⇒ *WebSphere variables*. Set the scope for the node in which your Dispatcher server runs, for example:

Node=wbnodec 

- Locate the variable `MQ_INSTALL_ROOT` and change its value<sup>41</sup> to match the MQ install path you captured back on page 27.
- Click "OK" and save and synchronize the changes.
- In your Telnet session change directories to the `/bin` directory of the managed node for which you just changed the `MQ_INSTALL_ROOT` variable<sup>42</sup>.
- From that directory, issue the following command:
 

```
./osgiCfgInit.sh -all
```

 You should see several messages indicating the OSGi cache has been cleared.
- Stop the server that hosts the Dispatcher function.
- If you determined STEPLIB was needed for your so it could access the MQ `SCSQAUTH` and `SCSQANLE` data sets, add that STEPLIB to the server's controller and servant region JCL start procedures.

**Important!** The next few steps are very important. It involves removing the `REUSASID=YES` parameter from the configured `START` command<sup>43</sup> for the controller of the server that hosts the Dispatcher function. Failure to do this will result in a `x0D3` abend because MQ BINDINGS mode is configured for the server.

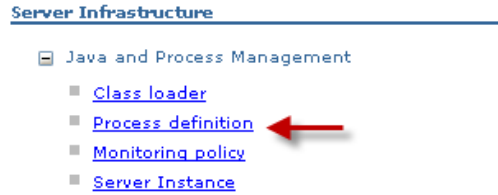
- In the Admin Console, go to *Servers* ⇒ *Server Types* ⇒ *WebSphere application servers*.
- Click on the link that represents the server that hosts the Dispatcher function (or what WebSphere Java Batch calls the "Job Scheduler" function).

<sup>41</sup> Copy the current value and paste it into the "Description" field as a way to preserve the previous value.

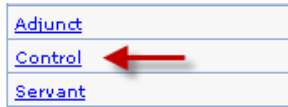
<sup>42</sup> For example, for us that was `/wasv85config/wbcell/wbnodec/AppServer/bin`

<sup>43</sup> This protects against the case where the server is started from the Admin Console. Starting the server from the z/OS operator console would work since you control the `START` command and you can omit the `REUSASID=` parameter. But when starting from the Admin Console the configured `START` command is used, and that has `REUSASID=YES` coded. Follow these instructions because you can be certain someone, at some point in time, is going to start that server from the Admin Console.

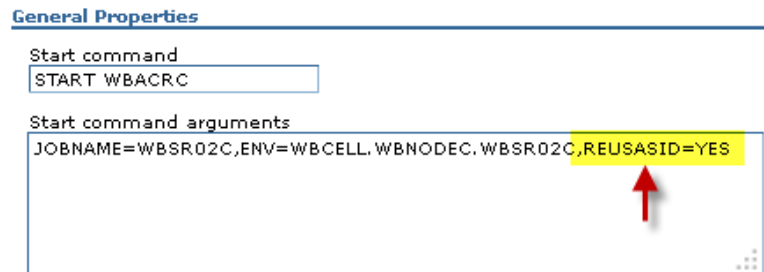
- Then click on "Process Definition":



- Click on "Control":

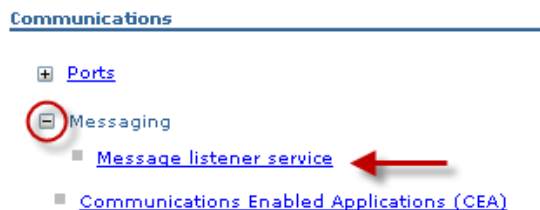


- **Remove** REUSASID=YES as well as the comma that precedes it:



Leaving just JOBNAME= and the ENV= string.

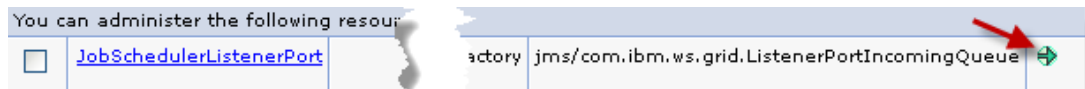
- Click "OK".
- Save and synchronize the changes.
- Start the server that hosts the Dispatcher function.
- When the server appears to be fully initialized, go into the Admin Console under *Servers* ⇒ *Server Types* ⇒ *WebSphere application servers* and click on the server that hosts the Dispatcher function.
- Click on "Message listener server" under "Messaging":



- Click on "Listener Ports":



- Make sure the status of "JobSchedulerListenerPort" is showing started with a green arrow:



## Create the WSGRID load module

The WSGRID program that serves as the link between the enterprise scheduler and the MDB listener function you just installed is shipped as a packed UNIX file. You need to get that out of UNIX and into a load module data set to use it.

To do that a REXX script called `unpackWSGRID` is supplied to copy the file from UNIX to an XMIT data set, then from TSO you `RECEIVE` into your load module data set.

**Note:** The REXX script is going to try to allocate the XMIT data set. To do that the ID under which the script runs must have authority to allocate data sets (the SAF `DATASET` class). If when invoking the script you see "ERROR: Cannot allocate <dataset>" then it is likely due to a lack of authority. If RACF you will see an ICH408I message indicating an intent of UPDATE but an access allowed of READ or NONE.  
Be sure to use an ID with sufficient authority.

Do the following:

- Connect to your system using Telnet<sup>44</sup>.
- Log on with an ID you know to have data set allocation authority.
- Change directories to your application server node's `/bin` directory.
- To see the script parameters, invoke the script *without* any parameters:

```
> ./unpackWSGRID
Syntax:
unpackWSGRID <was home> <hlq> <work dir> <batch> <debug>
  <was home> specifies required WAS HOME directory
  <hlq> specifies optional HIGH LEVEL QUALIFIER of output data sets
           default = <Logged_on_userid>
  <work dir> specifies optional working directory
           default = /tmp
  <batch> specifies optional run mode for this script
           specify 'batch' or 'interactive'
           default is 'interactive'
  <debug> specifies optional debug mode
           specify 'debug' or 'nodebug'
           default is 'nodebug'
```

Note the default value of `/tmp` for `<work dir>`. The answer to that determines the command syntax you enter<sup>45</sup>:

If `/tmp` is okay:

```
./unpackWSGRID <was home>
```

Where `<was home>` is the location of the application server node home; for example, `/wasv85config/wbcell/wbnodec/AppServer`.

If `/tmp` is *not* okay, then:

```
./unpackWSGRID <was home> <hlq> <work dir>
```

Where

`<was home>` is the application server node home directory; for example, `/wasv85config/wbcell/wbnodec/AppServer`.

`<hlq>` is the high-level qualifier for the XMIT data set; for example, `USER.CG.LOAD`

`<work dir>` is the work directory other than `/tmp`; for example, `/u/user`

<sup>44</sup> In this case OMVS will also work. The input string is not that long.

<sup>45</sup> The parameter for the temporary directory is the third positional parameter. If you need to specify a non-default temp directory, then you must also specify the first and second parameters as well.

- Compose the command string for your environment.
- Invoke the script. You should see *something* like this (with the highlighted **y** indicating where user input was prompted for and issued):

```
Unpack WSGRID with values:
```

```
WAS_HOME=/wasv85config/wbcell/wbnodec/AppServer
HLQ      =USERID46
WORK_DIR=/tmp
BATCH    =INTERACTIVE
DEBUG    =NODEBUG
```

```
Continue? (Y|N)
```

```
y
```

```
User response: Y
```

```
Unzip /wasv85config/wbcell/wbnodec/AppServer/bin/cg.load.xmi.zip
```

```
inflated: cg.load.xmi
```

```
Move cg.load.xmi to /tmp
```

```
Delete old dataset 'USERID.CG.LOAD.XMI'
```

```
Allocate new dataset 'USERID.CG.LOAD.XMI'
```

```
Copy USS file /tmp/cg.load.xmi to dataset 'USERID.CG.LOAD.XMI'
```

```
Delete USS file /tmp/cg.load.xmi
```

```
Delete old dataset 'USERID.CG.LOAD'
```

```
Go to TSO and issue RECEIVE INDSN('USERID.CG.LOAD.XMI') to create CG.LOAD
```

- Go to TSO Option 6 and issue the command indicated by the last message. You should see something like this:

```
INMR901I Dataset BBUILD8.CG.LOAD from BBUILD8 on PLPSC
INMR154I The incoming data set is a 'PROGRAM LIBRARY'.
INMR906A Enter restore parameters or 'DELETE' or 'END'
```

- Press the Enter key to proceed. You should see something like:

```
INMR001I Restore successful to dataset 'USERID.CG.LOAD'
***
```

- Optional:* rename the created load dataset to a high-level qualifier that starts with your cell short name. We named it `WBCELL.WSGRID.LOAD`.

### Configure JCL and validate

The load module created in the previous section was called `WSGRID`. What you need now is a set of JCL that will run that program and pass in the required values to it can connect to the MQ queue and submit the job.

Do the following:

- Create a JCL member that looks something like the following. Notes after the example correspond to the numbered blocks:

```
//WSGRID JOB 1, 'WSGRID', CLASS=A, REGION=0M, NOTIFY=?,
// USER=WBADMIN, PASSWORD=***** 1
//SUBMIT EXEC PGM=WSGRID
//STEPLIB DD DISP=SHR, DSN=WBCELL.WSGRID.LOAD 2
//          DD DISP=SHR, DSN=SYS1.MQM710.SCSQLOAD 3
//          DD DISP=SHR, DSN=SYS1.MQM710.SCSQAUTH
//SYSPRINT DD SYSOUT=* 4
//WGCNTL DD * 5
queue-manager-name=QMLC 6
scheduler-input-queue=WBCELL.INPUT.QUEUE 7
scheduler-output-queue=WBCELL.OUTPUT.QUEUE
timeout=5000
submit-timeout=30000
//WGJOB DD * 8
```

<sup>46</sup> This will default to the ID under which the script is invoked.

```

repository-job=SimpleCI 9
//WGSUBS DD * 10
substitution-prop.calctime=60 11
substitution-prop.outfile=/tmp/simpleciout.txt
//*

```

**Notes:**

1. The job submitted using WSGRID is subject to the same EJBROLE authority checking as the Job Management Console. Earlier we had you assign the WAS Admin ID to the `lradmin` EJBROLE. Coding `USER=` on the JOB card and providing the WAS Admin ID and password is the simplest way to insure the submitted job will have the authority. You may also grant whatever ID your WSGRID JCL runs under `READ` to `lradmin` or `lrsubmitter` EJBROLES defined earlier.

The symptom for lack of authority is `RC=4080` on the submitted JCL and "Login failure during job submission" as part of the output.

2. STEPLIB to the load module in which the WSGRID program resides.
3. STEPLIB to the MQ data sets if necessary (it was for our test system).
4. The "control properties" provided inline with the JCL. These are documented in the InfoCenter under search string `rgrid_xdcgproperties`.
5. The Java Batch job output flows back to SYSOUT, which means SYSPRINT DD in the JES held output.
6. The first control property names the MQ queue manager to connect to using BINDINGS mode.
7. The next two specify the input and output queues to use.
8. The "job properties" provided inline with the JCL. These are documented in the InfoCenter under search string `rgrid_xdcgcommon`.
9. In this example the xJCL will come from the repository<sup>47</sup>. Earlier we had you same the SimpleCI sample xJCL in the repository. Here we're using it with WSGRID.
10. The job's substitution properties provided inline with the JCL. We determined what those properties were by using the JMC to submit with substitution properties<sup>48</sup>:

Property	Value
calctime	30
outfile	/path/simpleciout.txt

11. Here the execution time is 60 seconds because we would like to reinforce a key point about WSGRID, which is this:

The submitted WSGRID job stays active for the life of the Java Batch job running back in WebSphere Java Batch. By setting this execution time at 60 you will have ample time to see this for yourself. JES will show WSGRID active for 60 seconds.

- Submit your JCL
- When the job completes, look at the held output for the WSGRID job you submitted. You should see the same output you would see in the JMC for the job log.

**Summary of native WSGRID on z/OS**

In this exercise we didn't actually integrate with a real enterprise scheduler program. But we hope you can see how easy it would be -- the WSGRID JCL you created is the job the enterprise scheduler submits. That job stays active for the life of the Java batch job in WebSphere Java Batch. To the enterprise scheduler, the WSGRID job *is* the batch job. The enterprise scheduler doesn't know anything about WebSphere Java Batch in the background.

<sup>47</sup> It is also possible to point to a location in the UNIX file system on z/OS.

<sup>48</sup> Rather than actually submit the job we simply canceled after capturing that information. You could also get those values from the xJCL.



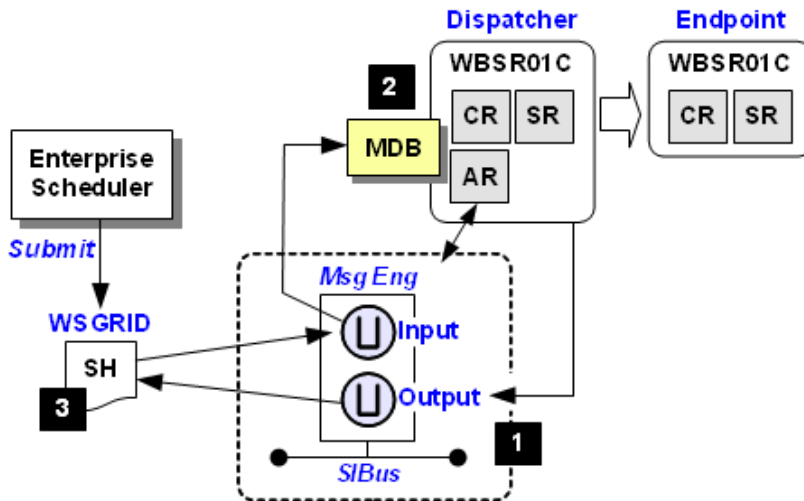
If you had 50 different Java Batch jobs that were part of a larger 200 job batch process, then you would create 50 JCL jobs for the enterprise scheduler to submit. Each would be coded to represent the Java Batch job in WebSphere Java Batch.

### WSGRID and Service Integration Bus

In some ways this is similar to the native WSGRID implementation just covered. But rather than use MQ, this approach uses the "default messaging" function of WAS<sup>49</sup>.

#### Overview

The following picture provides an overview of the process you will follow:



Notes:

1. Run a WSADMIN script that creates the SIBus, Messaging Engine and Destination (queue) definitions inside WAS. This results in the starting of the "Adjunct Region," which is where the messaging engine runs.
2. The WSADMIN script also deploys the MDB function into the server.
3. The client in this case is a supplied shell script to connect to the SIBus and pass messages back and forth. The script may be run from JCL (using BPXBATCH) or directly if you prefer to integrate with your enterprise scheduler using a command line interface.

### Run wsgridConfig.py script to configure default messaging

Do the following:

- Make sure your Deployment Manager is up and running. The WSADMIN script will connect to the DMGR to make the changes.
- In the Admin Console, go to *Servers* ⇒ *Server Types* ⇒ *WebSphere application servers*.
- Click on the link that represents the server in which you have the Dispatcher function configured.
- Under "Communications," expand the list of ports for the server:



<sup>49</sup> The two may coexist within the same server, though it is not necessary to implement both.

- Locate the two `SIB_ENDPOINT` ports, for example:

SIP_DEFAULTHOST_SECURE	32078
SIB_ENDPOINT_ADDRESS	32073
SIB_ENDPOINT_SECURE_ADDRESS	32074
SIB_MQ_ENDPOINT_ADDRESS	32075

Capture the values for your server here:

SIB_ENDPOINT_ADDRESS	
SIB_ENDPOINT_SECURE_ADDRESS	

- Open up a Telnet session to your system and log on as WAS Admin ID<sup>50</sup>.
- Change directories to your Deployment Manager's `/bin` directory.
- In Notepad<sup>51</sup> (or other editor), compose the following command<sup>52</sup> as *one line*:
 

```
./wsadmin.sh -lang jython -conntype SOAP -host aaaaa -port bbbbbb
                -user ccccc -password dddddd -f wsgridConfig.py -install
                -node eeeeee -server fffff
                -providers "host,port1;host,port2" -filestoreroot ggggg
```

**Notes:** Pay particular attention to the syntax of the `-providers` string: double quotes surround the string, comma between host and port values, semi-colon separates the first host,port pair from the second<sup>53</sup>.

If later you wish to remove this function use the same script but with `-remove`:  
`./wsadmin.sh <connect_parms> -remove -node <node> -server <server>`

Where:

- *aaaaa* -- is the host where the Deployment Manager is listening
  - *bbbbbb* -- is the DMGR's SOAP port
  - *ccccc* -- is the WAS Admin ID
  - *dddddd* -- is the WAS Admin ID password
  - *eeeeee* -- is the node *long* name where the Dispatcher function server is configured
  - *fffff* -- is the server long name of the Dispatcher function server
  - *host* -- is the host on which the Dispatcher function server is configured
  - *port1* -- is the `SIB_ENDPOINT_ADDRESS` port value
  - *port2* -- is the `SIB_ENDPOINT_SECURE_ADDRESS` port value
  - *ggggg* -- is a where the SIBus file store will be created<sup>54</sup>.
- Paste the line into the Telnet session and submit for execution.

If there's a problem with syntax, it'll tell you fairly quickly. Correct and resubmit.

However, if the command is correct it will proceed to build the various artifacts. Many lines of output will be generated. It will finish with something like this:

```
Creating JMS Activation Spec
wsgridConfig.py INFO: created JMS activation spec com.ibm.ws.grid.ActivationSpec
saving config...
Done saving..
INFO: getting cell/node/server...
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
```

<sup>50</sup> Again, we have you do this because it makes establishing the SSL connection to the Deployment Manager easier when you're logged on as the WAS Admin ID. Other IDs can work, provided they have access to the keyring with the CA cert that signed the server certificate.

<sup>51</sup> Composing the command in an editor first and then pasting it into the Telnet session is often easiest.

<sup>52</sup> This command is slightly different if the target is a cluster. Here we're showing a single server.

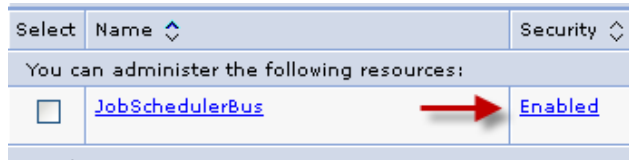
<sup>53</sup> For an example, see "wsgridConfig.py command example" on page 42.

<sup>54</sup> You may omit the `-filestoreroot` parameter and its value if you wish. If you omit, it will default to `/tmp`.

- In the Admin Console, go to *Service integration* ⇒ *Buses*:



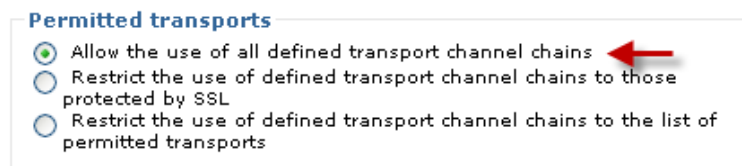
- When the SIBus display comes up, you should see one bus: `JobSchedulerBus`. Note that by default security is "enabled." We're going to disable security<sup>55</sup>. Click on the "Enabled" link:



- Uncheck the "Enable bus security" box:



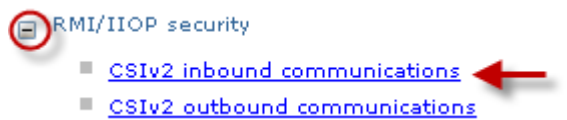
- In the same bus security panel, set the "Permitted transports" to the first radio button, which allows connections across all transport chains:



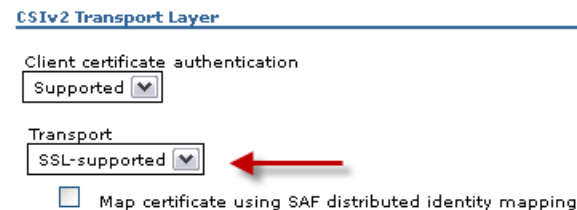
- Click "OK," then save and synchronize.
- Next, go to *Security* ⇒ *Global security*:



- Over on the right side of the screen, locate and expand "RMI/IIOP security," then click on the "CSIv2 inbound communications" link:

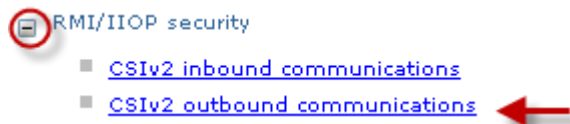


- Then, under "CSIv2 Transport Layer," change the "Transport" pulldown from "SSL-required" to "SSL-supported:"

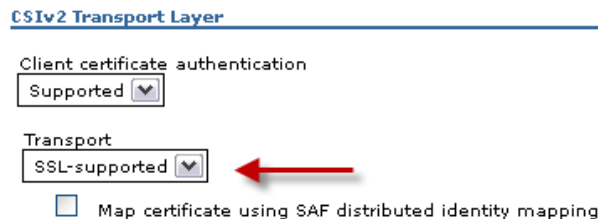


<sup>55</sup> Configuring bus security is complicated and will be a distraction to the more central point of this document, which is showing the essential functions of WebSphere Java Batch.

- ❑ After making the change, click on "OK". That should take you back to the previous Global security panel.
- ❑ Now click on "CSIv2 **outbound** communications:"



- ❑ Once more, change the Transport pulldown to "SSL-supported:"



- ❑ Click "OK"
- ❑ Save and synchronize all changes.
- ❑ Stop and restart the **entire cell** -- servers, Node Agent, DMGR and Daemon.
- ❑ Restart all the servers.

### Run WSGrid.sh shell script to submit job

Do the following:

- ❑ Create a JCL job based on this sample:

```
//WSGRIDJ JOB <your JOB information>
//NODEC      EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN   DD *
BPXBATCH SH +
  /<WAS_home_dir>/DeploymentManager/profiles/default+
  /bin/WSGrid.sh +
  /<path>/wsgrid.cntl +
  /<path>/wsgrid.props +
  1> /<path>/wsgridj.out +
  2> /<path>/wsgridj.err
/*
/*
```

Substitute in a value for `<WAS_home_dir>` to match your environment.

The value for `<path>` may be any directory of your choosing. The key is the directory must allow READ and WRITE permission for the ID that runs the job you just created<sup>56</sup>.

- ❑ In the directory indicated by `<path>` in the previous step, create the `wsgrid.cntl` file and populate it, in EBCDIC, with the following:

```
scheduler-host=<host>
submitter-userid=<ID>
submitter-password=<password>
#debug=true
debug=false
scheduler-port=<port>
```

<sup>56</sup> It will *read* the `cntl` and `props` file; it will *write* the `out` and `err` files.

**Where:**

- `<host>` is the TCP host of the system on which the Dispatcher function server runs.
- `<ID>` and `<password>` are the credentials for the submitted job. This ID must be one that has access to the EJBROLEs created earlier.
- `<port>` is the Dispatcher function server HTTP port.

**For example:**

```
scheduler-host=wsc3.washington.ibm.com
submitter-userid=WBADMIN
submitter-password=*****
#debug=true
debug=false
scheduler-port=32067
```

**Note:** The password here is shown as a string of asterisks, but in reality it was typed in the clear in the file. It may be encoded using the "PropFilePasswordEncoder" utility of WAS<sup>57</sup>. When encoded, the property looks *something* like this:

```
submitter-password={xor}OTAwPT4tbj4=
```

- In the directory indicated by `<path>` in the previous step, create the `wsgrid.props` file and populate it, in EBCDIC, with the following:

```
repository-job=SimpleCI
substitution-prop.calctime=15
substitution-prop.outfile=/<path>/SimpleCI.out
```

You should still have the SimpleCI xJCL saved in the job repository from when you did that back on page 20. If not, go back and save the xJCL and match the name in the props file to the name you gave it in the repository<sup>58</sup>.

The value of `<path>` represents where the output file will go.

- Submit the JCL you created above.
- That job will stay active for the duration of the job in WebSphere Java Batch. Go to the JMC and see your job get submitted, execute and end.

**Summary of native WSGRID and default messaging**

The story here is very similar to that for WSGRID using MQ and BINDINGS mode, except the messaging provider was the built-in messaging function of WAS, and the client was a shell script. In our example we packaged the shell script invocation in JCL.

Again, we did not actually integrate it with an enterprise scheduler. The key thing to keep in mind is this: it was a JCL job, which enterprise schedulers integrate with quite well; and the WSGrid JCL job *stays up for the life of the Java Batch job back in WebSphere*. That's the key -- to the enterprise scheduler the JCL containing WSGrid.sh is the batch job, with the actual batch job operating in WebSphere Java Batch.

<sup>57</sup> Search the InfoCenter for that string.

<sup>58</sup> It is *not* required that WSGrid jobs be submitted from the repository. An alternative is to point to an xJCL file directly from the job JCL that runs the `WSGrid.sh` shell script. Replace the second parameter -- in our example above it's the pointer to the props file -- with a pointer to a saved xJCL file for the job. If you wish to pass in substitution properties, then use the method illustrated here with the props file.

## When Compute Grid 8 on WAS V7 or V8 is the Environment

The first section of this document focused on WAS z/OS Version 8.5, which included the WebSphere Java Batch function as part of the product. That makes things somewhat simple: there's no need to add the Batch function to WAS z/OS; it's included.

But with WAS z/OS V7 or V8 that's not the case. The WebSphere Java Batch function is *not* included in those versions. It is a separate product called IBM WebSphere Compute Grid. That product must be *added*<sup>59</sup> to the WAS z/OS runtime.

### Configuration similar ... function identical

It turns out the configuration steps needed to enable Compute Grid in V7 or V8 are very similar to what we saw earlier with WAS V8.5:

Create database tables	Same process <sup>60</sup>
Configure JDBC providers and data sources	Same process
Create EJBROLE definitions	Same process
Configure Job Scheduler server	Same process
Configure Joe Endpoint server	Same process

Further, the WebSphere Java Batch function is *identical* between Compute Grid V8 and what's included in WAS V8.5.

### Creating the runtime with CG is different between V7 and V8

How this is done is a function of what level of WAS z/OS you are using.

WAS z/OS V7	<ul style="list-style-type: none"> <li>WAS z/OS V7 itself is installed using SMP/E.</li> <li>Compute Grid V8 is installed using Installation Manager (IM).</li> <li>The WAS z/OS runtime is constructed using WebSphere Customization Tool (WCT).</li> <li>The WCT is then used to create a job that <i>augments</i> the Compute Grid function to the existing WAS z/OS runtime<sup>61</sup>.</li> <li>When the WAS z/OS runtime is restarted, the Compute Grid function is available for configuration using the steps outlined in this document.</li> </ul>
WAS z/OS V8	<ul style="list-style-type: none"> <li>WAS z/OS V8 itself is installed using Installation Manager (IM)<sup>62</sup></li> <li>Compute Grid V8 is installed using IM, <i>and is installed into the same install image<sup>63</sup> as WAS z/OS V8</i>. One file system with WAS z/OS and the Compute Grid "stack product" combined<sup>64</sup>.</li> <li>The WAS z/OS runtime is constructed using WCT, but because WAS and CG are combined the WCT offers an option to create WAS+CG in one set of jobs. There is no need to build the WAS z/OS runtime and then augment separately. The jobs used to create the WAS z/OS runtime will include CG into that runtime.</li> <li>When the WAS z/OS runtime is restarted, the Compute Grid function is available for configuration using the steps outlined in this document.</li> </ul>

This document will not go into details on using Installation Manager<sup>65</sup> or how to use WCT. Instead, we'll assume the WAS z/OS runtime is constructed -- at V7 or V8 -- with the CG function available as outlined above.

<sup>59</sup> The term you will see used is "augmented." The Compute Grid product is *augmented* to the WAS z/OS configuration to provide the WebSphere Java Batch function.

<sup>60</sup> Though the DB2 definitions are provided in a different location in the file system.

<sup>61</sup> This involves the creation of symlinks to the CG code as well as updates to quite a few XML files.

<sup>62</sup> Though the initial IM *repository* is installed using SMP/E. Then IM is used against that SMP/E installed repository to create the WAS V8 install image.

<sup>63</sup> By "install image" we mean the output of IM, which is a file system that represents the IBM product files.

<sup>64</sup> See "Example of IM used to install WAS 8 and CG 8 into same install image" on page 42 for an example of IM commands used to do this.

<sup>65</sup> See <http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102014>

## Configuring the Compute Grid Function

### Create the LRS and LREE database tables in DB2

**Note:** For WAS V8.5, the discussion for this topic started on page 5.

The location of the file with the DB2 definitions is<sup>66</sup>:

```
/<was_mount>/AppServer/stack_products/WCG/longRunning/SPFLRS
```

Work with your DBA to customize those definitions to your environment for the storage group, table space, database name and schema values. There is a list of variables at the top of the definition table that may be used to guide you in global changes within the file.

**Note:** We spotted one small problem: the line that creates the unique index on table SUBMITTEDJOBS extends past column 72. That results in the "S" on the table name being truncated. That creates a mismatch between the table name specified for the index with the table name created just a few lines earlier.

The fix is easy enough:

- Locate the create unique index line for SUBMITTEDJOBS table
- Split that line so ON LRSSHEMA.SUBMITTEDJOBS is on the next line
- The resulting create unique index line would then look like this:

```
CREATE UNIQUE INDEX LRSSHEMA.SUBMITTEDJOBSIDX
ON LRSSHEMA.SUBMITTEDJOBS
(JOBID, SUBMITTEDJOBID)
USING STOGROUP LRSCHSG;
```

### Create the JDBC provider and data sources

**Note:** For WAS V8.5, the discussion for this topic started on page 6.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

The important thing is to delete any provider or data source scoped at the cell level that looks to be related to Compute Grid but using the Derby database. Look for data sources with `jdbc/lrs` or `jdbc/lree` that reference the Derby provider. Delete those provider and data source definitions and recreate using JDBC Type 4 XA to DB2.

### Create the EJBROLE definitions to secure the Job Management Console (JMC)

**Note:** For WAS V8.5, the discussion for this topic started on page 12.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

### Turn on application security and validate secure access to JMC

**Note:** For WAS V8.5, the discussion for this topic started on page 13.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

### Configure the Job Dispatching function in a server

**Note:** For WAS V8.5, the discussion for this topic started on page 13.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

### Deploy sample application into a server and validate endpoint configuration

**Note:** For WAS V8.5, the discussion for this topic started on page 15.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

Compute Grid V8 comes with the sample application EAR files and xJCL, unlike V8.5 which supplied those from a website. The V8.5 sample applications will work with Compute Grid V8 on top of either WAS V7 or WAS V8.

<sup>66</sup> This is *different* from where it's located in WAS V8.5, which as `/AppServer/util/Batch`. When WAS z/OS V7 or V8 with Compute Grid V8 augmented on top, do **not** use the definitions found at `/AppServer/util/Batch`. Use location and `SPFLRS` indicated above.

However, if you'd like to use the samples supplied in the file system, then you may get those from the following locations:

EAR files:	<code>/&lt;was_mount&gt;/AppServer/stack_products/WCG/installableApps</code>
xJCL files:	<code>/&lt;was_mount&gt;/AppServer/stack_products/WCG/longRunning</code>

### Prepare xJCL, submit, and validate successful execution

**Note:** For WAS V8.5, the discussion for this topic started on page 17.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

### Storing xJCL in the Job Repository and submitting from there

**Note:** For WAS V8.5, the discussion for this topic started on page 20.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

### Running the XDCGIVT sample

**Note:** For WAS V8.5, the discussion for this topic started on page 21.

This is the same process for WAS V7 or V8 as it was for WAS V8.5.

### Configuring the WSGRID interfaces

**Note:** For WAS V8.5, the discussion for this topic started on page 25.

This process is the same, but the location of the scripts to implement the WSGRID interfaces is different. With Compute Grid V8 those scripts (`installWSGridMQ.py`, `unpackWSGRID`, `wsgridConfig.py` and `WSGrid.sh`) may be found at this location:

`/<WAS_mount_point>/AppServer/stack_products/WCG/bin`

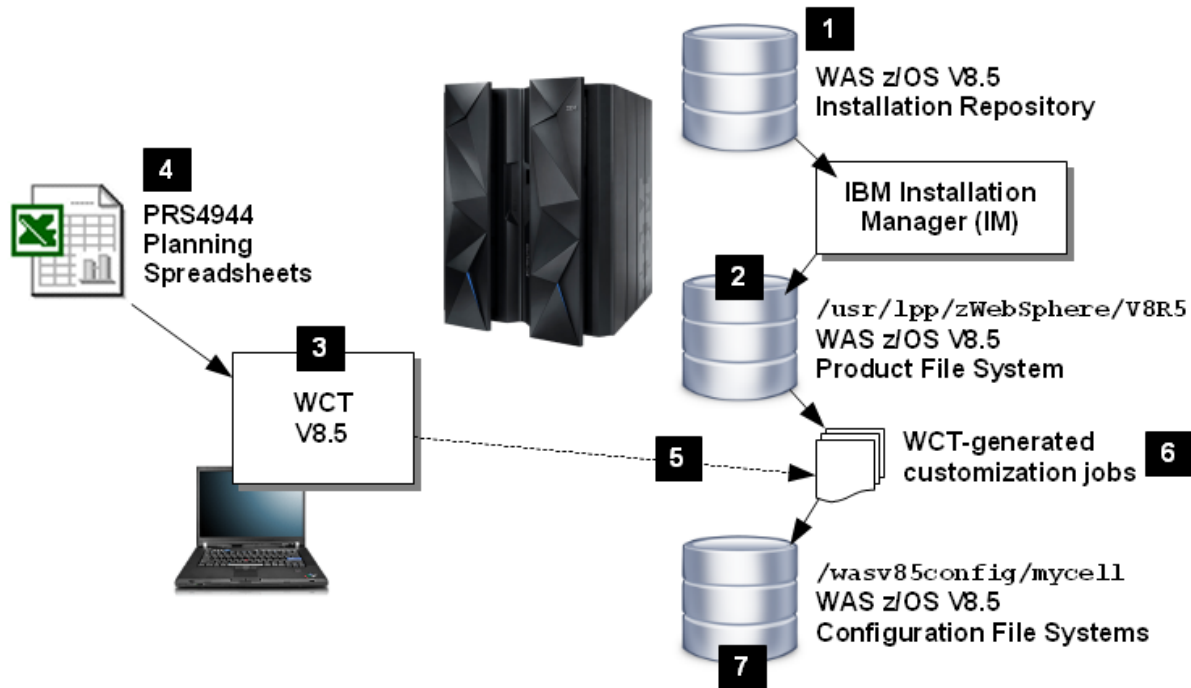


## Miscellaneous Information

### *How to create and customize a WAS z/OS runtime environment*

Earlier we stated that a valid WAS z/OS V8.5 runtime environment was assumed to be in place. That was so this document could focus on enabling the Java Batch function.

A high-level picture of the process would look like this:



With the following notes explaining each of the numbered blocks in the picture:

- 1 It starts with an Installation Manager "Repository," which is a package of product files created by IBM. This Repository is understood by Installation Manager and is used as the source for installing the product on the target z/OS system. For WAS z/OS this IM Repository is initially installed using SMP/E to lay down the Repository file system.
- 2 Then Installation Manager is used to install WAS z/OS V8.5. This implies the creation of a file system containing thousands of directories and files. When complete, this file system represents the WAS z/OS product. This is *not* your server runtime; this is the product binaries from which your servers will operate.
- 3 Your server runtime is created by running a sequence of z/OS customization batch jobs. The batch jobs are created with a workstation tool called Workstation Customization Tools (WCT). WCT is a graphical tool that takes your customization input and generates the jobs needed to create the runtime on z/OS.
- 4 You could manually input all the customization information WCT requires, but an easier method is to use the "Planning Spreadsheets." The spreadsheets capture a very few pieces of customization information from you, then it generates an input file for WCT. In WCT you simply import the input file and generate the customized jobs. It makes things much simpler, and more important it helps enforce consistency between all the names and values that make up a runtime.
- 5 The WCT has a facility to upload the customized jobs to your target z/OS system.
- 6 The customized jobs end up on z/OS as a series of members in two PDS data sets: one is a FB 80 data set with JCL; the other is a VB data set that holds shell scripts and other non-JCL files. You execute the jobs in order according to the instructions provided. Those jobs do things such as: allocate and mount the file systems; create RACF profiles; copy JCL procs into proclib; and copy all the directories and XML files that make up your WAS z/OS runtime configuration.

- 7 The result is a file system with a set of directories and files that represents your server runtime environment. This file system does not contain the actual product code. Rather, this file system contains XML files that describes your runtime, with symbolic links back to the Product File system.

That seems like a lot of steps, and in some ways it is. But for those who have done this a few times the process is quite easy to accomplish.

The following Techdoc provides a guide to the various other Techdocs where we describe much of this process in greater detail. The PDF in this Techdoc is organized by activity.

<http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102205>

### **wsgridConfig.py command example**

Here's an example of the command we ran in our test environment<sup>67</sup>:

```
./wsadmin.sh -lang jython -conntype SOAP -host wsc3.washington.ibm.com
  -port 32002 -user wbadmin -password <password> -f wsgridConfig.py
  -install -node wbnoddec -server wbsr01c -providers
  "wsc3.washington.ibm.com,32073;wsc3.washington.ibm.com,32074"
```

### **Example of IM used to install WAS 8 and CG 8 into same install image**

What follows is an example of the IM commands we used to install WAS 8.0.0.6 and CG 8.0.0.2 into the *same target install image*<sup>68</sup>. What you see is that the two invocations of IM -- one to install WAS z/OS and the other to install CG -- target the same installation directory.

#### **Install of WAS z/OS V8.0.0.6:**

```
BPXBATCH SH /Service/InstallationManager/bin/eclipse/tools/imcl install
  com.ibm.websphere.zOS.v80,core.feature,ejbdeploy,thinclient ,embeddablecontainer,samples
  -installationDirectory /Service/usr/lpp/zWebSphereCG/V8R0FP02
  -installFixes recommended
  -sharedResourcesDirectory /Service/InstallationManager/sharedResources
  -repositories <URL to repository>,
  http://www.ibm.com/software/repositorymanager/com.ibm.websphere.zOS.v80
  -preferences com.ibm.cic.common.core.preferences.preserveDownloadedArtifacts=false
  -acceptLicense -keyring /u/user1/imkeyring
:
Installed com.ibm.websphere.zOS.v80_8.0.6.20130328_1654 to the
  /Service/usr/lpp/zWebSphereCG/V8R0FP02 directory.
```

#### **Install of CG 8.0.0.2:**

```
BPXBATCH SH /Service/InstallationManager/bin/eclipse/tools/imcl install
  com.ibm.websphere.WCG.zOS.was8.v80
  -installationDirectory /Service/usr/lpp/zWebSphereCG/V8R0FP02
  -installFixes recommended
  -sharedResourcesDirectory /Service/InstallationManager/sharedResources
  -repositories <URL to repository>
  -preferences com.ibm.cic.common.core.preferences.preserveDownloadedArtifacts=false
  -acceptLicense -keyring /u/user1/imkeyring
:
Installed com.ibm.websphere.WCG.zOS.was8.v80_8.0.2.20120731_1353 to the
  /Service/usr/lpp/zWebSphereCG/V8R0FP02 directory.
```

<sup>67</sup> We did not code `-filestoreroot`. The default of `/tmp` was okay for our test system. It might not be for yours. So check.

<sup>68</sup> We are not showing the commands to create and mount the file system to hold the install elements.

---

## Documentation References

### General guide to WAS z/OS documentation:

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102205>

Consider that Techdoc to be a "table of contents" into all the other online documentation for WAS z/OS in general.

### WebSphere Java Batch Techdoc:

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP101783>

That is the general WebSphere Java Batch Techdoc page. This document along with many other PDF files -- from high level brochures to detailed white papers -- are included there.

### Compute Grid Wildfire Workshop material:

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/PRS4644>

The section on "Programming Framework - Develop your first Java batch job for WCG" includes the hands-on labs for that portion of the workshop. They provide a step-by-step illustration of how WebSphere Java Batch applications are developed using the support included with IBM Rational Application Developer (RAD).

### Installation Manager on z/OS

<http://www.ibm.com/support/techdocs/atmastr.nsf/WebIndex/WP102014>

This document provides a "cookbook" (step-by-step) approach to installing and using IM on z/OS for managing WAS z/OS installations.

### "Batch Modernization on z/OS" Redbook

<http://www.redbooks.ibm.com/redbooks/pdfs/sg247779.pdf>

This is a fairly comprehensive Redbook on the broad topic of batch modernization.

---

## Document Change History

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

---

<i>June 1, 2013</i>	Original document
---------------------	-------------------

---

End of Document