WebSphere Application Server z/OS V8.5

Hands-on Labs

Lab Version Date: October 24, 2013

Table of Contents

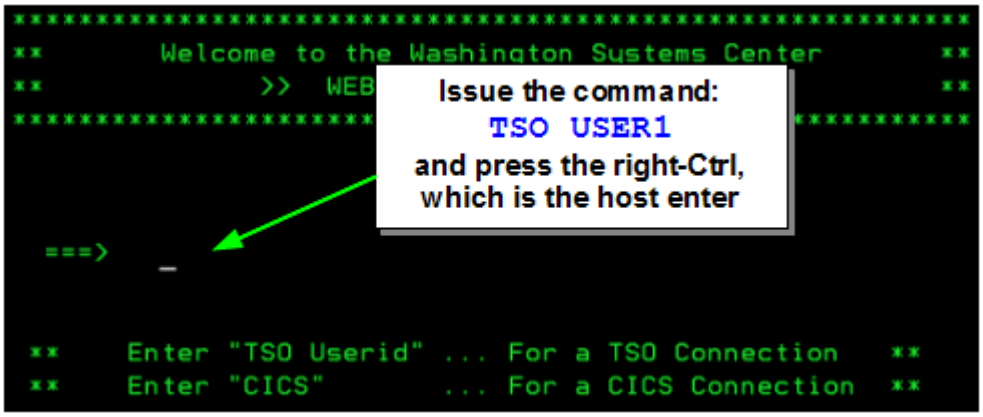
Unit 2 Lab - Administrative Model....3
Start a portion of your WAS z/OS runtime environment....3
Perform simple Administrative Console activities....3
Configuration File System Exploration....6
WSADMIN....7
HPEL....13
Output Enhancements with PM74923....19
Unit 3 Lab - Server Models....22
Multiple servants....22
WLM "round robin" distribution....25
WLM classification....26
Granular RAS....29
Environment cleanup....32
Liberty Profile - first steps....33
Liberty Profile - z/OS Started Task....35
Liberty Profile - JDBC Type 2 using Angel Process....36
Unit 4 Lab - Accessing z/OS Data....38
JDBC....38
CICS....54
MQ....60
Unit 6 Lab - WebSphere Optimized Local Adapters....64
Setup of WOLA in WAS runtime....64
Setup of WOLA in CICS region CICSX....66
Restart the environment and validate....67
Start Link Server Task, Register into WAS and perform basic tests....68
Inbound: CICS OLAUTIL application to WAS....71
Inbound: batch applications to WAS....72
Outbound: WAS to CICS using Link Server Task....76
Outbound: WAS to CICS with "alternative JNDI failover"....77
Answers, Hints and Tips....82
Brief tutorial on 3270 and MVS usage....82
System inventory answers....84
Configuration file system lab answers....85
WSADMIN SuperSnoop Installation Mini-Quiz Answers....85
Example of Test Connection Failure when Scope=Node....86
Using the WS-FTP client....86

Unit 2 Lab - Administrative Model

Start a portion of your WAS z/OS runtime environment

Note: We've built the WAS z/OS runtime environment for you. The process for doing that is the same it's been for several releases now. As you start this first lab you need to start the Deployment Manager so you'll have access to the Administrative Console, and the Node Agent so changes you make can be synchronized.

- ☐ Open up a 3270 session using the "WG31" icon on your desktop.
- ☐ At the system logon panel, do the following:



- ☐ Then logon with the password supplied to you by the instructors.
- ☐ If you see *** simply press host enter to clear that screen and proceed.
- ☐ Issue the command **=SDSF.LOG**
- ☐ Now open the command extension (single forward slash -- " / ") and then issue the long command to start the Deployment Manager server:

  S Z9DCR,JOBNAME=Z9DMGR,ENV=Z9CELL.Z9DMNODE.Z9DMGR

  Notes:
  - See "Entering long commands" on page 82 for hints on entering this command.
  - Desktop "Lab Command Strings" has file of commands for cut/paste.

  The controller region will start first, then the servant will start automatically. A minute or two after the servant starts check its output for `BBOO0248I INITIALIZATION COMPLETE.`

- ☐ Start the Node Agent:

  S Z9ACRA,JOBNAME=Z9AGNTA,ENV=Z9CELL.Z9NODEA.Z9AGNTA

  This will be a controller only. A minute or two after you start this check its output for

  `ADMS0003I: The configuration synchronization completed successfully.`

Perform simple Administrative Console activities

Note: Some of this you may be familiar with. Consider it a "warm up" to later Admin Console work.

- ☐ Using whatever browser you prefer, go to:

  `http://wg31.washington.ibm.com:10005/ibm/console`

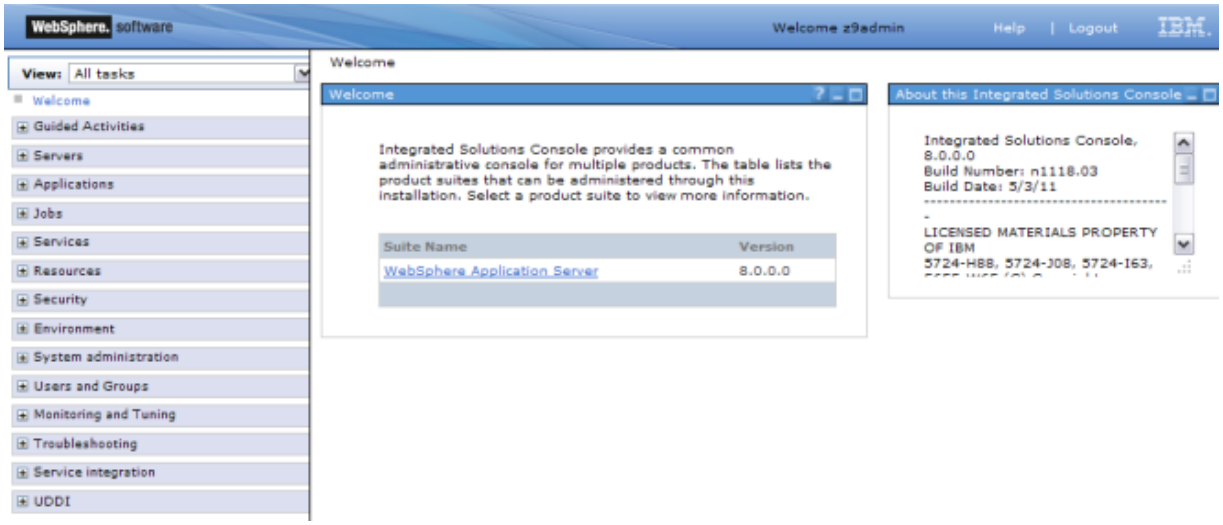
- ☐ Accept any security challenges you see[1].

1 Those are due to the fact the Deployment Manager's server certificate was "self-signed" (by RACF) and not by a "well known" authority. The browser is doing its job by alerting you to this. There's no risk in this case; accept and move on.

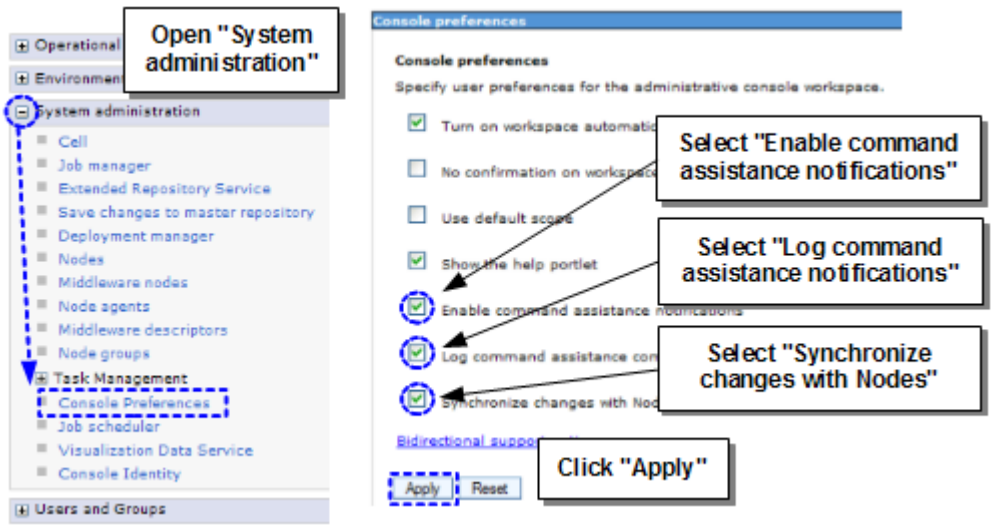
☐ You should see the following logon panel:

Logon using the ID **`z9admin`** and the password as supplied by the instructors.

You should now be at the main Administrative Console panel:

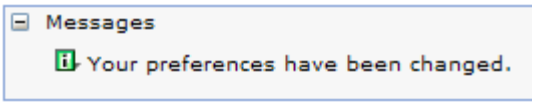


☐ Do the following[2]:



2 "Command assistance" will be used when you work with WSADMIN; "Synchronize" enables synchronization each time "save" is processed.

☐ You should see the following confirmation at the top of the page:



☐ Go under *System Administration* ➔ *Nodes*. Take inventory[3] of what you see:

Node Name	*What type of node? (See Below)*	*Synchonization Status?*
	☐ Deployment Manager ☐ Managed Node	☐ Green circle synchonized ☐ Red symbol not synchronized
	☐ Deployment Manager ☐ Managed Node	☐ Green circle synchonized ☐ Red symbol not synchronized
Note: The "type" can be inferred by name ("dm" = Deployment Manager), or by looking for a checkbox next to the node, which means it's a managed node. A checkbox implies the node can be synchronized, and that means the node has a Node Agent, which means it's a managed node.		

☐ Go under *Servers* ➔ *Server Types* ➔ *WebSphere Application Servers.* Take inventory of what you see:

Server Name	*Which node does it belong to?*	*Status?*
	☐ Deployment Manager node ☐ Managed Node	☐ Green arrow started ☐ Red X not started

☐ Click on the link representing the first server in the list and then *Server Infrastructure* ➔ *Java and Process Management* ➔ *Server instance.* Take inventory:

Multiple Instances Enabled	☐ Yes ☐ No
Minimum Number of Instances	☐ 1 ☐ 2 ☐ other: _____
Maximum Number of Instances	☐ 1 ☐ 2 ☐ other: _____

☐ Go under *Resources* ➔ *JDBC* ➔ *JDBC Providers.* Is there anything there that suggests the IBM DB2 JDBC driver is installed?

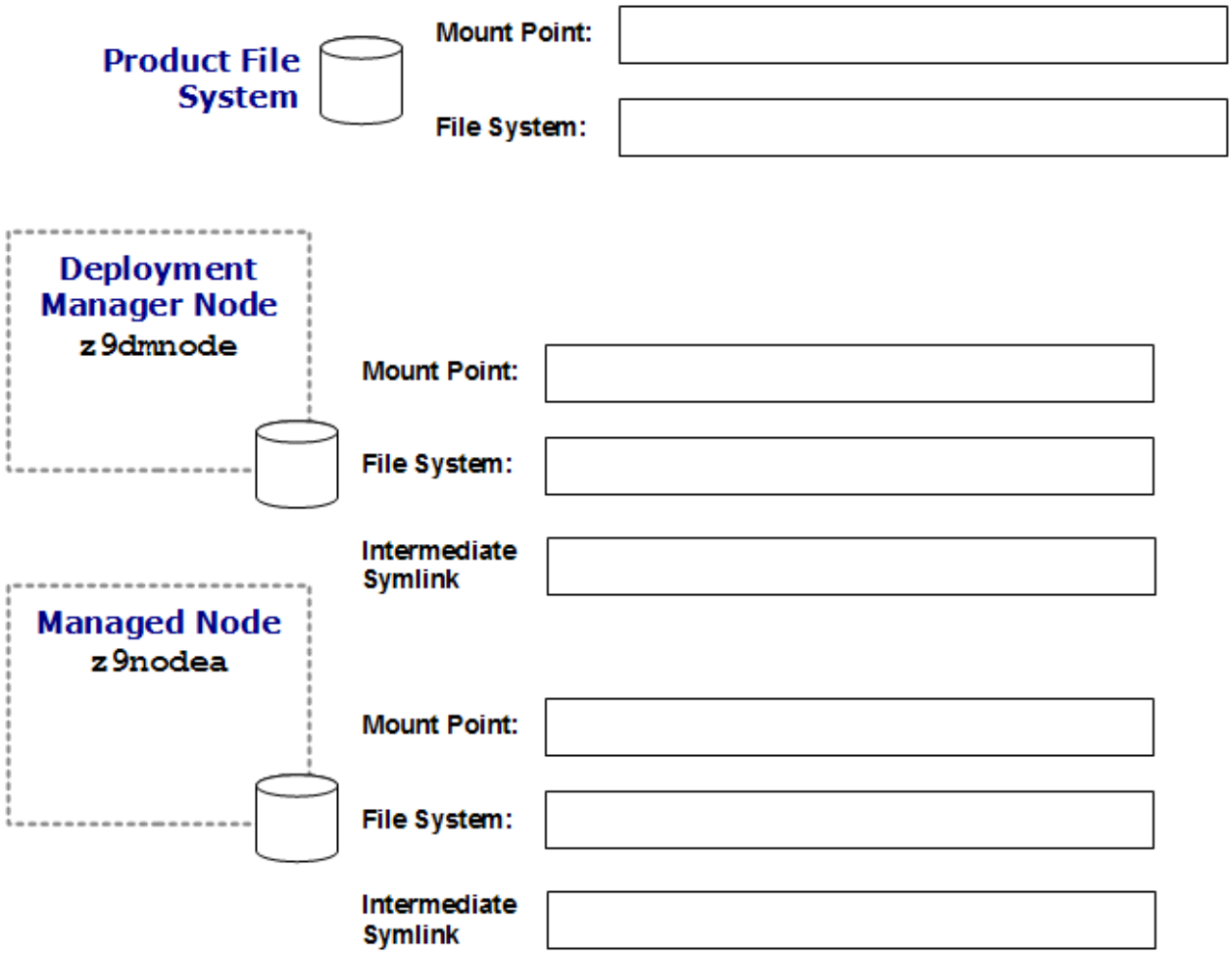
☐ Go under *Resources* ➔ *Resource Adapters* ➔ *Resource adapters.* Is there anything there that suggests either the CICS JCA resource adapter or the WOLA resource adapter is installed?

3 Answers to the "inventory" questions may be found under "System inventory answers" on page 84.

Configuration File System Exploration

Note: In this lab you'll simply do some guided exploration of the file systems and the symlink structures, just to familiarize you with the runtime.

Using the following picture, fill in the requested information using the guided instructions that follow[4]:



- ☐ Using the TeraTerm or PuTTY icon on the desktop, telnet into the **`wg31`** system. Logon as `USER1`. The instructors will provide the password.
- ☐ Change directories: **`cd /wasv85config`**, then issue **`ls -al`** to see the sub-directories. The sub-directory is the cell *long* name. How many cells do you see?
- ☐ Change directories to the **`z9cell`** directory, then issue **`ls -al`**. You should see "home" and two directories that correspond to the two nodes in the cell.
- ☐ Issue the command **`df | grep /wasv85config/z9cell`** to show the two file systems that are mounted for the cell's two nodes. Write the values in the picture above.
- ☐ Issue the command **`df | grep SBBOHFS`** to show the file systems mounted that hold the WAS z/OS product code. You should see several. Locate the one whose name suggests Version 8.5 Fixpack 2. Write the mount point and file system name for the V85 FP02 product in the picture above.

4 See "Configuration file system lab answers" on page 85 for this picture filled in with the values you should discover in this exercise.

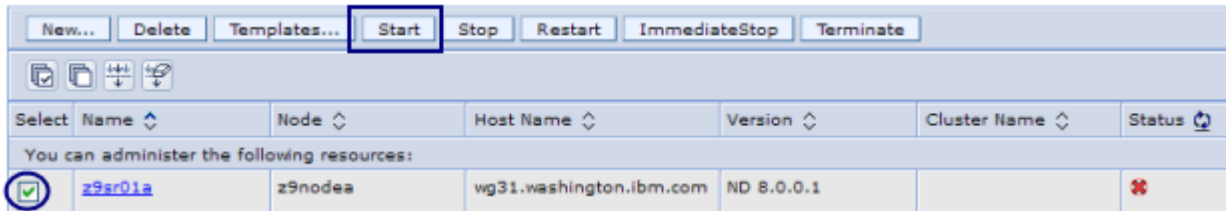
- ☐ Change directories to the **`/wasv85config/z9cell/z9dmnode`** directory. Issue the command **`ls -al`** to show the sub-directories *and symlinks* in this directory. The symlink named "wasInstall" is the intermediate symlink. Write that value in the picture above. Note where it points.
- ☐ Change directories to **`/usr/lpp/zWebSphere/V8R5FP02`**, then issue the command **`pwd`**. What does UNIX tell you is the present working directory? Does it match the mount point for the WAS V85 product file system?

  Note: The actual mount point is `/shared/zWebSphere/V8R5FP02`. That's a vestige of where the z/OS system for the workshop came from, which was a Sysplex with shared file systems. Your workshop system is a monoplex. `/usr/lpp/zWebSphere` is really a symlink.

- ☐ Change directories to **`/wasv85config/z9cell/z9nodea`** and repeat the previous two steps. Note the intermediate symlink and where it points.
- ☐ Close the TeraTerm (or PuTTY) session by typing **`exit`** and then enter.

WSADMIN

- ☐ In the Admin Console go to Servers ➔ Server Types ➔ WebSphere application servers. Then start the `z9sr01a` server:



  Wait a minute or two until the red X becomes a green arrow.

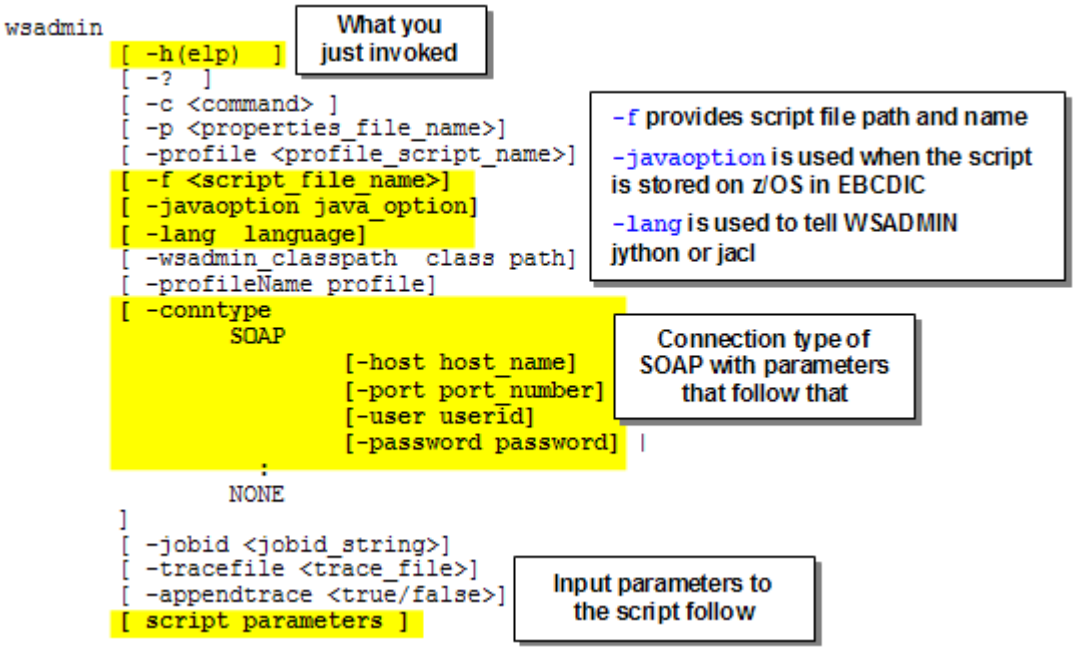
- ☐ Open up a TeraTerm or Putty session and connect to **`wg31`**. Log on as **`z9admin`** with the password supplied by the instructors.
- ☐ Change directories to:

  `/wasv85config/z9cell/z9dmnode/DeploymentManager/profiles/default/bin`

- ☐ Invoke the `wsadmin.sh` client shell script with `-help`:

  `./wsadmin.sh -help`

  You should get back two or so pages of help output. Take note of what our focus will be:



- ☐ In your Admin Console, go to *System Administration* ➔ *Deployment manager*, then off to the right locate *Ports* and click on that. Locate the `SOAP_CONNECTOR_ADDRESS` port value and write it down here:

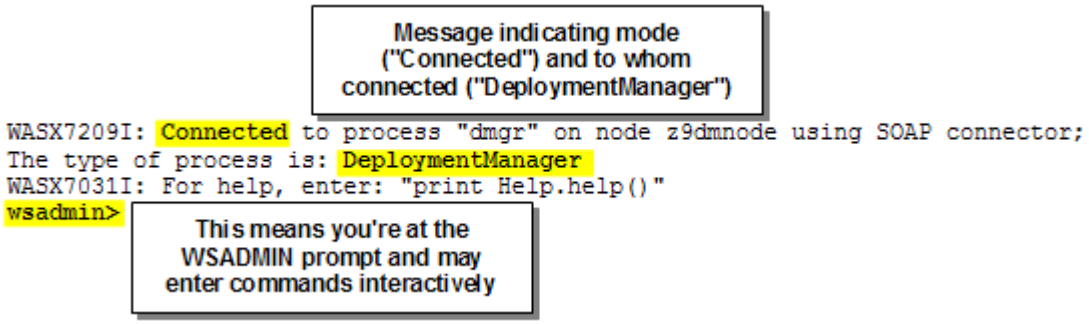
`SOAP_CONNECTOR_ADDRESS`	

- ☐ Go back to your TeraTerm/PuTTY session and enter the following command as one long line[5].

```
./wsadmin.sh -lang jython -conntype SOAP -host wg31.washington.ibm.com
                                  -port 10002 -user z9admin -password xxxxxxx
```

Where `xxxxxxx` is the password for the `z9admin` userid supplied by the instructors.

The first time you run this you'll see a bunch of `*sys-package-mgr*` messages, then you should see something like this:



- ☐ Issue the command **`print Help.help()`** and note the response you get, in particular help for the four primary command objects:

```
   :
AdminControl        Returns general help text for the AdminControl object
AdminConfig         Returns general help text for the AdminConfig object
AdminApp            Returns general help text for the AdminApp object
AdminTask           Returns general help text for the AdminTask object
   :
```

5 It is broken here so it'll fit. See "Command Lab Strings" file on desktop for cut-and-paste example of this command.

- ☐ Issue the command **`print AdminApp.help()`** and note the options that exist for that command object. In particular, look for `install`, `uninstall` and `list`. That's what the script we'll provide makes use of.
- ☐ Issue the command **`print AdminApp.help('install')`** and note what comes back to you. This is displaying the help for the `install` method of AdminApp:

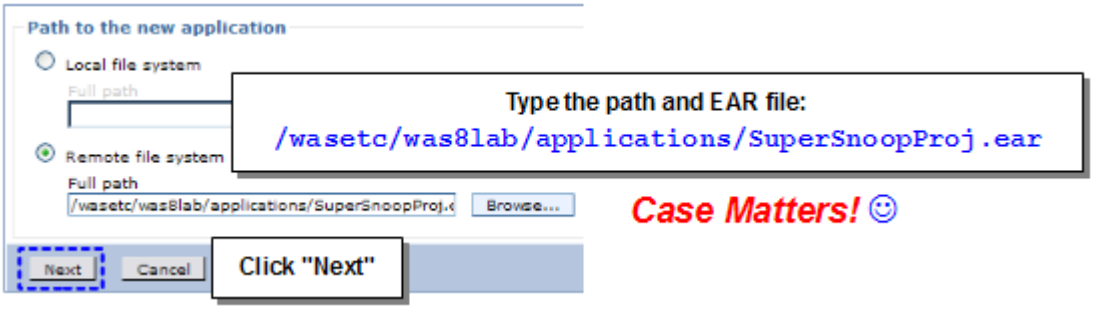
```
wsadmin>print AdminApp.help('install')
WASX7096I: Method: install

          Arguments: filename, options

          Descriptio  nstalls th  application...
```

Path and file name of the EAR file to install

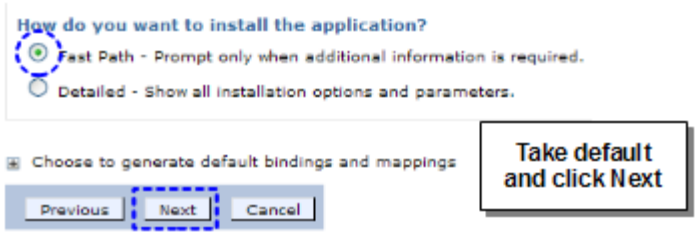
Options for the install of the application

Note: This is where things get a bit more complicated. The "options" that apply are a function of what's in the application. Simple applications have fewer options; more complex applications more options. What those options are and what the syntax is can be challenging. Thankfully there's a "command assistant" function of the Admin Console to help.

- ☐ At the `wsadmin>` prompt, issue command **`exit`**. This will return you to the UNIX prompt.
- ☐ Earlier we had you set a console preference of "Enable command assistance notifications." Now you'll make use of that. Go to *Applications* ➔ *New Application* ➔ *New Enterprise Application*.
- ☐ Do the following:

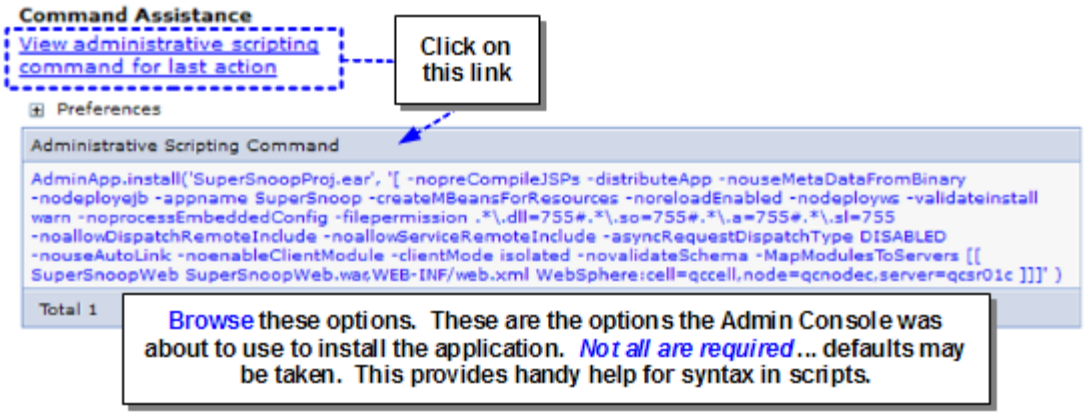


- ☐ Then:



- ☐ Then:

- ☐ Off to the right you'll see a link titled "Command Assistance." Do the following:

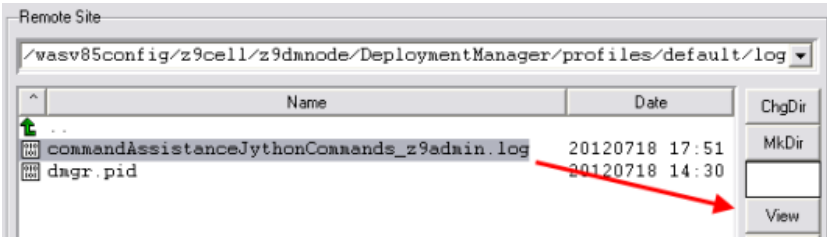


> **Note:** The point of that was to show you that with "Command Assistance" you can get a much better sense for the syntax and layout of a WSADMIN command. The Admin Console is telling you what it is about to use under the covers. This works for most configuration activities, not just application installations.

- ☐ Close the Administrative Scripting Commands window.
- ☐ Open the WS-FTP client[6] on the desktop, select the "wg31" profile and click "OK." Then navigate the host side (right side) to:

  `/wasv85config/z9cell/z9dmnode/DeploymentManager/profiles/default/logs/dmgr`

- ☐ You should see the file `commandAssistanceJythonCommands_z9admin.log`. Select that file, make sure transfer mode radio button is set at "Binary" and then click on the "View" button:



- ☐ Scroll to the bottom of the file view that appears. You should see a very long AdminApp.install command. This is exactly the same as the console window command you saw a moment ago. This illustrates an alternative way to get at scripting commands.
- ☐ Close the file view and exit WS-FTP.
- ☐ *Cancel the application install*. You'll install SuperSnoop with WSADMIN instead.
- ☐ Open a Notepad session[7] -- the command you're about to issue is very long and it's best to compose it ahead of time rather than in the TeraTerm/PuTTY window. Compose the following command as *one long line:*

```
./wsadmin.sh -lang jython -conntype SOAP -host wg31.washington.ibm.com
                                  -port 10002 -user z9admin -password xxxxxxx
                              -f /wasetc/was8lab/scripts/installSuperSnoop.py
```

> **Note:** Where `xxxxxxx` is the password for the `z9admin` ID. This is the same invocation command as before, but this time with the `-f` switch pointing to the Jython script to execute.

6 See "Using the WS-FTP client" on page 86 if you're unfamiliar with the use of this tool.
7 Or copy/paste the command from the "Lab Command Strings" file on your desktop.

- ☐ Once you have it composed to your satisfaction in Notepad, copy and paste it into the TeraTerm or PuTTY session, which should be at the UNIX prompt. Press **`Enter`** to process the command.
- ☐ The output is pretty chatty with various "ADMA" messages. Sift through those and find these key messages:

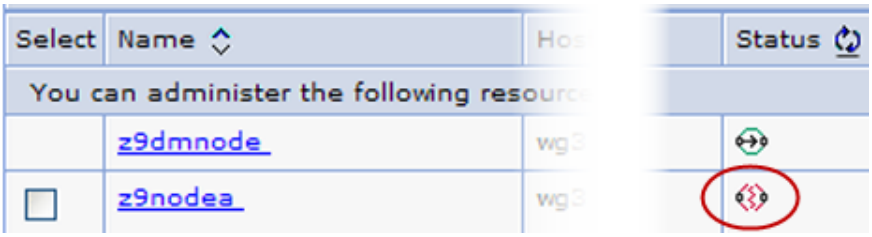
```
Installing application SuperSnoop  1
appopts = [-appname SuperSnoop -MapModulesToServers [[ SuperSnoopWeb
    SuperSnoopWeb.war,WEB-INF/web.xml
    WebSphere:cell=z9cell,node=z9nodea,server=z9sr01a ]]]   2
ADMA5013I: Application SuperSnoop installed successfully.   3
Application installed and saved but not synchronized or started  4
```

Notes:
1 - Simple echo back of application name hard-coded in script.
2 - Part of script that echos out the constructed options list.
3 - A WSADMIN system message indicating success.
4 - Script message including important information about what was not done.

- ☐ Run the *exact same command* again. This time take a look at the message output starting from right after WSADMIN begins ... you should see something like this amongst other messages[8]:

```
Found and uninstalling SuperSnoop
ADMA5017I: Uninstallation of SuperSnoop started.
```

- ☐ In your Admin Console[9] go to System *Administration* ➔ *Nodes*. You should see the status[10] of your node as "broken synchronization":



➔ *Check the box next to `z9nodea` and click on the "Synchronize" button.*

- ☐ Use the WS-FTP client on your desktop to connect to your host system and view the `/wasetc/was8lab/scripts/installSuperSnoop.py` file. View this file with the "binary" option selected. Take this mini-quiz[11]:

#	*Question*	*Your Answer*
Q1	What WSADMIN command object *and method* is used to **uninstall** SuperSnoop if it is found?	

8 The script first checks if the application is there. If so, it uninstalls it.
9 You may see the message "Your workspace has been auto-refreshed from the master configuration." This is because WSADMIN was making changes to the master configuration. The Admin Console is synching up with those changes.
10 The script does not attempt to synchronize nodes. That *is* possible with WSADMIN, just not used in *this* script.
11 See "WSADMIN SuperSnoop Installation Mini-Quiz Answers" on page 85 for the answers to this mini-quiz.

Q2	After the application has been uninstalled, what command object *and method* is used to save the changes?	
Q3	How many lines in the script are used to construct the `AdminApp.install()` option list?	
Q4	How many option parameters are in the application installation options list?	
Q5	What's the purpose of the `import sys` line at the top and the `splitlines()` used on `AdminApp.list()`?	

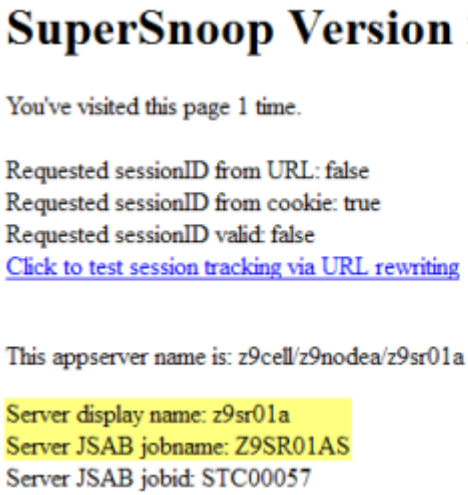
- ☐ In the Admin Console, go to the list of applications and start your newly installed SuperSnoop application. Wait for red X to turn to green arrow. To refresh, click little circular arrow icon.

  Note: If you don't see the SuperSnoop application then log off the Admin Console and log back in. That'll refresh the Console cache and pick up the changes made by WSADMIN.

- ☐ Invoke the application with this URL:

  `http://wg31.washington.ibm.com:10067/SuperSnoopWeb/SuperSnoop`

  You should see a very long browser page response with the top having something like this:



  Note the dipslay of your server *long* name but your servant *short* name as well.

Please Note: Now we're going to have you invoke a much more sophisticated script. This one will create a new server, remap the port values to fit the range conventions we use, create the virtual host alias entries and synchronize the changes to all the nodes

This is straight from the Mike Loos' TD105447 Techdoc at `ibm.com/support/techdocs`. It says V7 on that Techdoc title but the script works perfectly well *unchanged* with V85.

Our purpose for having you run this is to show how powerful a script may be. It also serves as an excellent reference when you go to craft other scripts.

- ☐ Step one is to take inventory of what you have for servers in your cell right now. In the Admin Console go to *Servers* ➔ *Server Types* ➔ *WebSphere application servers*. You should see only one server there.
- ☐ Click on that server link, then off to the right click on the "Ports" link. Note the ports for the first server ... they're in the `10060` - `10079` range. That means the next server starts at `10080`.

- ☐ In a Notepad session compose this very long command as one line, just as before. (If you still have the earlier Notepad session around modify that. Or use cut/paste file on desktop.)

```
./wsadmin.sh -lang jython -conntype SOAP -host wg31.washington.ibm.com
                                -port 10002 -user z9admin -password xxxxxxx
  -f /wasetc/was8lab/scripts/createNewServerv85.py z9sr02a z9nodea 10080
                             1                    2       3       4     5
```

Notes:
1 - The path to the file system location where we've provided the script file
2 - The script file from the TD105447 Techdoc.
3 - First parameter passed in -- the new server long name[12].
4 - Second parameter passed in -- the node long name in which the server will be created
5 - Third and last parameter passed in -- the starting port value for the range of ports

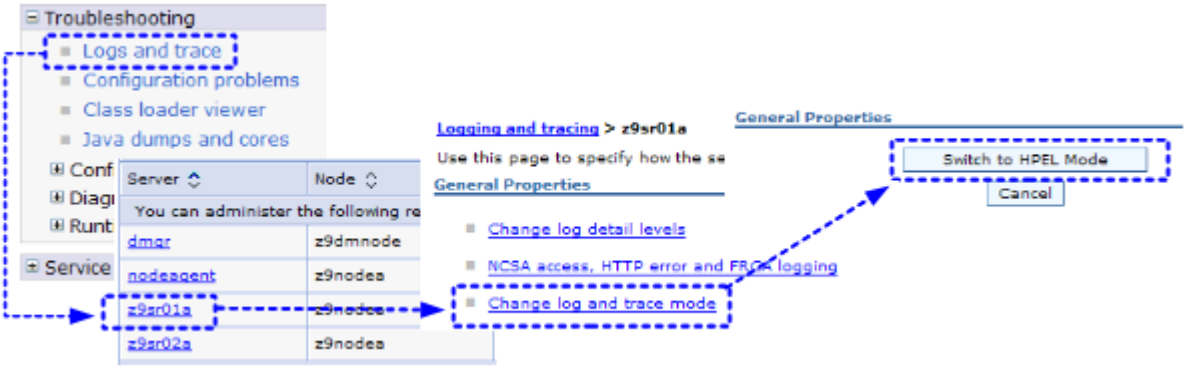
- ☐ Copy/paste that long command into your TeraTerm or PuTTY session and hit enter.
- ☐ Look for the message "All Done!" to indicate the server has been created.
- ☐ Go into the Admin Console and see that your node is already synchronized (*System Administration* ➔ *Nodes*)
- ☐ In the Admin Console go to *Servers* ➔ *Server Types* ➔ *WebSphere application servers*. You should see your new server.
- ☐ Go into the ports for the new server. You should see ports in the `10080 - 10099` range
- ☐ Go to *Servers* ➔ *Server Types* ➔ *WebSphere application servers* and start your new server.

Note: The new server did not require a new JCL start procedure. It used the same JCL start procedure as the first server. It did not require any additional RACF profiles because the RACF profiles created for the Z9 cell were sufficiently generic to allow all servers for the cell to use the same set of profiles.

- ☐ Shut down that second server. We don't need it any more.

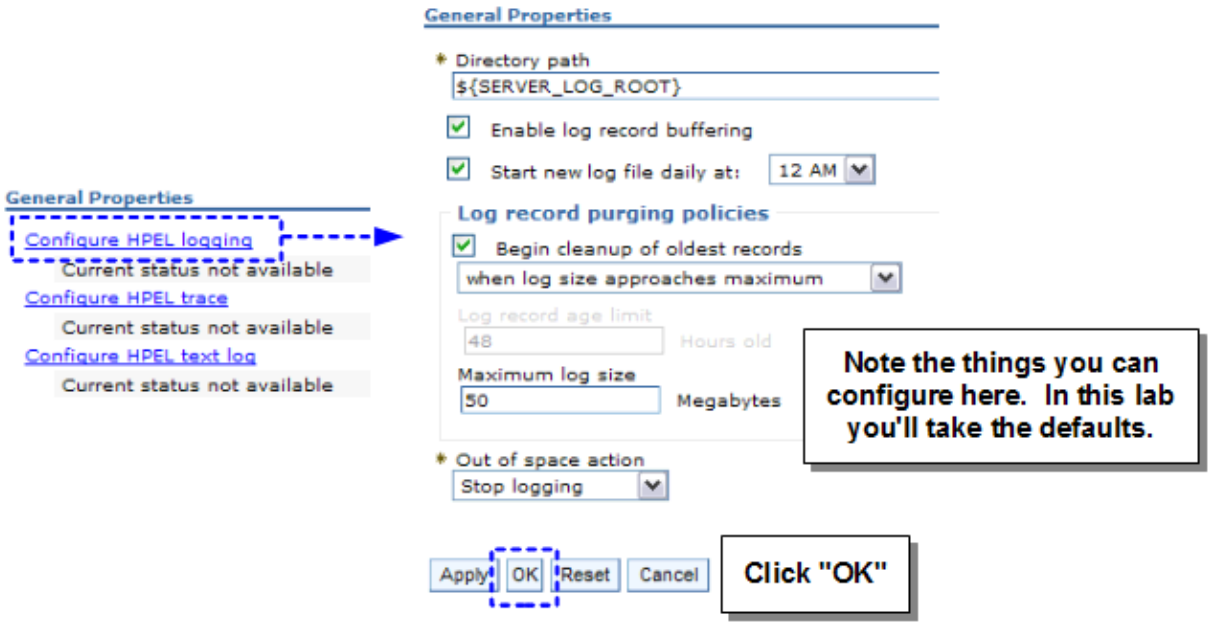
HPEL

- ☐ In the Admin Console, begin the process of configuring HPEL for the `z9sr01a` server:

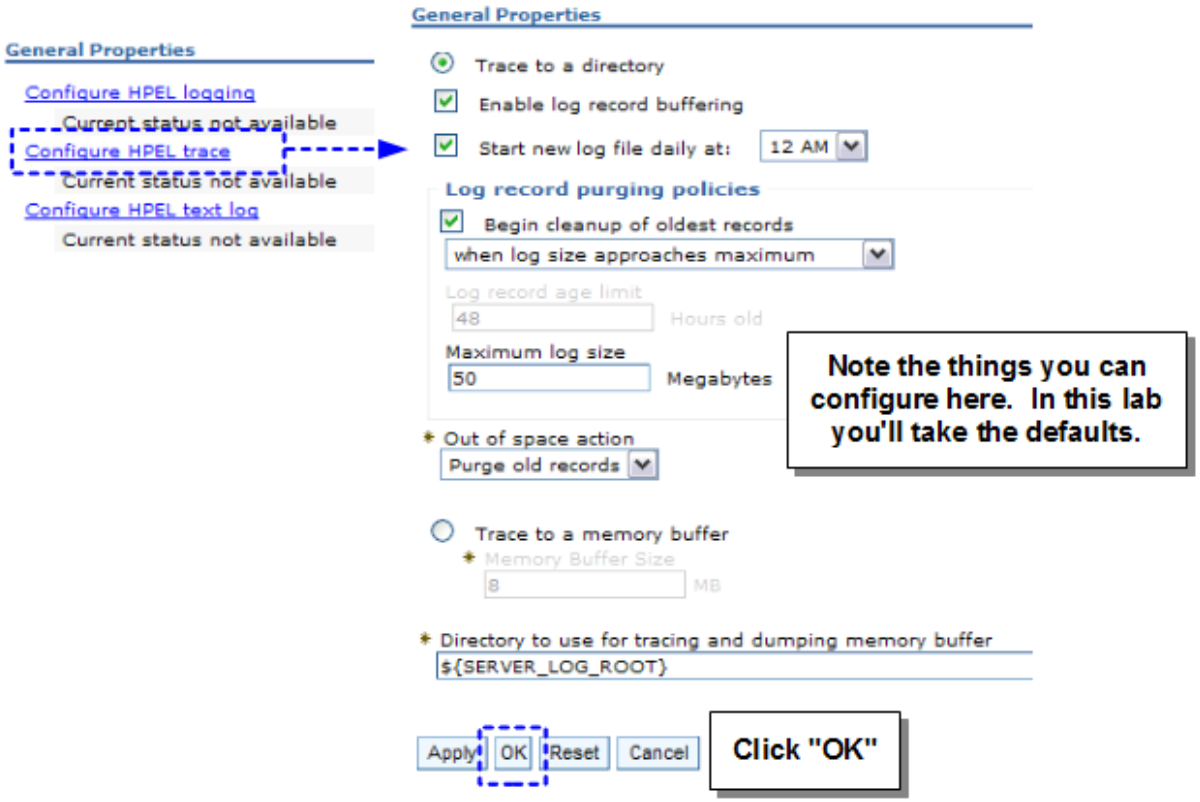


12 The name conforms to the naming standards of the cell. It is one up from `z9sr01a`.

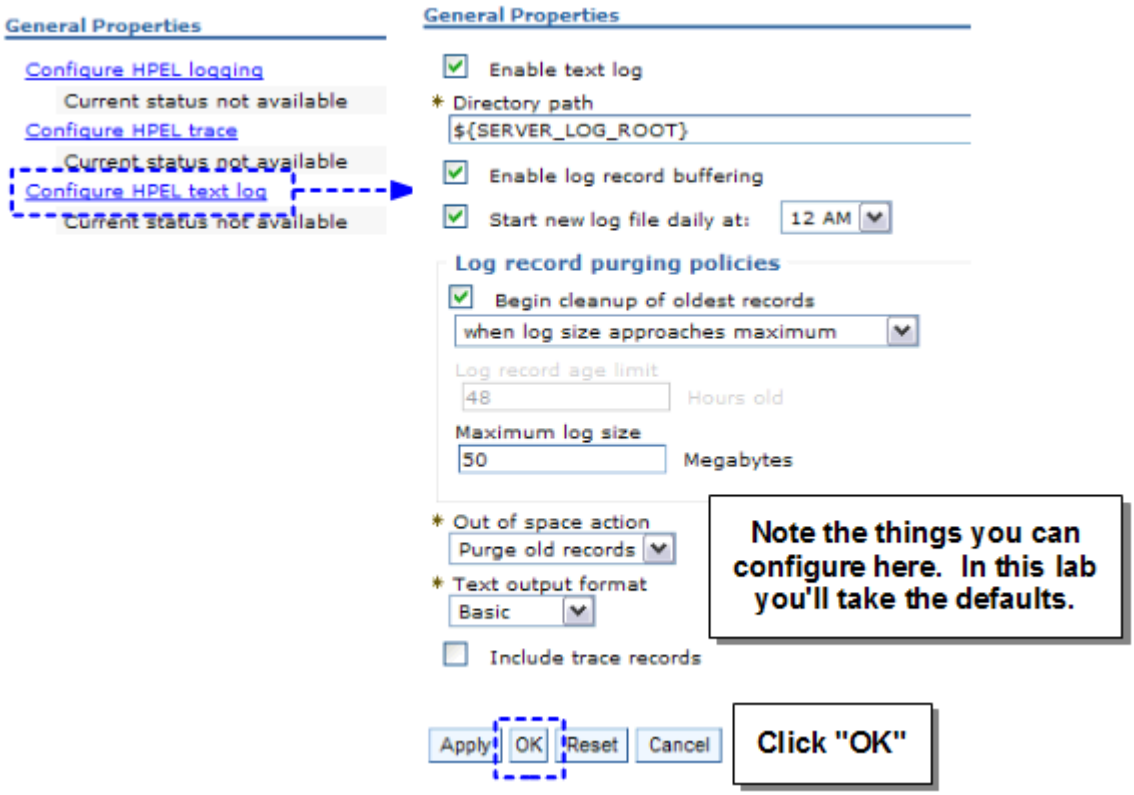
- Then take a look at the configurable options for *logging*:



- Next, take a look at the configurable options for *tracing*, which looks very similar to logging but gives you an option to trace to a memory buffer first if you wish.

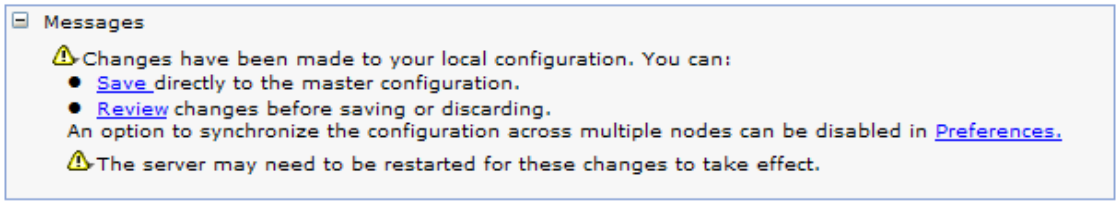


- Finally, take a look at the configurable options for the text log, which is a human-readable log you can view using tools such as TeraTerm or PuTTY and tail:



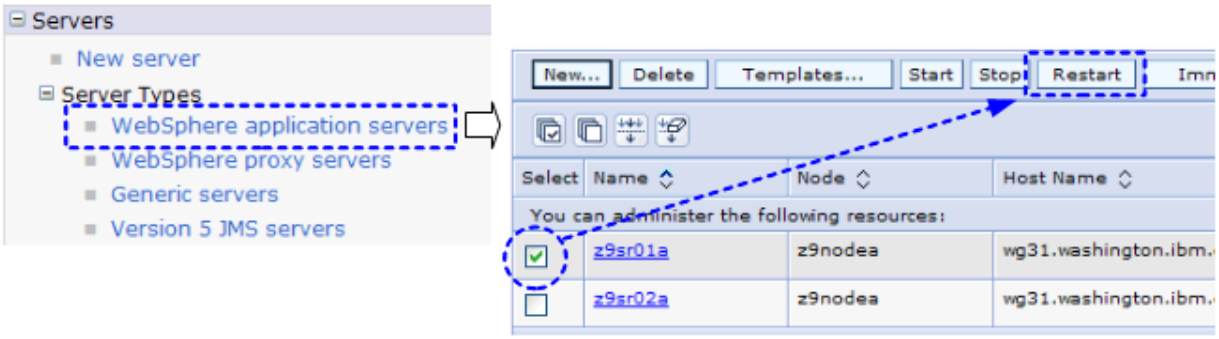
Note: The text logger is not as useful for WAS z/OS because it only applies to the control region JVM output. It defaults to on. It does represent at least some overhead. If you'd like, disable it.

- Because you've changed the logging mode to HPEL you have changes that need saving and synchronizing:

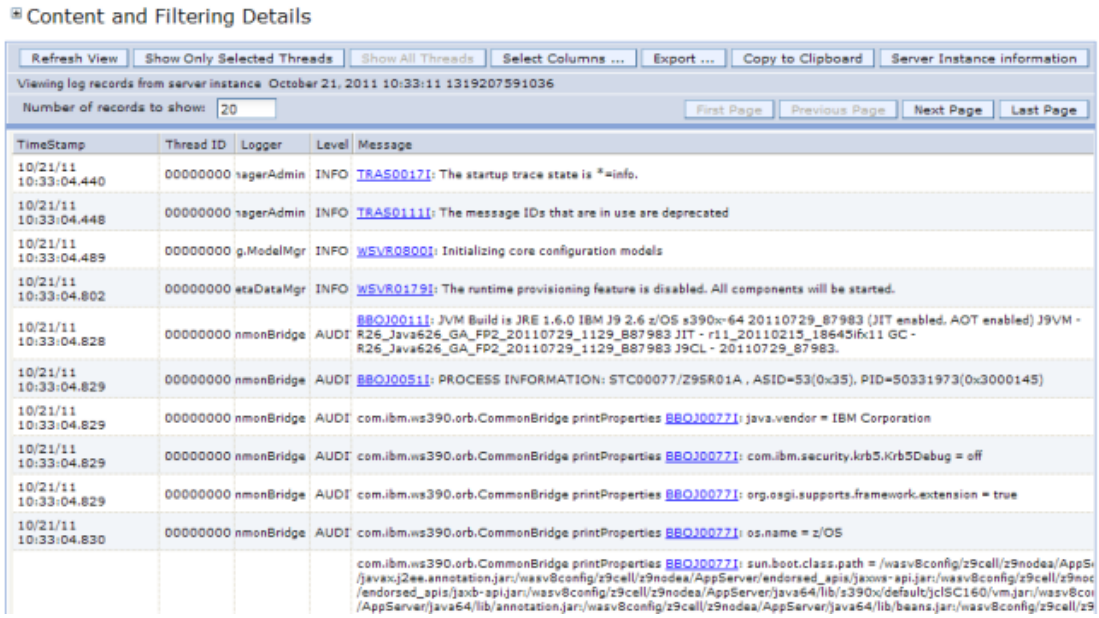


Do that now by clicking "Save."

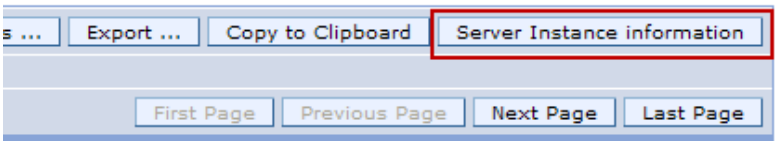
- To pick up the changes you need to restart your server:



- ☐ Go back to *Troubleshooting* ➔ *Logs and trace*. You should see four servers there. Click on the `z9sr01a` server, which is one in which you enabled HPEL.
- ☐ Under "Related Items" at the bottom of the page, click on "View HPEL logs and trace." This will begin to launch the Java applet log viewer in your browser window. It may take a moment or two. When it completes you should see something like this:



- ☐ Click the "Server Instance Information" button near the top right:



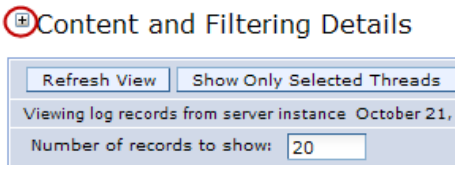
From the resulting panel (and looking nowhere else) capture the following information:

`os.version`	
`systemName`	
`java.version`	
`jobId`	

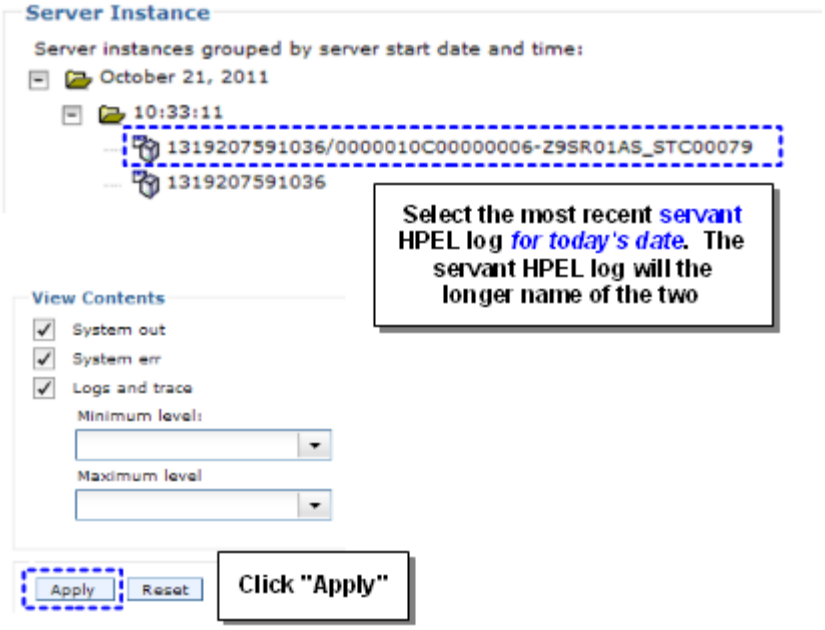
Note: The purpose of this exercise was to show that with HPEL this "Server Instance Information" button is a nice single place to get a lot of information about the server environment.

Close the information window when you're through.

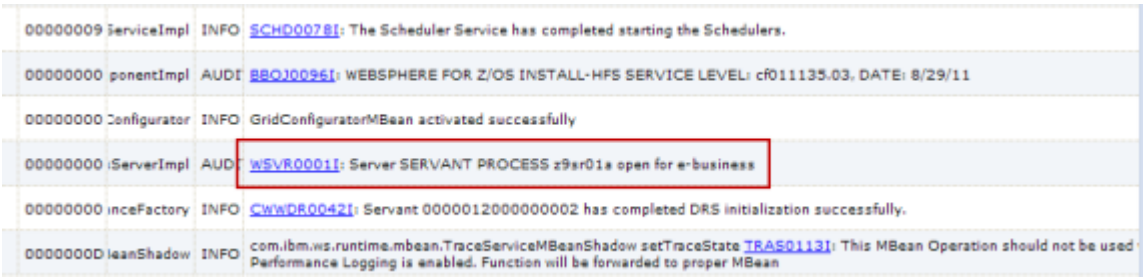
- ☐ Click the "Select Columns..." button and simply note you have the ability to tailor what columns appear.
- ☐ Now click the little "+" sign to the left of "Content and Filtering Details":



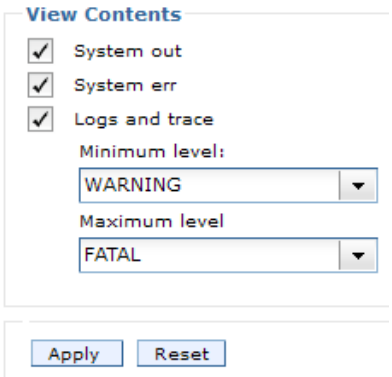
- ☐ Then:



- ☐ Click on the "Last Page" button ... you should see the "Open for e-business" message:



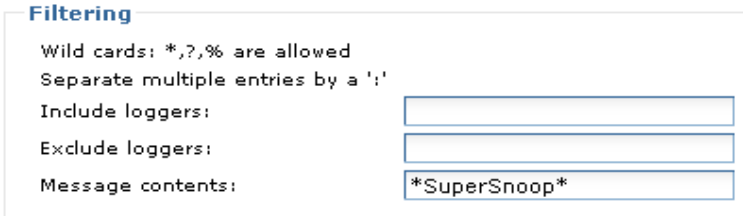
- ☐ Go back to the "Content and Filtering Details" section at the top of the page and set the minimum level to WARNING and the maximum level to FATAL and click "Apply"



How many records do you know see? What type are they -- warning, severe or fatal?

- ☐ Set the minimum and maximum back to blank ... click "Apply."

- ☐ Set the filtering to filter on the message contents for the string `*SuperSnoop*` :

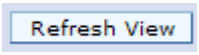


  Then click "Apply." At first you won't see any messages.

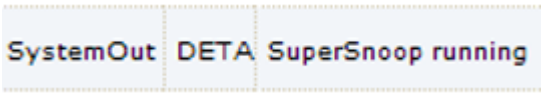
- ☐ From your browser, drive the SuperSnoop application:

  `http://wg31.washington.ibm.com:10067/SuperSnoopWeb/SuperSnoop`

- ☐ Refreshing the display in the log viewer with the "Refresh View" button:



- ☐ You should now see what the SuperSnoop application writes out to System Out:



- ☐ Go back to the browser and drive SuperSnoop several times. You should see the "You've visited this page" counter near the top of the page increment each time. Go back to the log viewer and "Refresh View." You should see more records:

- ☐ Now you'll switch to using the `logViewer.sh` shell script. Open TeraTerm/PuTTY, log in with the `z9admin` ID and change directories to the z9cell's `/AppServer/profiles/default/bin` directory.
- ☐ Enter the following command: **`./logViewer.sh -listInstances`**
- ☐ Copy the long instance value, which is the servant region instance. It'll look *something* like this:

  `1319235052657/0000012000000002-Z9SR01AS_STC00119`

  Paste that for safekeeping in a Notepad session, or update the "Lab Command Strings" file on the desktop.

- ☐ Issue the following command to create a human readable log file that contains every record in the binary HPEL log:

  `./logViewer.sh -instance <instance_string> -outLog /tmp/HPEL_full.out`

- ☐ When that completes, use the WS-FTP client to "view" the file on the mainframe. It's in EBCDIC, so select the "ASCII" option so it views in human readable form.

> **Note:** A file that may be downloaded or viewed on the mainframe using standard tools UNIX tools. The format of this file should be very recognizable to distributed WAS administrators.

- ☐ Now compose in Notepad the following command *as one line* and paste it into the TeraTerm/PuTTY session and execute:

```
./logViewer.sh -instance <instance_string>
                -includeLoggers SystemOut -outLog /tmp/HPEL_filter.out
```

- ☐ From a TeraTerm or PuTTY session, issue command **`cat /tmp/HPEL_filter.out`**

  You should see the SYSTEMOUT records for the SuperSnoop invocations you did earlier.

- ☐ For the final lab exercise issue the following command:

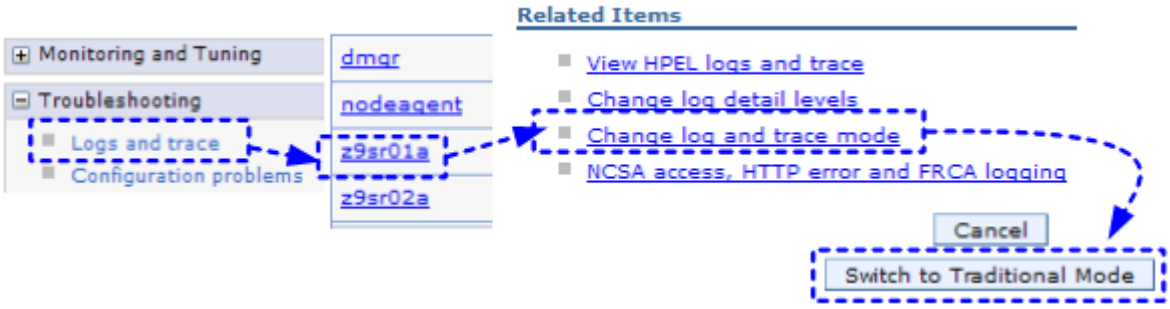
```
./logViewer.sh -instance <instance_string>
                                    -includeLoggers SystemOut -monitor 2
```

> **Note:** Rather than write to a file this will put output to the Telnet console. And the `-monitor 2` parameter tells it to check every two seconds and output any new records that meet any filtering criteria.
>
> When you execute that command it'll at first look like nothing is happening. But when you invoke SuperSnoop again you'll see those output lines come out.

- ☐ From your browser drive SuperSnoop again several more times. Go back to the TeraTerm or PuTTY session and you should see the new lines displayed. Drive SuperSnoop again and see those lines displayed.

> **Note:** This provides a way to monitor the activity of a server based on fairly granular filtering criteria.

- ☐ Issue a Ctrl-C command in the TeraTerm/PuTTY session to quit the `-monitor` processing.
- ☐ Close the TeraTerm or PuTTY session by issuing `exit` command.
- ☐ Turn *off* HPEL and reconfigure logging back to use JES spool:



- ☐ Save and synchronize the changes
- ☐ Restart the server.

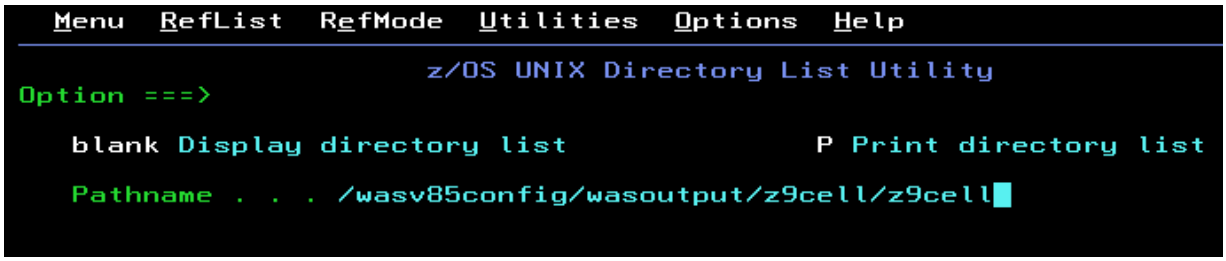
Output Enhancements with PM74923

- ☐ Logon to the AdminConsole and go to *Environment* ➔ *WebSphere variables.*
- ☐ Set the scope to `Cell=z9cell`.
- ☐ Add a new variable with Name = `DAEMON_redirect_server_output_dir` and Value = `/wasv85config/wasoutput/z9cell/z9cell`
- ☐ Add another new variable with Name = `redirect_server_output_dir` and Value = `/wasv85config/wasoutput/z9cell/z9cell`
- ☐ Save and synchronize the changes.

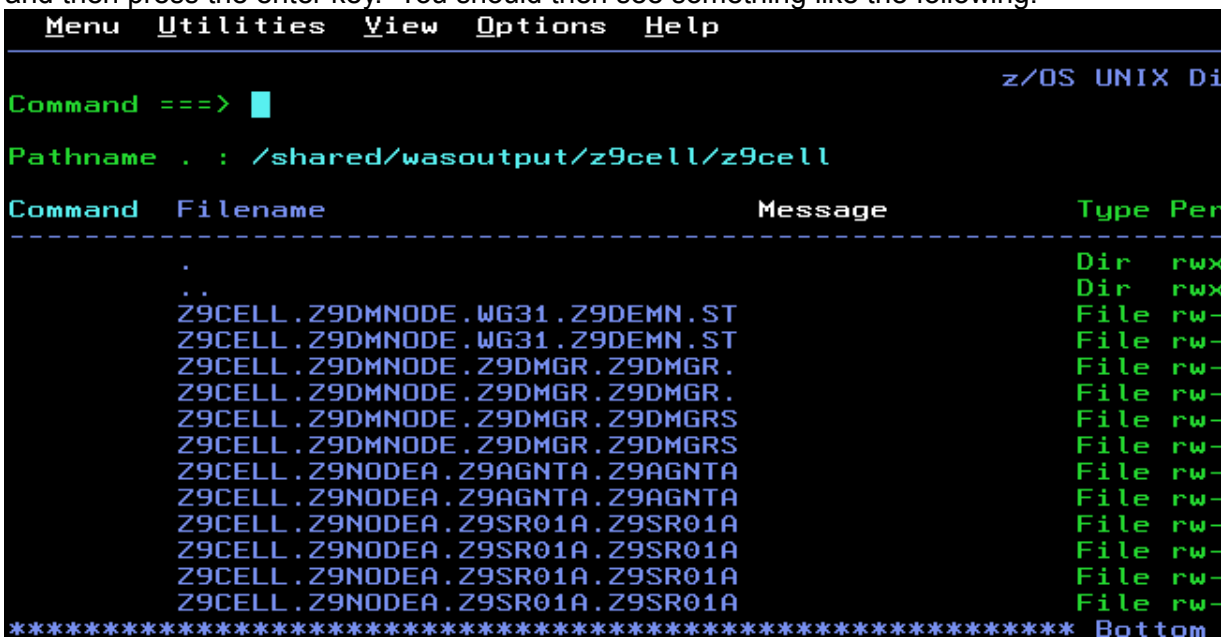
- ☐ Restart the Cell (in z/OS `=ISPF.LOG`, stop the Daemon with `/P Z9DEMN`, wait for everything to complete, then start the deployment manager, nodeagent, and server).

  Note: We have chosen to put all of the output components into the cell level directory, for demonstration purposes in this lab.

- ☐ Logon to TSO and go to option =3.17 to view the ouput.
- ☐ Provide the path to the output: `/wasv85config/wasoutput/z9cell/z9cell`.



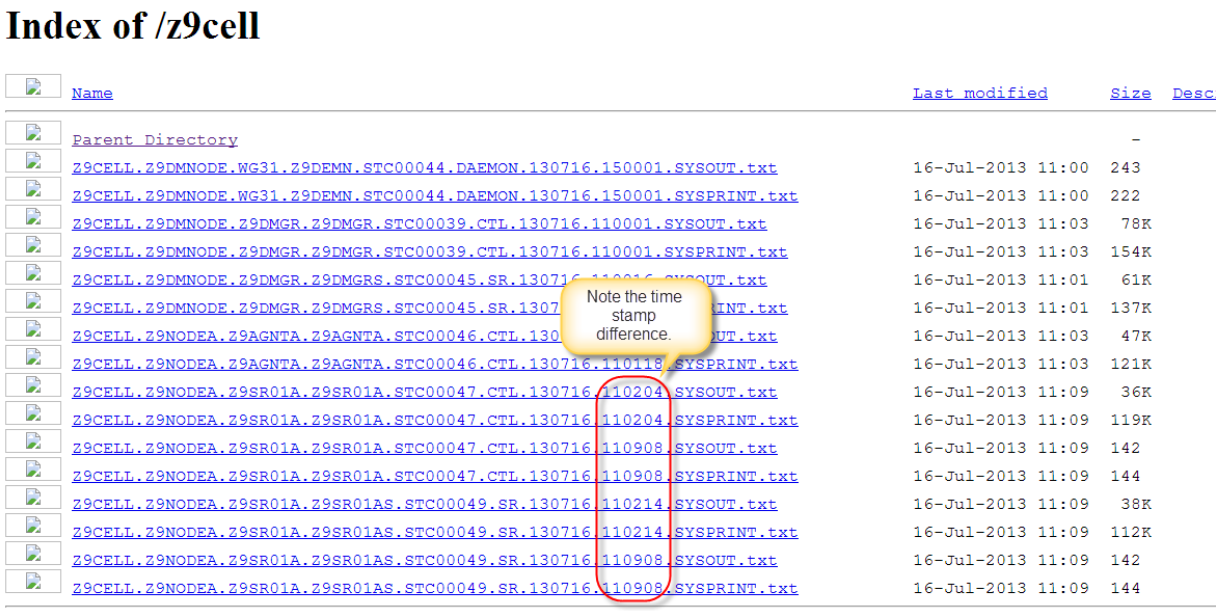
and then press the enter key. You should then see something like the following:



You can browse any of those output files by placing the letter **b** in the command field and pressing enter.

- ☐ An alternative way to view the output is with a browser and an HTTP server. To try this, go to `=SDSF.LOG` and enter the start command for the HTTP server: **`/S Z9HTTP01`**
- ☐ Open a browser session and enter the following URL:
  `http://wg31.washington.ibm.com:9080`
- ☐ You will be prompted for a userid and password. Use ID `USER1`.
- ☐ Navigate through to the `z9cell` directory where the output files are stored. You may then click on any one of them to view the output.
- ☐ To demonstrate the ability to switch to new output files for a server, enter the command in `=SDSF.LOG` to roll the logs for the `z9sr01a` server:
  `/F Z9SR01A,ROLL_LOGS`

- ☐ Hit the browser “refresh" button. Note the new output files for the server:



Note: If you examine the “old" and “new" output files, you should see the backward and forward pointing “links" in them. The last message in the “old" file should point to the location of the “new" file and the first message in the “new" file should point to the location of the “old" file.

- ☐ Logon to the AdminConsole and go to *Environment* ➔ *WebSphere variables.*
- ☐ Set the scope to `Cell=z9cell`.
- ☐ ***Delete*** the two variables we added at the beginning of this exercise that have the names `Daemon_redirect_server_output_dir` and `redirect_server_output_dir`
- ☐ Save and synchronize the changes.
- ☐ Shut down the HTTP Server in an SDSF with the following command:

```
S Z9HTTP01,action='stop'
```

Important: Be careful that you do this in the *extended command area* (type a single forward slash and hit enter). Otherwise, the stop command will be uppercased and will not work. The only part of the command where case is important is between the two apostrophes.

- ☐ Restart the Cell (stop the Daemon, wait for everything to complete, then start the deployment manager, nodeagent, and server).

End of Unit 2 Lab

Unit 3 Lab - Server Models

Multiple servants

Note: In this section you will explore the two means of achieving multiple servants: by way of the Administrative Console and use of the MVS `MODIFY` command.

☐ **Important!** Check and make sure you've set HPEL off (page 19), the PM74923 function is off (page 21), and the HTTP server is shut down (page 21).

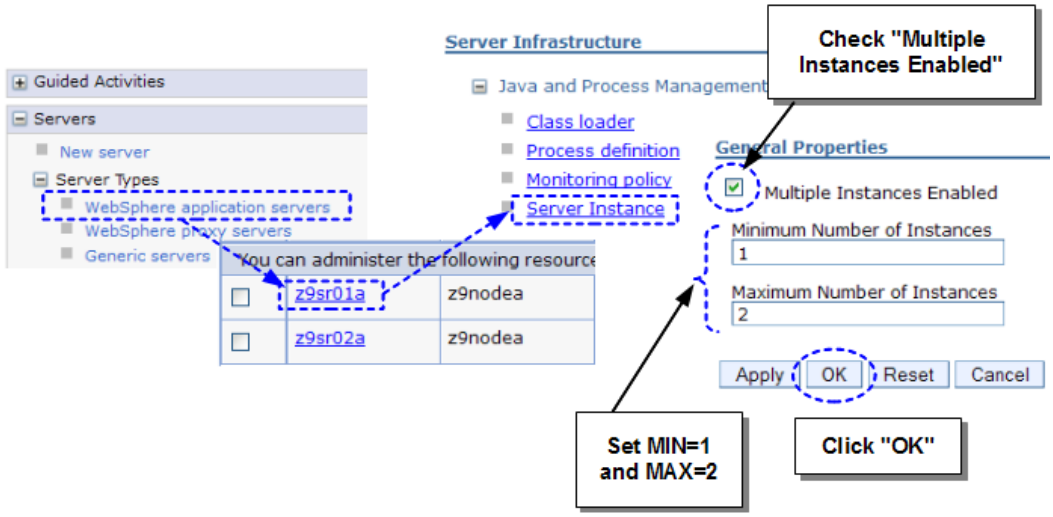
☐ In your 3270 session, go to `=SDSF.LOG` and issue the following command:

`/F Z9SR01A,WLM_MIN_MAX=2,2`

You should see an error message[13]:

```
BBOO0342E MODIFY COMMAND WLM_MIN_MAX=2,2 FAILED: WLM DAE CONFIGURED
SINGLE SERVER
```

☐ Let's fix that. In the Admin Console, navigate to the following location and do as indicated[14]:



☐ Save and synchronize all changes made up to this point.

☐ Restart your server to pick up all these changes.

☐ Check in the controller held output for the following messages:

```
wlm_maximumSRCount: 2.
wlm_minimumSRCount: 1.
```

☐ You should at this point have only one servant active. That's because of the minimum servant count of 1. Verify this by going to **`=SDSF.DA`** in the 3270 screen and issuing command **`PREFIX Z9*`**

```
SDSF DA WG31      WG31      PAG   0  CPU    2
COMMAND INPUT ===> _
NP    JOBNAME  StepName ProcStep JobID    ASIDX
      Z9DEMN   Z9DEMN   BBODAEMN STC00037 000F
      Z9DMGR   Z9DMGR   BBOPDCR  STC00034 003E
      Z9DMGRS  Z9DMGRS  BBOPDSR  STC00038 0041
      Z9SR01A  Z9SR01A  BBOPACR  STC00150 0035
      Z9SR01AS Z9SR01AS BBOPASR  STC00156 0021
```

If the names seem out of order, issue `SORT JOBNAME` to make the controller and servant appear in order

13 That's because by default servers are configured with "multiple instances" *disabled*. You'll change that next.
14 This tells WLM to enable "multiple instances" and to start with 1 and allow up to 2.

- ☐ Now open a TeraTerm or PuTTY session to your z/OS system and log on as **USER1** (*not* Z9ADMIN)[15]
- ☐ Change directories to `/wasetc/was8lab/other`
- ☐ Issue the command `./runjmeter` ... which starts JMeter[16] on the mainframe. In a moment you should see:

  ```
  Waiting for possible shutdown message on port 4445
  ```

- ☐ Then go to `=SDSF.DA`, hitting "enter" periodically to refresh the screen. In about ten seconds or so you should see your *second* servant come up:

  ```
  Z9SR01A  Z9SR01A
  Z9SR01AS Z9SR01AS
  Z9SR01AS Z9SR01AS
  ```

  Verify that you see "SuperSnoop running" messages in *both* servant region's SYSPRINT[17].
- ☐ Go back to your TeraTerm or PuTTY session and issue Ctlr+C to issue an interrupt and **stop** the running JMeter process.

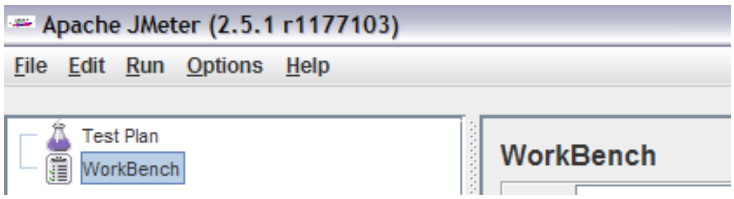
  The second servant will stay up. WLM can dynamically close that servant, but in general WLM will not close down servants quickly, particularly if nothing else on the system is competing for resources.
- ☐ In the Admin Console, go to the list of application servers and *restart* `z9sr01a`. It should come back up with one servant region, but be defined with `MIN=1` and `MAX=2`.
- ☐ For the next test section we want to show work being balanced across multiple servant regions. For that we'll need two active.

  You may manually increase the number of servant regions using the MVS `MODIFY` command. You will now see how that is done. In z/OS issue the command:

  `/F Z9SR01A,WLM_MIN_MAX=2,2`
- ☐ Give it a moment and then you should see the second servant region come up:

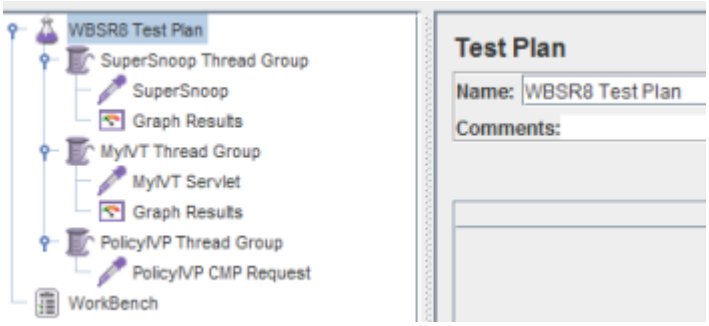
  ```
  JOBNAME  StepName ProcStep
  Z9DEMN   Z9DEMN   BBODAEMN
  Z9DMGR   Z9DMGR   BBOPDCR
  Z9DMGRS  Z9DMGRS  BBOPDSR
  Z9SR01A  Z9SR01A  BBOPACR
  Z9SR01AS Z9SR01AS BBOPASR
  Z9SR01AS Z9SR01AS BBOPASR
  ```

- ☐ On your desktop you should see an icon labled "JMeter." This is the GUI version of the tool. Double-click on that icon to launch the tool. If challenged ... click "Run" to authorize its execution. You should then see:

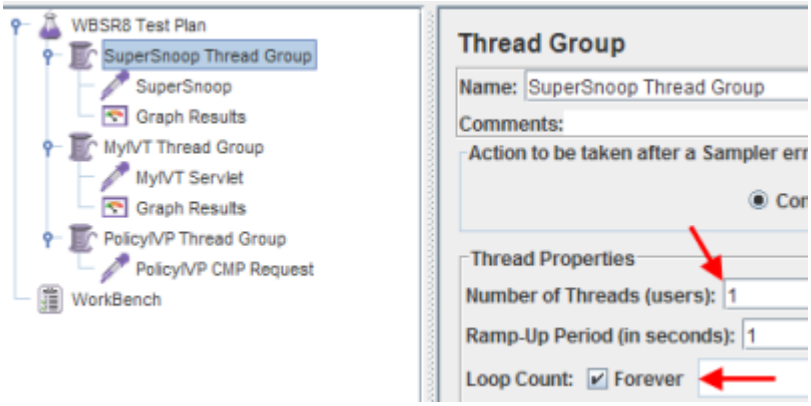


15 You'll get a funny FSUM error if you use Z9ADMIN. This is due to UNIX permissions. USER1 has authority.
16 JMeter is an open-source workload simulator test tool. We have that configured to drive 50 simulated users into your server. You may download a copy of the JMeter tool free of charge from: `http://jakarta.apache.org/site/downloads/downloads_jmeter.cgi`
17 Or System Out if HPEL.

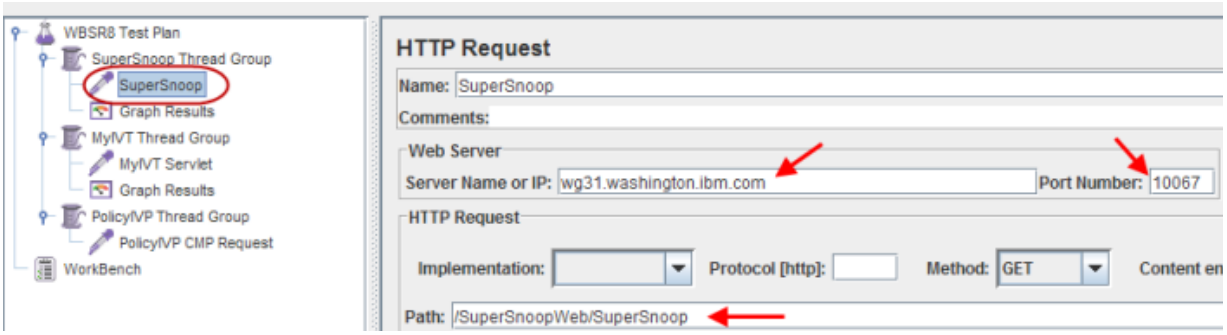
- ☐ In JMeter, go to *File* ➔ *Open* and select the `WBSR8.jmx` file that should appear in the selection list and click "Open". You should then see something like this:



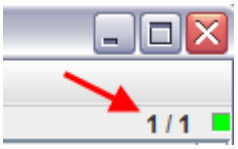
- ☐ Click on "SuperSnoop Thread Group" and note that it *starts* with only 1 simulated user defined and loops forever.



- ☐ Now click on "SuperSnoop" and note how the tool defines the host, port, the context root and servlet mapping as well as the HTTP method:



- ☐ Look at the "MyIVT Thread Group" as well as "PolicyIVP Thread Group" and you'll see both start with 0 thread users. Initially JMeter will drive only the SuperSnoop application.
- ☐ To run this, select the test plan name ("WBSR8 Test Plan"), then *Run* ➔ *Start.* Then look to the right and make sure you see a little green box with the number of active threads indicated:



- ☐ Look in both servant region's held output and you'll see that SuperSnoop is only seen running in one. WLM by default will favor one servant even if multiple are present, provided that one servant can do the work. With only one simulated users it *can* handle the work.

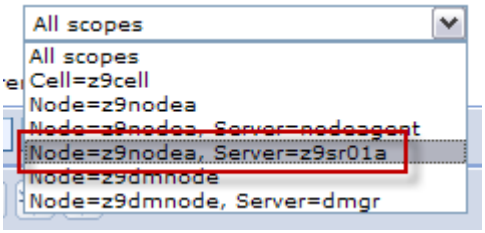
- ☐ In JMeter go to *Run* ➔ *Stop.* Look to make certain the active threads goes to 0.
- ☐ Change the SuperSnoop thread group threads to 10 and try again. You'll very likely still see messages in one servant but not the other.
- ☐ Stop the test run. Increase the number of JMeter thread users to 25 and restart. Look in the second servant ... do you see "SuperSnoop" running? If not, increase the threads to 50. Increase until the messages in the second are steady. How many threads did it take?
- ☐ Stop the test plan in JMeter.
- ☐ In the Admin Console, *stop* (but do ***not*** restart) the `z9sr01a` server again. This will provide a clean SYSPRINT for the next lab step.

Lesson While we might think WLM would "round robin" between available servants, that's not really true. By default it will favor the first until it has reason to place work in the second. Increasing the threads in JMeter was a means of placing enough work in the server for WLM to take action.

WLM "round robin" distribution

Note: What if you want WLM to "round robin" work between available servants? There's an environment variable that will do something close to "round robin," but not exactly[18].

- ☐ Create an environment variable. Go to *Environment* ⇨ *WebSphere Variables*.
- ☐ Set the scope to your "z9sr01a" server:



- ☐ Click on the "New" button and provide the following variable:

Name:	**`wlm_stateful_session_placement_on`**
Value:	**`true`**

- ☐ Click on the "OK" button, then save and synchronize.
- ☐ In the Admin Console go in and configure the servant regions to `MIN=2` and `MAX=2`. See page 22 for a refresher of where that panel is located.
- ☐ When complete, save and synchronize the changes and then start your server.
- ☐ In the controller region held output look for this message[19]:

```
BBOM0001I wlm_stateful_session_placement_on: 1.
```

- ☐ Insure you have two servants started.
- ☐ In JMeter, set the SuperSnoop Thread Group user count back to a value of **2**.
- ☐ Start the test case (select, *Run* ➔ *Start).*

18 Full details are in the WP101740 Techdoc at `ibm.com/support/techdocs`. In a nutshell, the environment variable we are about to explore will attempt to balance *affinities* across available servants. The SuperSnoop application establishes a servant affinity because it creates an HTTP session object.

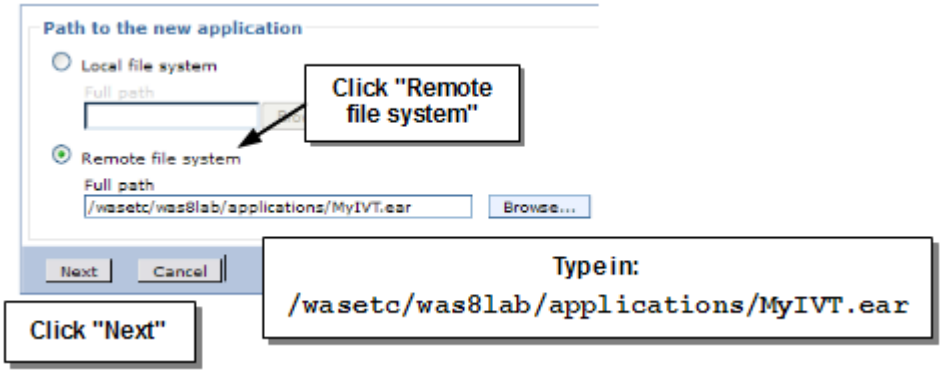
19 We have you check for these messages to insure the variable has taken effect. One small typo could invalidate it.

- ☐ Look in the held output for each servant region for the "SuperSnoop running" messages. Recall that earlier we saw with two users the work all went to one servant. Here you should see a kind of "balance" between the servants.
- ☐ After a minute or two stop the JMeter test case.
- ☐ Delete the `wlm_stateful_session_placement_on` environment variable. Save and synchronize.
- ☐ Stop the server but *do not* restart it. We have a few other changes to make before restarting.

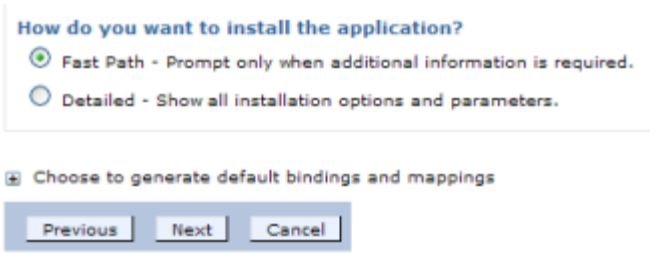
WLM classification

Note: In this section you will use the XML classification file to achieve separate WLM classification for two applications. You will see how this results in the placement of each application work into its own servant region.

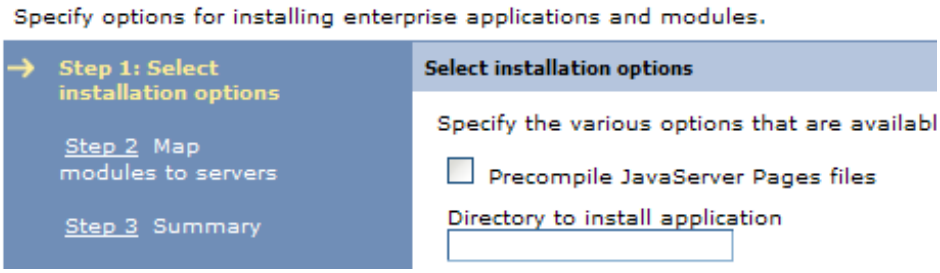
- ☐ We'll install a second application so we can easily tell that work for one is placed in a servant and work for the other is placed in the *other* servant. Go to *Applications* ➔ *New Application* ➔ *New Enterprise Application*. Then:



- ☐ On the next panel take the defaults and click "Next"



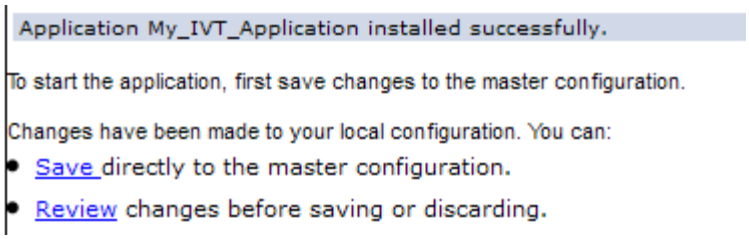
- ☐ The next panel you see begins the application installation:



However, you will take all the defaults so click on "Step 3 Summary"

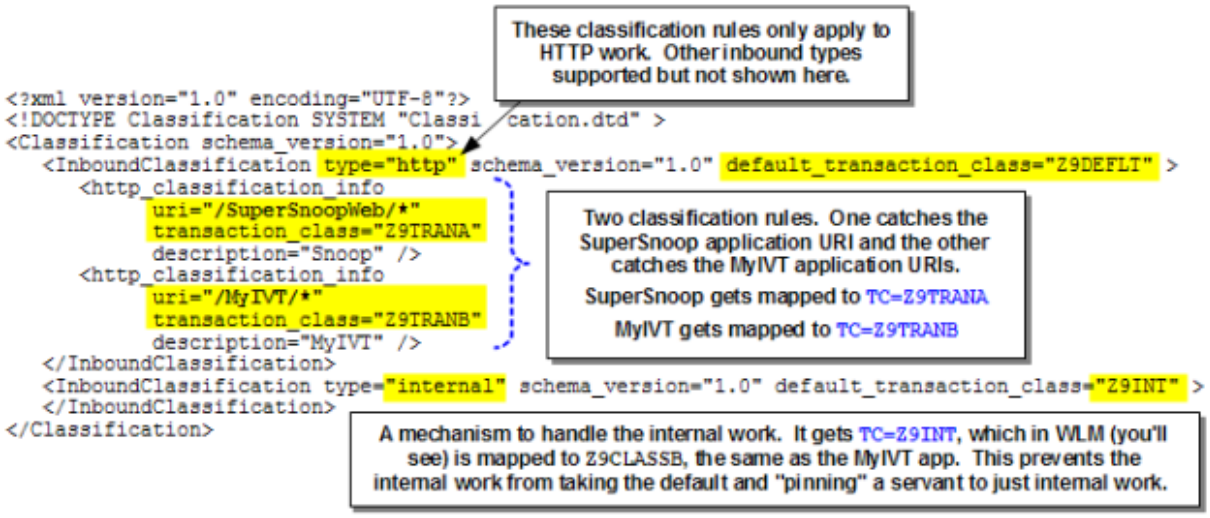
- ☐ Click on the "Finish" button.

- ☐ Then click on the "Save" link:

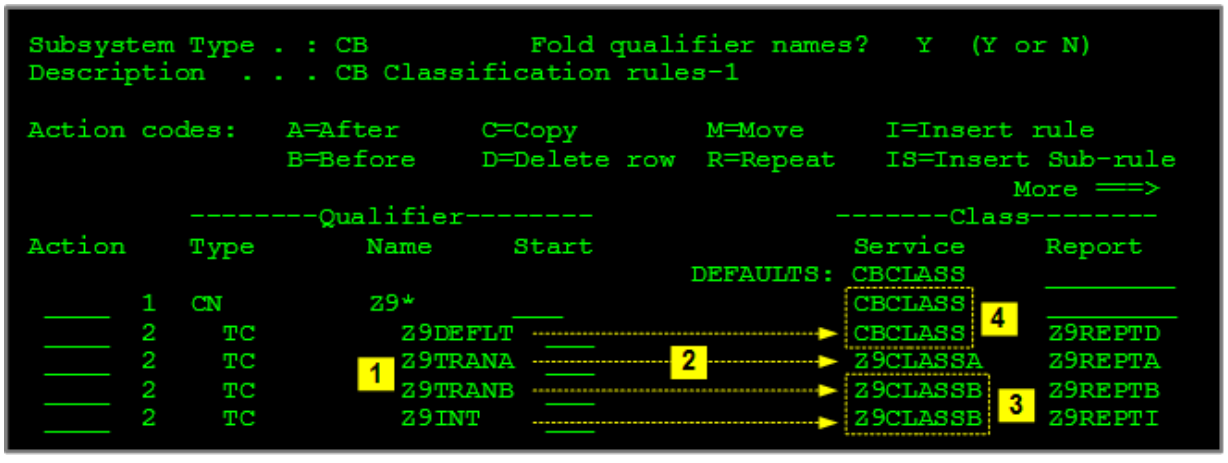


- ☐ When synchronization completes, then go to *Applications* ➔ *Application types* ➔ *WebSphere Enterprise Applications*. You should see `My_IVT_Application` in the list along with `SuperSnoop`.
- ☐ Using the WS-FTP client, view the following file that's stored in the host file system:
  `/wasetc/was8lab/other/`**`classification.xml`**

  That file is stored in ASCII so in WS-FTP you must select the "Binary" option when viewing. Here's a roadmap to what's in that file:



- ☐ Take a look at this picture. It's a bitmap clip[20] from your system's WLM CB classification rules we set up before the class. Just review the picture; don't worry about going into WLM:



1 The four transaction classes specified in the classification file.

20 We're providing a bitmap rather than driving you into WLM. For those unfamiliar with WLM it can be a daunting place.

2 The mapping of transaction class to service class.

3 Note that both `Z9TRANB` and `Z9INT` get mapped to the same Service Class (`Z9CLASSB`), but different reporting classes. This means for reporting and analysis you can separate out the internal work from your real work even though the same service class is in use.

4 The `Z9DEFLT` transaction class gets mapped to `CBCLASS`, which is the same Service Class as defaults when just the `CN` name of `Z9*` is used.

☐ Now take a look at what the service class goals we set up ahead of time:

```
Z9CLASSA   90% complete within 1.0,  Imp=1
Z9CLASSB   95% complete within 0.5,  Imp=1
```

Both are response time goals. `Z9CLASSB` (MyIVT) will have a slightly higher priority.

☐ The thing that allows the controller to identify inbound work and assign a TC is the classification XML file. The next step is to tell the `Z9SR01A` server about the classification file so it may load it. This is done with an environment variable.

Create the following environment variable at the server scope, just like the other environment variables you created:

Name:	**wlm_classification_file**
Value:	**/wasetc/was8lab/other/classification.xml**

Save and synchronize.

☐ Start the server.

☐ When the server comes up, check in the *controller* region's held output for either success or failure[21] loading the classification file.

Success:	**BBOJ0129I**: The /wasetc/was8lab/other/classification.xml workload classification file was loaded ...
Problem:	**BBOJ0085E**: PROBLEMS ENCOUNTERED PARSING WLM CLASSIFICATION XML FILE ...

☐ One final verification of this before we start driving work. Issue the following `MODIFY` command:

/F Z9SR01A,DISPLAY,WORK,CLINFO

You should see a response that looks *something* like this:

```
BBOJ0129I: The /wasetc/was8lab/other/classification.xml workload
classification file was loaded at 2011/10/23 13:15:14.861 (EDT)
BBOO0281I CLASSIFICATION COUNTERS FOR HTTP WORK
BBOO0282I CHECKED 0, MATCHED 0, USED 0, COST 3, DESC: HTTP root
BBOO0282I CHECKED 0, MATCHED 0, USED 0, COST 2, DESC: Snoop
BBOO0282I CHECKED 0, MATCHED 0, USED 0, COST 3, DESC: MyIVT
BBOO0283I FOR HTTP WORK: TOTAL CLASSIFIED 0, WEIGHTED TOTAL COST 0
BBOO0281I CLASSIFICATION COUNTERS FOR INTERNAL WORK
BBOO0282I CHECKED 368, MATCHED 368, USED 368, COST 1, DESC: Internal root
BBOO0283I FOR INTERNAL WORK: TOTAL CLASSIFIED 162, WEIGHTED TOTAL COST 162
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,WORK,CLINFO
```

21 Common problems include (a) file not found; (b) file permissions do not allow read; (c) file stored in EBCDIC, not ASCII; or some XML structure problem in an otherwise accessible file.

Note: For now this validates that WAS z/OS recognizes the HTTP element in the XML and the two classification rules Snoop and MyIVT as well as internal. Later it can be used to validate *how* the classification rules are being evaluated and met.

- ☐ Now, back in JMeter, set the SuperSnoop thread group user count to **2** and the MyIVT thread group user count to **2** as well. Start the test case.
- ☐ Check the held output for each servant. You should see SuperSnoop running in one and MyIVT in the other.

SuperSnoop	`SuperSnoop running`
MyIVT	`### ivtServlet ### -- in method service and about to write HTML` `### ivtServlet ### -- finished writing HTML`

You should *not* see intermixing of the two.

- ☐ After a period of activity, issue **`F Z9SR01A,DISPLAY,WORK,CLINFO`** again. You should see a lot more numbers for "checked," "matched" and "used." Can you determine if each request is evaluated against the XML from top to bottom? If so, then does it make sense to sequence the highest volume classification rules higher?[22] That's what the "cost" factor is getting at.
- ☐ In your TSO session, go to **`=SDSF.ENC`** to display active WLM enclaves, which will be in bright white type. If you don't see any, press the "enter" key to refresh a few times. The enclaves come and go quickly. You *may* see something like this:

NAME	SSType	Status	SrvClass	Per	PGN	RptClass
F80002BDC8	CB	ACTIVE	Z9CLASSA	1		Z9REPTA
1340002BDCA	CB	ACTIVE	Z9CLASSA	1		Z9REPTA
2400000002	STC	INACTIVE	OPS_LO	1		
2000000001	STC	INACTIVE	SYSTEM	1		

You won't likely see the MyIVT enclave ... that application executes very quickly. SuperSnoop does a bit more work and thus we're able to see the enclaves active.

- ☐ Now issue **`=RMF`** to go the Report Monitoring Facility. Then enter **`3`** for "3 Monitor III." Then **`S`** for "Sysplex." And finally **`1`** for "SYSSUM." You should see *something* like this:

```
                 ------- Goals versus Actuals --------  Trans
                 Exec Vel  --- Response Time ---  Perf  Ended
Name      T  I   Goal Act  ---Goal--- --Actual--  Indx  Rate

ONL_WKL   W           100                                18.91
CBCLASS   S  2     75 0.0                             N/A  0.020
Z9CLASSA  S  1        100  1.000 90%          99%  0.60  3.620
Z9CLASSB  S  1        0.0  0.500 95%         100%  0.50  15.27
```

In this example 99% of the `Z9CLASSA` work is seeing better than 1 second response time. And 100% of the `Z9CLASSB` work is seeing better than 0.5 second response time.

What you see may be different. These numbers change all the time.

- ☐ Exit from RMF
- ☐ Stop the JMeter test case that's running.

Granular RAS

Note: In WAS z/OS V8 they took advantage of this XML classification file to provide certain controls down to those requests that match in the file. In this lab we'll illutrate two: `message_tag` and `classification_only_trace`.

22 Yes, and yes.

- ☐ Using the WS-FTP client on your desktop[23], download the `classification.xml` file in *binary* to your workstation.
- ☐ *Rename* the file on your workstation to something like `classification2.xml`

> **Why?** We're about to show you how to dynamically load a *different* classification file into WAS.

- ☐ Edit it and *add* the following:

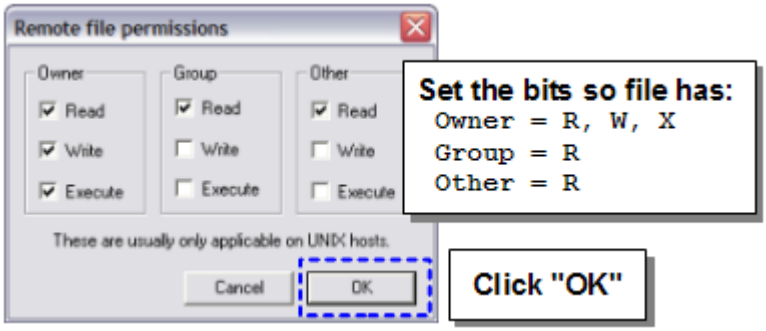
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Classification SYSTEM "Classification.dtd" >
<Classification schema_version="1.0">
   <InboundClassification type="http" schema_version="1.0" default_transaction_class="CBCLASS" >
      <http_classification_info
            uri="/SuperSnoopWeb/*"
            transaction_class="Z9TRANA"
            description="Snoop" />
      <http_classification_info
            uri="/MyIVT/*"
            transact
            descript
   </InboundClassifi
   <InboundClassific
   </InboundClassifi
</Classification>
```

```
uri="/SuperSnoopWeb/*"
transaction_class="Z9TRANA"
description="Snoop" message_tag="WBSR8" />
```

space

> **Note:** This will append the custom message tag to all log and trace records associated with requests that match.

- ☐ Upload that back to the mainframe in *binary*.
- ☐ On the right side of the WS-FTP client, select the new file and *right*-mouse click. Select the option `chmod(UNIX)` and then:



- ☐ You'll use the `MODIFY` command to dynamically load the *new* classification file. From `=SDSF.LOG` enter a single forward slash (/) to open the command extension, then enter:

```
F Z9SR01A,RECLASSIFY,
        FILE='/wasetc/was8lab/other/classification2.xml'
```

  No spaces in that. Case matters for the path and file name so type carefully.
- ☐ It will respond with a message indicating either success or failure. If failures, then look in the controller region output for some hint of what error was encountered. Correct and re-issue the command until you get success.
- ☐ Go back to JMeter. Start the test case, let it run for 10 seconds or so, then stop it.

23 See "Using the WS-FTP client" on page 86 for guidance on usage.

- Just like before, you should see SuperSnoop going to one servant and MyIVT going to another. But you'll now see something like this:

  `(tag=WBSR8)` `SuperSnoop running`

- On the workstation create a copy of the XML file. Call it `classification3.xml`. Edit that file and make the following changes:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Classification SYSTEM "Classification.dtd" >
<Classification schema_version="1.0">
   <InboundClassification type="http" schema_version="1.0" default_transaction_class="CBCLASS" >
      <http_classification_info
            uri="/SuperSnoopWeb/*"
            transaction_class="Z9TRANA"
            description="Snoop" message_tag="WBSR8" />
      <http_classification_info
            uri="/MyIVT/*"
            transaction_class="Z9TRANB"
            description="MyIVT" />
   </InboundClassification>
   <InboundClassification type="internal" schema_version="1.0" default_transaction_class="Z9INT" >
   </Inbound
</Classifica
```

```
uri="/MyIVT/*"
transaction_class="Z9TRANB"
description="MyIVT" classification_only_trace="1" />
```

space

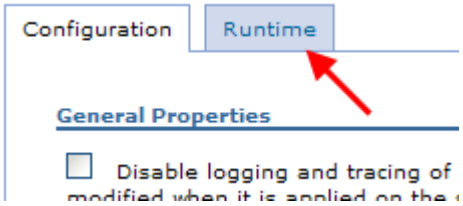
Note: This will provide detailed tracing for only those requests to match.

- Upload that file *in binary format* to the `/wasetc/was8lab/other directory`.
- After upload, `chmod` the file like you did on the previous page.
- Issue the following command *as one line* from `=SDSF.LOG`

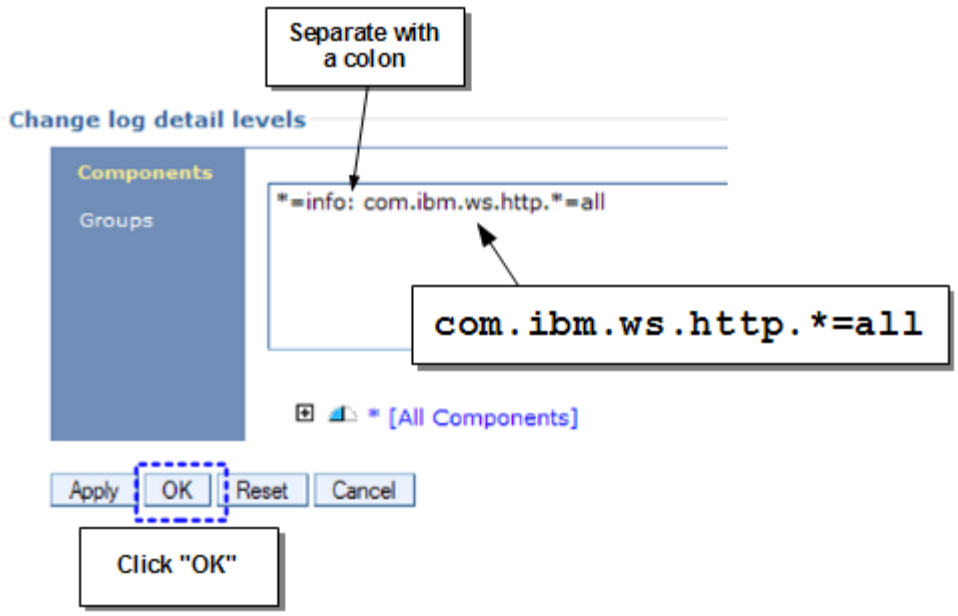
```
F Z9SR01A,RECLASSIFY,
        FILE='/wasetc/was8lab/other/classification3.xml'
```

  Make sure it loaded successfully. If errors, check the contoller region for a hint. Fix the error, re-upload and reclassify.

- In the Admin Console go to *Troubleshooting* ➔ *Logs and trace*. Select the `z9sr01a` server.
- Click on "Change log detail levels", then click on the "Runtime" tab:



- ☐ Then do the following:



- ☐ Go back to JMeter. Start the test case, let it run for 10 seconds or so, then stop it.
- ☐ If you look in the servant region for SuperSnoop you'll still see the message tags:

```
(tag=WBSR8) SuperSnoop running
(tag=WBSR8) SuperSnoop running
(tag=WBSR8) SuperSnoop running
```

Notice there's no trace records ... it's all the SuperSnoop messages. No tracing records. Prior to V8 the update to the tracing level would have applied to everything in the server. Now it can be made to apply to only specified requests.

But if you go to the other servant region you'll see:

```
Trace: 2011/10/24 16:16:28.480 02 t=9CB4F8 c=UNK key=P8 tag= (13007004)
  SourceId: com.ibm.ws.http.channel.impl.HttpBaseMessageImpl
  ExtendedMess                                    t (Cookie)
Trace: 2011/10                                   9CB4F8 c=UNK key=P8 tag= (13007004)
  SourceId: co                                   impl.HttpBaseMessageImpl
  ExtendedMessage: Header not present (Cookie2)
### ivtServlet ### -- in m                       t to write HTML
### ivtServlet ### -- finis
Trace: 2011/10/24 16:16:28.                      key=P8 tag= (13007004)
  SourceId: com.ibm.ws.http.channel.impl.HttpResponseMessageImpl
  ExtendedMessage: Serializing: com.ibm.ws.http.channel.impl.HttpResponseMessage
```

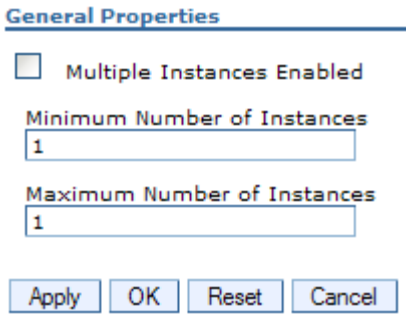
Trace records based on the tracing update you made.

MyIVT output as before

Environment cleanup

- ☐ **Stop** the JMeter test run ← *Important*
- ☐ Close JMeter.
- ☐ Go to *Troubleshooting* ➔ *Logs and trace*. Select the `z9sr01a` server. Set the trace level back to just `*=info` (in other words, remove the more detailed tracing set earlier).

- ☐ Set the servant MIN, MAX back to default:



See page 22 for a reminder of where this is located.

- ☐ Save and synchronize the changes.
- ☐ Stop the `z9sr01a` server.

Liberty Profile - first steps

- ☐ Open a TeraTerm or PuTTY session to your system. Log on as **`USER1`**.
- ☐ Issue the command **`pwd`** ... you should see you're in the `/u/user1` directory
- ☐ Issue the command **`mkdir liberty`** to create that directory under `/u/user1`
- ☐ Issue the command **`export WLP_USER_DIR=/u/user1/liberty`** ... this creates and exports that value to the UNIX shell environment.
- ☐ Validate that: **`echo $WLP_USER_DIR`** ... you should see the proper value echoed back
- ☐ Issue **`export JAVA_HOME=/shared/zWebSphere/V8R5FP02/java64`** ... this provides the UNIX shell environment with a pointer to a valid 64-bit Java installation[24].
- ☐ Validate that as well: **`echo $JAVA_HOME`**
- ☐ Change directories to `/shared/zWebSphere/V8R5FP02/wlp/bin` ... that is the read-only installation location for WAS V8.5. The `/wlp` directory is where Liberty was installed.
- ☐ Issue the command **`server create server1`** ... this will create a basic server configuration in the `WLP_USER_DIR` location, which you set earlier.
- ☐ Issue the command **`server start server1`** ... this will start the server. You should see a response *something* like this:
  ```
  Server server1 started with process ID 67108958.
  ```
- ☐ Issue the command **`server status server1`** ... that should echo back the same message you just saw when you started the server.
- ☐ Using ISHELL, navigate to the `/u/user1/liberty/servers/server1/logs` directory. Place an `E` next to the `messages.log` file and press Enter.[25] Scroll to the right (F11) and you should see something like this:
  ```
  The server server1 has been launched.
  The angel process is not available.  No authorized services will be loaded.
  Authorized service group SAFCRED is not available.
  Authorized service group TXRRS is not available.
  Authorized service group ZOSDUMP is not available.
  Authorized service group ZOSWLM is not available.
  IBM product WAS FOR Z/OS version 8.5 successfully registered with z/OS.
  Feature update started.
  ```

24 In this case it is the Java that's provided with WAS z/OS Version 8.5.
25 The file is actually stored in ASCII on the mainframe, but it's tagged so tools like OEDIT may auto-convert it.

```
The kernel started after 2.021
TCP Channel defaultHttpEndpoint has been started and is now listening
                        for requests on host LOCALHOST  (IPv4: 127.0.0.1) port 9080.
Monitoring dropins for applications.
Monitoring dropins for applications.
Feature update completed in 0.56 seconds.
The server server1 is ready to run a smarter planet.
```

- ☐ Using ISHELL, navigate to the `/u/user1/liberty/servers/server1` directory and edit the `server.xml` file. Change the `host=` value to an *asterisk* (*) and save the file:

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

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

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


</server>
```

Change to: host="*"

- ☐ Go back and edit the `messages.log` file and you'll see Liberty dynamically detected the change and started listening on all hosts, not just "localhost":

```
TCP Channel defaultHttpEndpoint has stopped listening for requests on host LOCALHOST
(IPv4: 127.0.0.1) port 9080.
TCP Channel defaultHttpEndpoint has been started and is now listening for requests
on host *  (IPv4) port 9080.
```

- ☐ There is a directory `/u/user1/liberty/servers/server1/dropins` that was created when the server was first started. By default Liberty is monitoring this directory for application updates. Copy a servlet application in with this command entered as one line in the TeraTerm or PuTTY session:

```
cp /wasetc/was8lab/applications/ATS_servlet.war
              /u/user1/liberty/servers/server1/dropins/ATS_servlet.war
```

- ☐ Go back to the `messages.log` file and notice that Liberty detected the new application and started it:

```
CWWKZ0018I: Starting application ATS_servlet.
SRVE0169I: Loading Web Module: ATS_servlet.
SRVE0250I: Web Module ATS_servlet has been bound to default_host.
CWWKT0016I: Web application available (default_host):
                            http://WG31.WASHINGTON.IBM.COM:9080/ATS_servlet/*
SRVE9998I: Application ATS_servlet added to web container.
CWWKZ0001I: Application ATS_servlet started in 0.83 seconds.
```

- ☐ Point your browser at this URL to invoke the application:
  `http://wg31.washington.ibm.com:9080/ATS_servlet/SimpleServlet`
  You should see something like this:

- ☐ From the `/shared/zWebSphere/V8R5FP02/wlp/bin` directory, issue the command: **`server stop server1`**

  You should see a message indicating the server has been stopped.

Note: That's the basics -- simple to set up, dynamic, fast. Server startup is measured in seconds. But the application was also very simple -- a servlet with no data access.

Next you'll start the server as a z/OS Started Task, then you'll use JDBC T2 with RRS.

Liberty Profile - z/OS Started Task

- ☐ Look in the `/shared/zWebSphere/V8R5FP02/wlp/templates/zos/procs` directory. You'll see two files -- `BBGZANGL` and `BBGZSRV`. Those are the sample JCL start procs. Those have been copied to `SYS1.PROCLIB` prior to the workshop.
- ☐ Edit the `SYS1.PROCLIB(BBGZSRV)` member and make the following two updates:

```
000010 //*           WLP_USER_DIR environment variable in the Unix shell.
000011 //*-------------------------------------------------------------------
000012 //  SET INSTDIR='/shared/zWebSphere/V8R5FP02/wlp'
000013 //  SET USERDIR='/u/user1/liberty'
000014 //*-------------------------------------------------------------------
000015 //* Start the Liberty server
```

  Be sure to enclose with single quotes, and make certain the case is correct.
- ☐ Before you can start that proc a RACF `STARTED` profile is needed to assign identity to the started task. The job `USER1.WAS.CNTL(LIBRACF1)` has been created for that purpose. Submit that job and inspect the output to make sure the `STARTED` profiles were created.
- ☐ Create the file[26] `/u/user1/liberty/servers/server1/server.env` with permission `755` and populate it with one line:
  `JAVA_HOME=/shared/zWebSphere/V8R5FP02/java64`
  Save the file.
- ☐ Go to `=SDSF.DA` and set the prefix to `PRE BBG*`
- ☐ Open the z/OS command extension[27] (single forward slash) and start the server with the following command:
  `S BBGZSRV,PARMS='server1'`

Note: If you wished you could add `JOBNAME=` to that.

26 Either ASCII or EBCDIC ... Liberty will work with either.

27 Your server is lower-case `server1`. The `PARMS=` field must have that in lowercase to match what's in the file system. Mixed-case commands are supported in the command extension. That's why we have you use the extension for this.

- ☐ You should see the server start up. Check the output for the job ... you should see messages indicating the application has started and the server is ready.
- ☐ Point your browser at:
  `http://wg31.washington.ibm.com:9080/ATS_servlet/SimpleServlet`

  You should see the same result you saw before.
- ☐ Stop the server:
  `/P BBGZSRV`

Liberty Profile - JDBC Type 2 using Angel Process

Note: In this section you'll use JDBC Type 2 to access DB2. JDBC T2 implies RRS, and that implies use of a z/OS authorized service. Which means you'll need the Angel process.

- ☐ Start DB2 by issuing the following command from `=SDSF.LOG`

  `/-DSNX START DB2`

  Give it a few moments to come up. Look for this message as a sign of success:

  `DSN9022I -DSNX DSNYASCP 'START DB2' NORMAL COMPLETION`
- ☐ The job `USER1.WAS.CNTL(LIBRACF2)` creates a set of `SERVER` profiles that provides access to z/OS authorized services[28]. Submit the job and insure the profiles are created.
- ☐ Edit the `SYS1.PROCLIB(BBGZANGL)` proc and make one update:

```
000001 //BBGZANGL PROC PARMS='',COLD=N
000002 //*--------------------------------------------
000003 //  SET ROOT='/shared/zWebSphere/V8R5FP02/wlp'
000004 //*--------------------------------------------
000005 //* Start the Liberty angel process
000006 //*--------------------------------------------
```

- ☐ Start the angel process with the following command:
  `/S BBGZANGL`
  In the job output you should see:
  `CWWKB0056I INITIALIZATION COMPLETE FOR ANGEL`
- ☐ To use JDBC T2 you'll need a few `server.xml` updates. Rather than having you type all that we supplied the file. Issue the following command (on one line) to copy the file:

```
cp /wasetc/was8lab/other/server.xml
                              /u/user1/liberty/servers/server1/server.xml
```

- ☐ Take a look at the new `server.xml` file and note the JDBC artifacts[29]. Try to see how the XML really just points to the DB2 JDBC libraries and provides a data source value for the application to use.
- ☐ Your z/OS system has two DB2 subsystems installed, so it's necessary to let Liberty Profile know *which* subsystem you wish it to connect to. To keep this simple we created the files ahead of time, and all you need to do here is perform a simple copy:

```
cp /wasetc/was8lab/other/jvm.options
                             /u/user1/liberty/servers/server1/jvm.options
```

Note: The `jvm.options` file contains a pointer to another file. That other file contains the subsystem ID to connect to. If you're curious, take a look in the just-copied `jvm.options` file and follow the pointer to see how the SSID is specified.

28 Six `RDEFINE` statements are present but only three are needed for this exercise (the first two and `TXRRS`). The others are present for completeness -- they represent the other three authorized services supported by Liberty z/OS.

29 Information on `server.xml` elements and syntax may be found in InfoCenter, search on `rwlp_metatype_4ic`

- From the command extension start the server again:
  `S BBGZSRV,PARMS='server1'`
- Look at the `/u/user1/liberty/servers/server1/logs/messages.log` file. You should see the following:
  ```
  Authorized service group SAFCRED is available.
  Authorized service group TXRRS is available.
  Authorized service group ZOSDUMP is available.
  Authorized service group ZOSWLM is available.
  ```
  That indicates the Angel process is present and the RACF `SERVER` updates are in place to allow the user (`USER1` in this case) to access the authorized services.
- In that same `messages.log` file you should also see messages indicating the JDBC Type 2 driver is available.
- Now drop the JDBC application into the `/dropins` directory with this copy command:
  ```
  cp /wasetc/was8lab/applications/ATS_jdbc.war
                  /u/user1/liberty/servers/server1/dropins/ATS_jdbc.war
  ```
- If you refresh your view of the `messages.log` file you should see that the JDBC application has been started.
- Point your browser at:
  `http://wg31.washington.ibm.com:9080/ATS_jdbc/sample.html`
  You should see a page that looks like this:

IBM Advanced Technical Skills, Gaithersburg, Maryland

ATS JDBC Type 2 Sample Applications

Insert Records into LIBERTY.RECORDS table in DB2 z/OS
Click here to run the servlet.

Select and display Records from LIBERTY.RECORDS table in DB2 z/OS
Click here to run the servlet.

Delete Records from LIBERTY.RECORDS table in DB2 z/OS
Click here to run the servlet.

- Click the first link "Insert" to insert 10 rows into the database table. It will echo back to your browser screen the records inserted.
- Use the browser "back" button to return to the initial application screen
- Click "Insert" again ... you should now see 20 rows in the table.
- Go back and click on the "Delete" link (last one) ... this should delete all the rows.
- Click on "Select" ... you should see no rows displayed since "Delete" removed them all.
- Stop the Liberty server: `/P BBGZSRV`
- Stop the Angel process: `/P BBGZANGL`

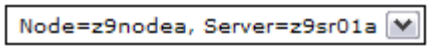
End of Unit 3 Lab

Unit 4 Lab - Accessing z/OS Data

JDBC

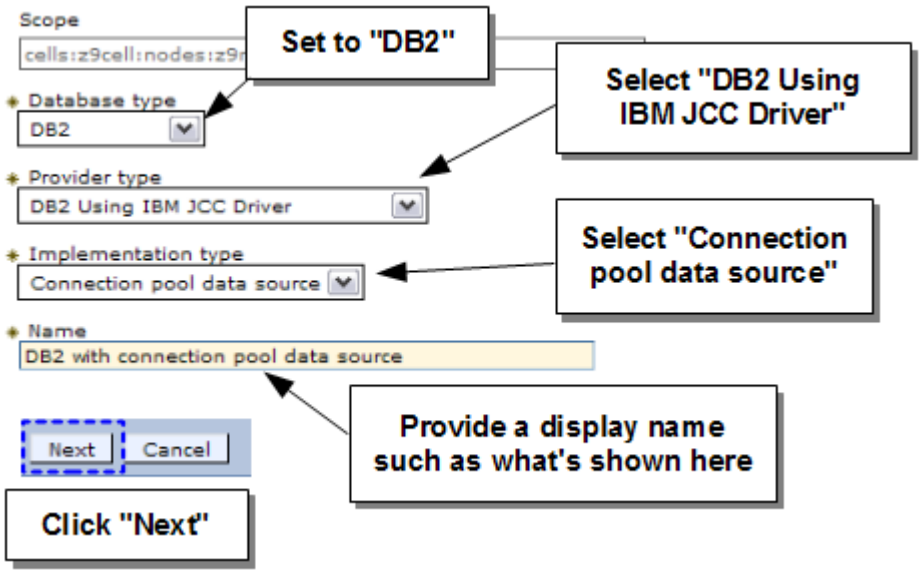
Note: This section will demonstrate three things: (1) manual failover and failback using the MODIFY command; (2) automatic failover upon detection of loss of DB2 with failback upon recovery; and (3) demonstration of some additional functions that are WAS z/OS-exclusive.

- ☐ In TSO `=SDSF.LOG`, check to see if DB2 is up with command `/D A,L`. You should see four address spaces that start with "DSNX". If not up, start with `/-DSNX START DB2`.
- ☐ In the Admin Console, go to *Resources* ➔ *JDBC* ➔ *JDBC providers*.
- ☐ Set the "Scope" pulldown to that of the z9sr01a server:



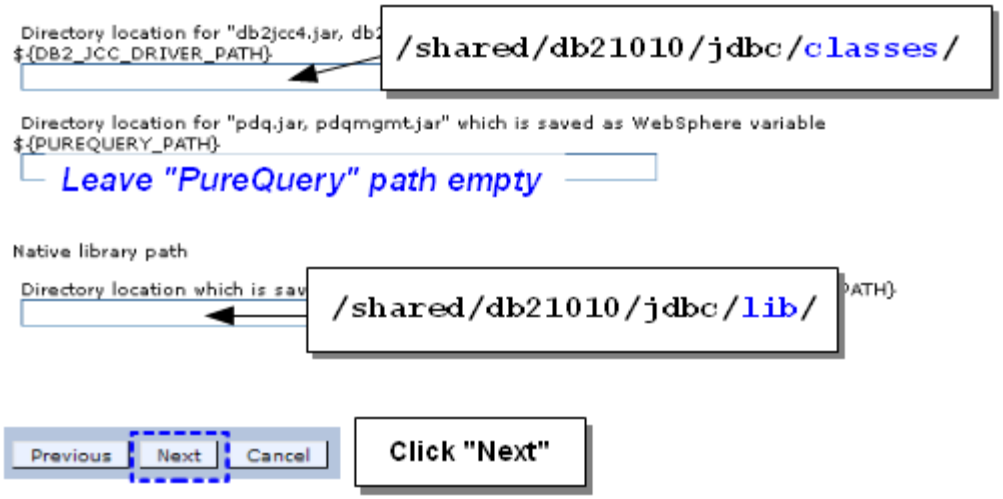
Normally JDBC provider resources are scoped at the "Node" level. But scoping at the server level will allow the "Test Connection" button to work[30]. That's the *only* reason we're scoping to the server. There is *no* technical reason to scope it to the server for the functions we're about to demonstrate.

- ☐ Click the "New" button, then do the following:

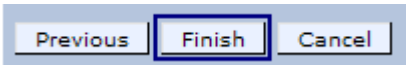


30 If scoped to the "node", WAS attempts to issue the Test Connection request from the *Node Agent*. That is a server with no servant region. So the Test Connection fails.

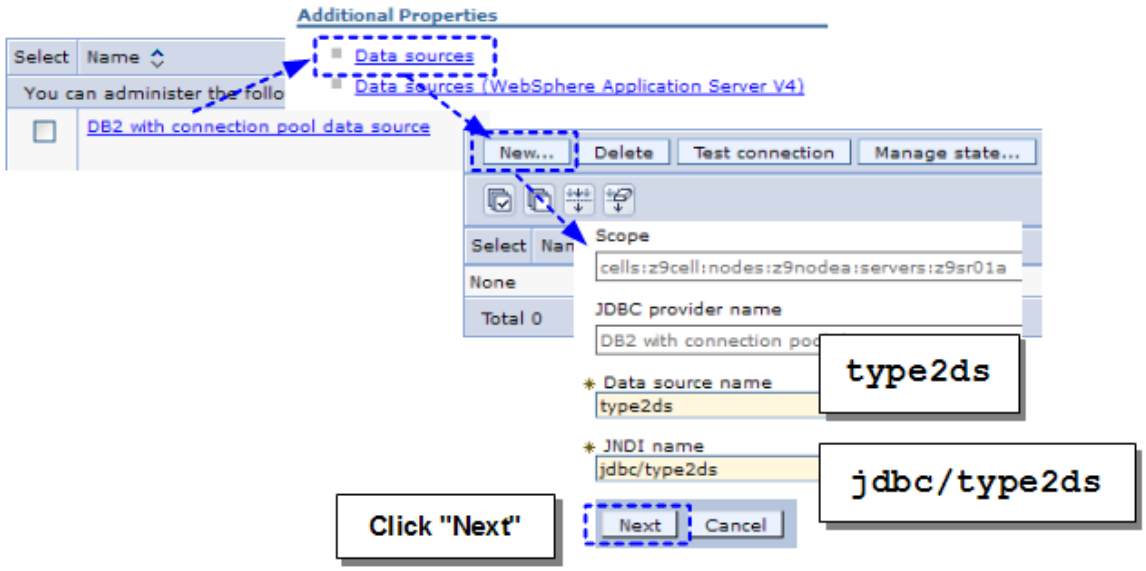
- Then:



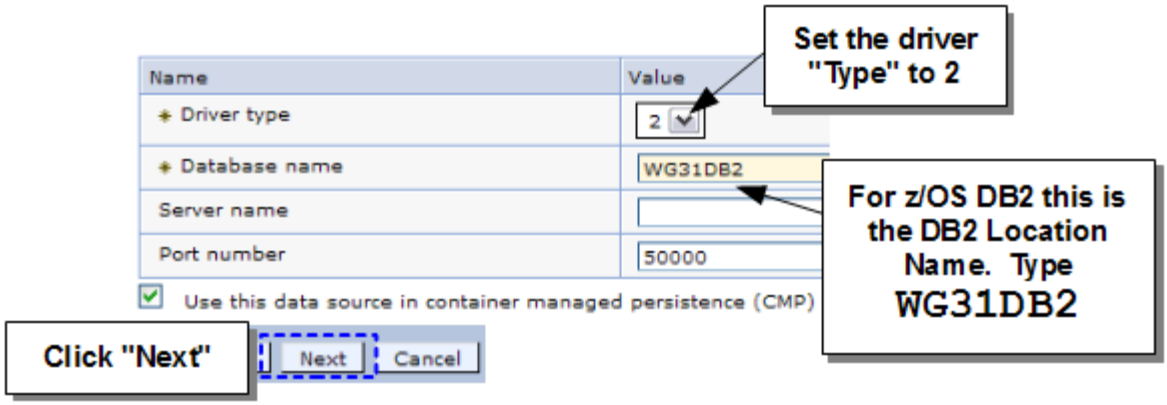
- At the summary panel, click "Finish":



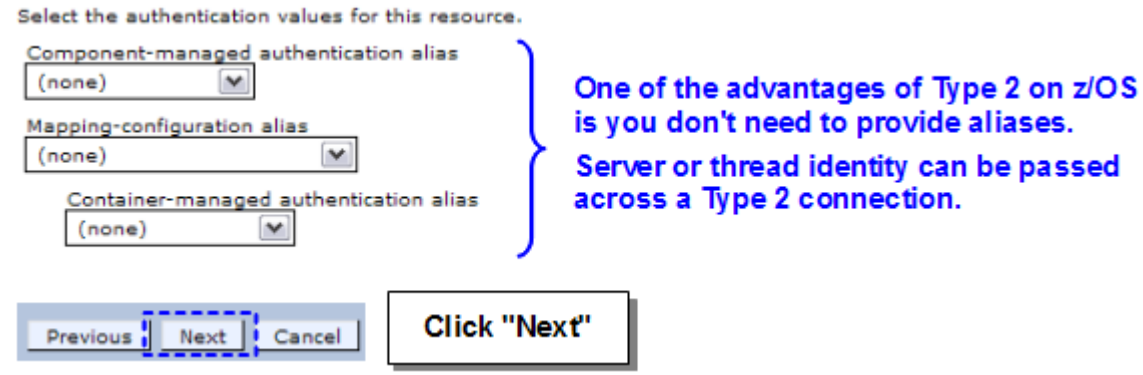
- You should see your new JDBC provider definition. Click on that link and do the following:



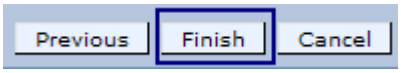
- Then:



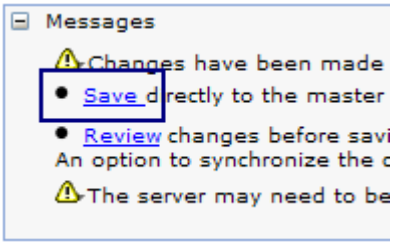
- And then:



- And at the summary panel, just click "Finish"



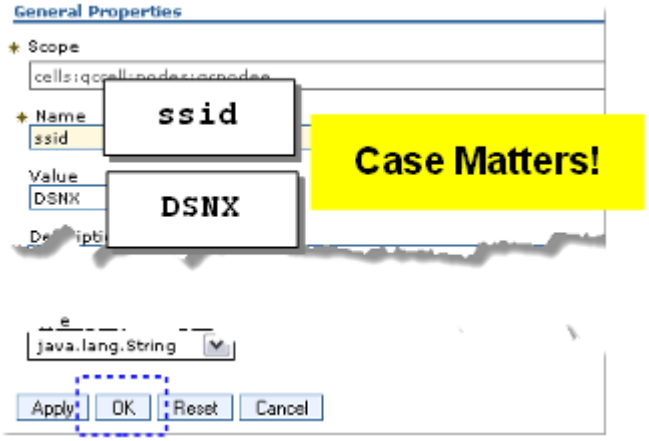
- Save and synchronize your changes:



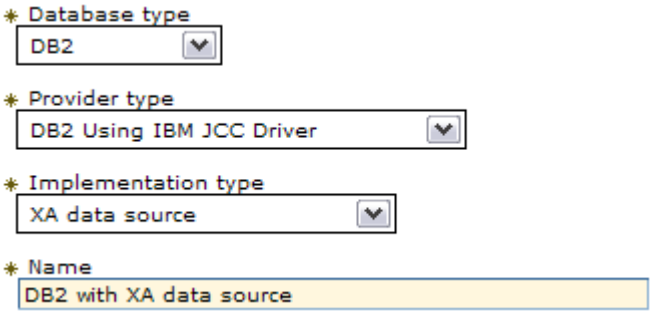
- Go to *Resources* ➔ *JDBC* ➔ *JDBC providers* ... you should now see your new provider:

- Click on that provider link, then click on "Data Sources," then click on the `type2ds` data source link, then under "Additional Properties" click on "Custom Properties."

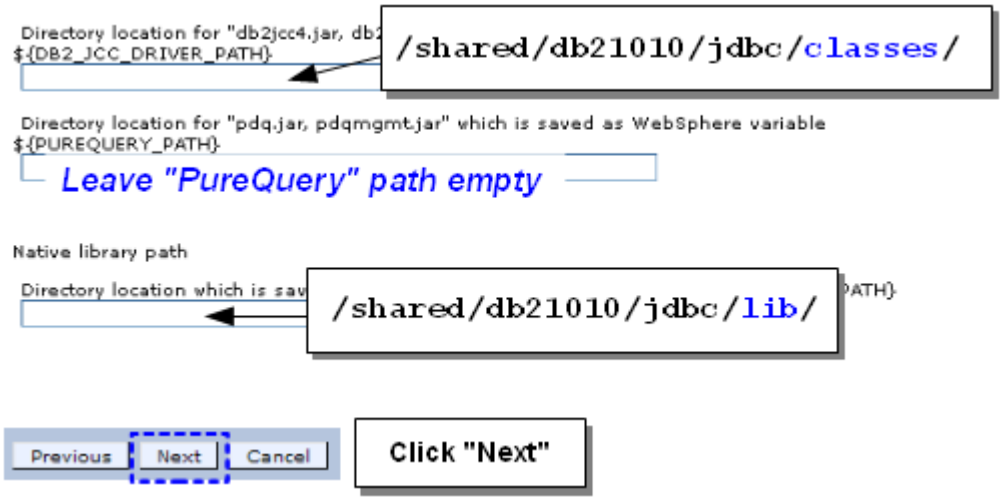
- ☐ Click the "New" button and add a custom property[31] with a name of `ssid` and a value of `DSNX`, then click "OK":



- ☐ Save and synchronize the changes.
- ☐ Now create a *second* provider and a *second* data source[32] using following few pictures to guide you. Click the "New" button, then set the provider information:



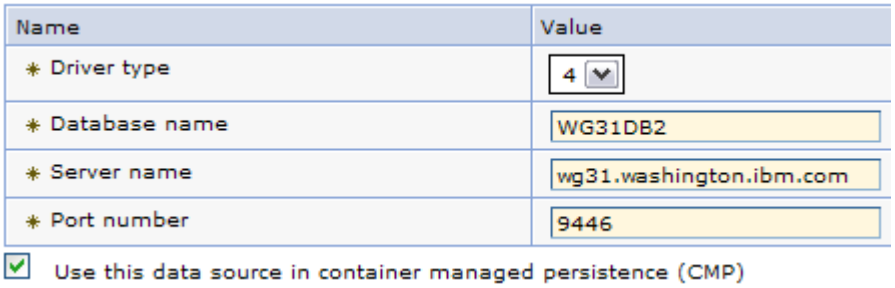
- ☐ The classpath and libpath information is the same as the first provider:



31 This tells WAS what DB2 subsystem ID to connect to with the Type 2 connection.
32 The panels will be identical; only the information entered will be different.

- ☐ Click through and finish. When you see your new provider listed, click on the link and create a *new* data source. Provide a unique name and JNDI name:

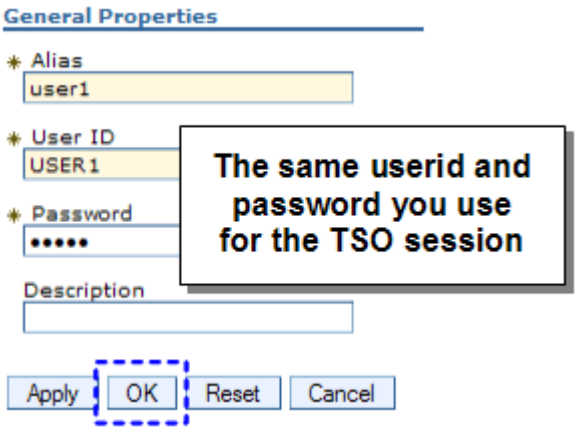
- ☐ Then set the specifics for the Type 4 connection to DB2:



Note: In this case our backup data source is pointing to the same DB2 subsystem on the same LPAR as the primary. In a real-world setting the Type 4 would more likely point to DB2 on another LPAR. This is a workshop lab environment ... *imagine* it's another LPAR. ☺

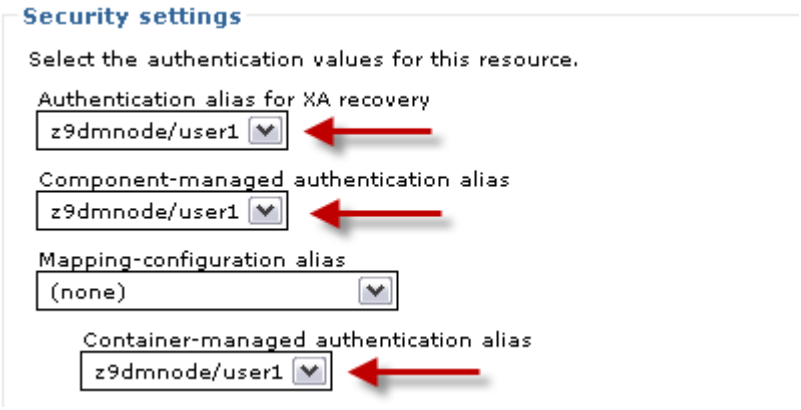
- ☐ When you get to the security aliases panel, leave all the pulldowns blank for now. Just click "Next" and then click "Finish".
- ☐ You should see your new data source in the list. Click on that data source link.
- ☐ Next, click on the "JAAS - J2C authentication data" link:

- ☐ Click the "New" button, then fill in the information for the authentication alias and click "OK":



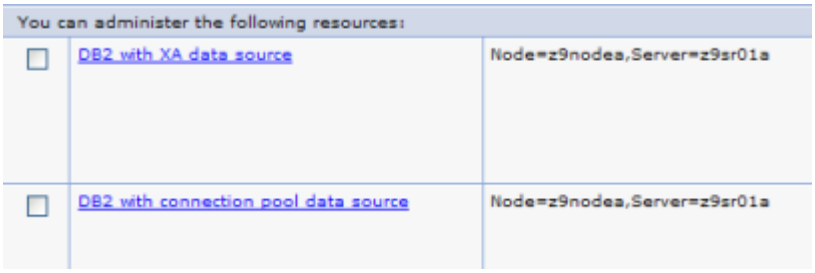
- ☐ You'll see your new alias in a list. Look to the top of that page and find the navigation path and click on "type4ds":

- ☐ On the resulting panel, scroll down until you see "Security settings" and use the drop-down lists to populate the fields as shown:

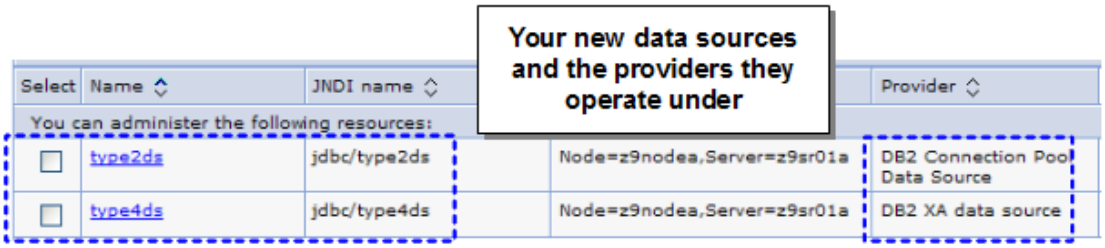


  Then click "OK".

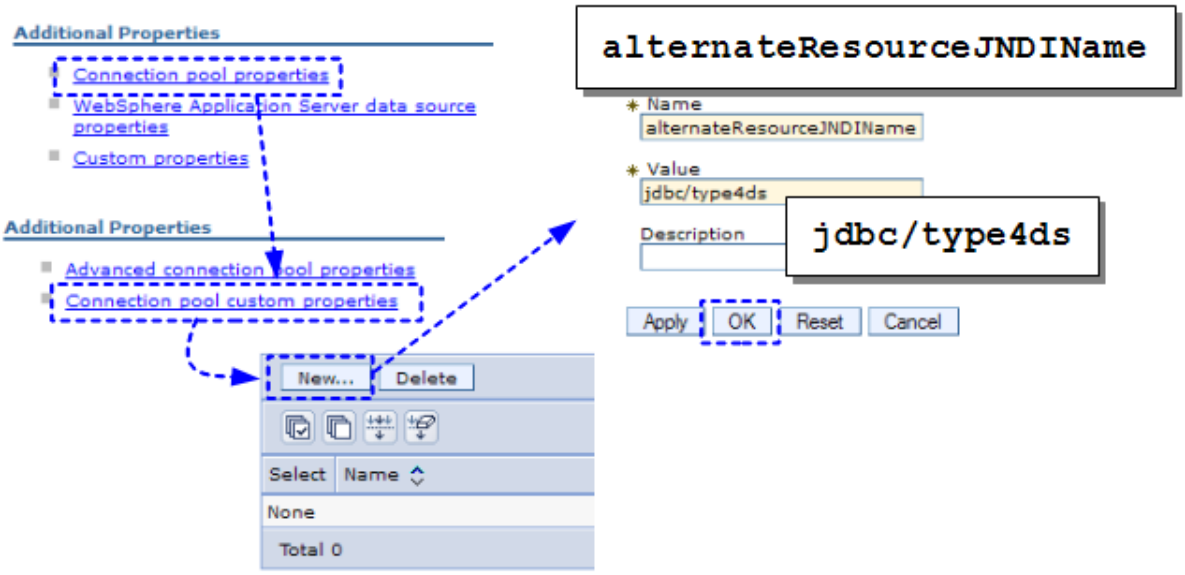
- ☐ Save and synchronize all the changes.
- ☐ Go to *Resources* ➔ *JDBC* ➔ *JDBC providers* ... you should now see both of your providers:



- ☐ Go to *Resources* ➔ *JDBC* ➔ *Data sources* ... you should now see both of your data sources as well as a few others:



- ☐ Click on the **type2ds** data source link, then do the following:



Note: A new property in V8. This tells WAS what JDBC data source to use (`jdbc/type4ds`) in the event the primary one (`type2ds` in this case) fails. We are illustrating local Type 2 as the primary with a alternate of a Type 4 connection across TCP.

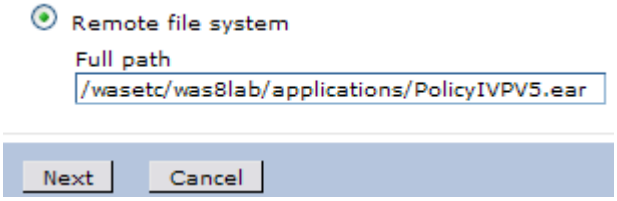
- ☐ Save and synchronize the changes.
- ☐ Start the `z9sr01a` server. Give that a few moments to start.
- ☐ When it comes active, go to *Resources* ➔ *JDBC* ➔ *Data sources* ...then select the `type2ds` checkbox and click the "Test connection" button[33]. You should see:

- ☐ Repeat with the `type4ds`. You should see the same result. You're now ready to proceed to the manual failover and failback testing.

Install PolicyIVP JDBC application

- ☐ In the Admin Console, go to *Applications* ➔ *New Application* ➔ *New Enterprise Application.*
- ☐ Click on the "Remote file system" radio button, then type in:

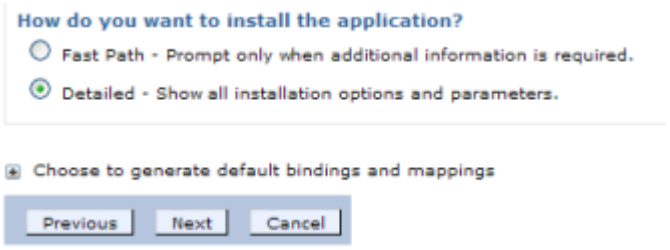
`/wasetc/was8lab/applications/PolicyIVPV5.ear`



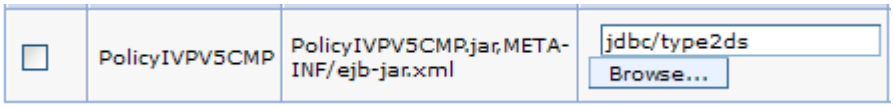
and click "Next."

33 See "Example of Test Connection Failure when Scope=Node" on page 86 for an example of the failure you see when the scope is at the node level.

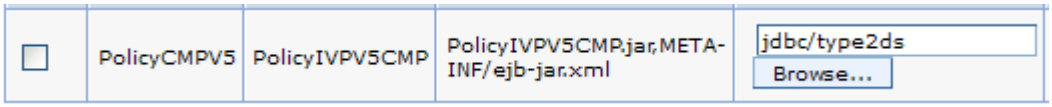
- ☐ Then select the "Detailed" radio button and click "Next":



- ☐ You'll see a long list of steps, for which most the defaults work well. But some you'll need to modify. Click on "Step 9" and then type or use the browse button to set the JNDI name to `jdbc/type2ds` which you set up earlier:



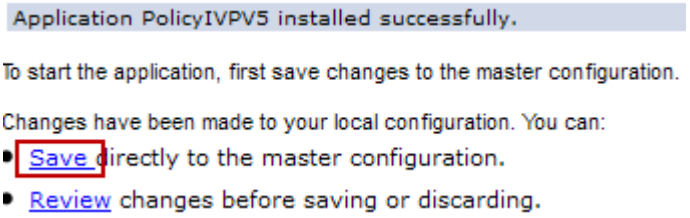
- ☐ For "Step 10" do the same type of thing:



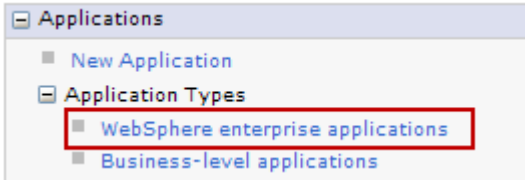
- ☐ For "Step 11" do the same again:

??? This application has two entity beans. Step 9 provided a default JNDI mapping for the entity beans, Step 10 mapped the Container Managed Persistence (CMP) bean directly, and Step 11 mapped the Bean Managed Persistence (BMP) bean directly.

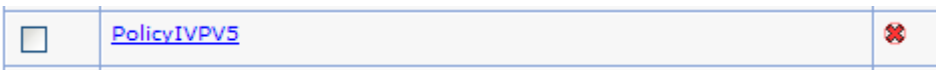
- ☐ Click on "Step 17 - Summary" then click the "Finish" button.
- ☐ Click the "Save" link to save/synchronize the application installation changes:



- ☐ Now go back and display the list of all installed applications:



You should see this new application present with a status of "red X":



- ☐ Select the box next to the application name and click the "Start" button. You should see the status go to "green arrow". You may need to click the "refresh" icon to properly reflect the status:

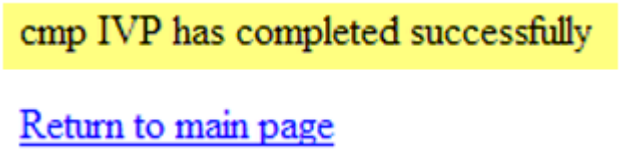
Eventually you should see "green arrow":

- ☐ Now perform a very simple validation of this application. Point your browser at the following URL:

`http://wg31.washington.ibm.com:10067/PolicyIVPV5Web/cebit.html`

You should see a very large splash page with a picture of the earth:

- ☐ Enter some integer values into both input boxes. What the numbers are doesn't matter, but they do have to be integers. Then click the "Submit" button.
- ☐ Check that you received the "completed successfully" message[34]:



- ☐ Click on the "Return to main page" link, populate the fields with some integer values (same as before or new), change the radio button from "CMP" to "BMP" and click submit. Again, check for the "completed successfully" message.

Manual failover testing using MODIFY command

- ☐ From the `=SDSF.LOG` *command extension* (enter single slash), issue the following command:

```
F Z9SR01A,FAILOVER,'jdbc/type2ds'
```

You should see the following messaging indicating success:

```
BBOO0211I MODIFY COMMAND FAILOVER COMPLETED SUCCESSFULLY
```

34 If you don't see "completed successfully" there's something wrong. Check the servant region output for clues.

- Now when you drive the PolicyIVP application it should use the defined alternate JNDI name you defined on the `jdbc/type2ds` data source back on page 44. Go and re-drive the PolicyIVP application, then go to the *servant* output. You should see something like this:

```
Trace: 2011/11/04 15:34:36.595 02 t=9CB938 c=UNK key=P8 tag= (13007004)
  SourceId: com.ibm.ws.rsadapter.spi.WSRdbDataSource
  ExtendedMessage: BBOO0222I: DSRA8208I: JDBC driver type  : 4
```

  That validates that in this simple test the alternate resource was in fact used.

- Now again from `=SDSF.LOG` *command extension*, issue the following command:

  `F Z9SR01A,FAILBACK,'jdbc/type2ds'`

  You should see the following messaging indicating success:

```
BBOO0211I MODIFY COMMAND FAILBACK COMPLETED SUCCESSFULLY
```

- Drive the PolicyIVP application again. It will use the original `jdbc/type2ds` data source, but proving that is a bit more difficult with simple single-invocation tests[35].

- Open JMeter (if it's not already up) and load the `WBSR8.jmx` test case. Set the SuperSnoop and MyIVT thread group user counts each to **`0`**.

- Set the PolicyIVP thread group user count to **`1`**. Then start the testcase (*Run* ➔ *Start*).

  Note: You may be tempted to set the user count higher, but please don't. The JMeter test case has the policy numbers hard-coded in the request. If you have more than one user thread running you'll encounter DB2 locking contention as two users try to update the same policy record simultaneously. JMeter is capable of using dynamic parameter input ... we just haven't coded it to do that.

- Go to `=SDSF.ENC` and hit the "enter" key periodically to refresh the screen. You should *occasionally* (not all the time) see something like this:

```
NAME             SSType Status   SrvClass Per
4800004F34       CB     ACTIVE   CBCLASS    1
2400000002       STC    INACTIVE OPS_LO     1
2000000001       STC    INACTIVE SYSTEM     1
```

  Keep hitting "enter" until you see something like that. The point is you should *not* see an enclave related to DDF. That means the JDBC traffic is all Type 2. There's no DDF work being done. The enclave created by WAS is propagated into DB2 because it's Type 2.

- Leave the JMeter test case running. From the command extension issue the command:

  `F Z9SR01A,FAILOVER,'jdbc/type2ds'`

  You should see the following messaging indicating success:

```
BBOO0211I MODIFY COMMAND FAILOVER COMPLETED SUCCESSFULLY
```

- Go back to `=SDSF.ENC` and periodically hit enter ... you should eventually see:

```
NAME             SSType Status   SrvClass Per
5000005951       CB     ACTIVE   CBCLASS    1
5400005953       DDF    ACTIVE   DB_DDF     1
2400000002       STC    INACTIVE OPS_LO     1
2000000001       STC    INACTIVE SYSTEM     1
2800000003       STC    INACTIVE SYSSTC     1
2C00000004       TCP    INACTIVE SYSOTHER   1
```

35 One way would be to use =SDSF.ENC and watch the creation of enclaves. With JDBC T4 you would see the creation of DDF enclaves. But they come and go quickly so it would be hard to catch it with one-off manual tests. But with JMeter we would be able to see them better. So that's what we'll use.

The "DDF" enclave means the flow into DB2 is using Type 4. You still see the CBCLASS service class information because JMeter is still driving WAS. But that enclave is *not* propagated into DB2 because Type 4 is in use. Instead, a *new* enclave is created when the request hits the DB2 listener service (`DSNXDIST`).

- ☐ Time to fail it *back*. From the command extension issue the command:

  `F Z9SR01A,FAILBACK,'jdbc/type2ds'`

  You should see the following messaging indicating success:

  ```
  BBOO0211I MODIFY COMMAND FAILBACK COMPLETED SUCCESSFULLY
  ```

- ☐ If you go back to `=SDSF.ENC` you should see the CBCLASS-related enclaves, but no DDF-related enclaves. You've reverted back to using Type 2.

- ☐ Stop the JMeter test case (*Run* ➔ *Stop*).

Lesson:

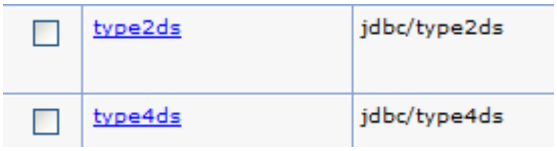
What you just did illustrated *manual* failover and failback, which would be useful for *planned* outages of local DB2 resources.

Next we'll illustrate *unplanned* outages. We'll cancel the local DB2 and watch as WAS detects the loss and automatically fails over to the alternate JNDI.

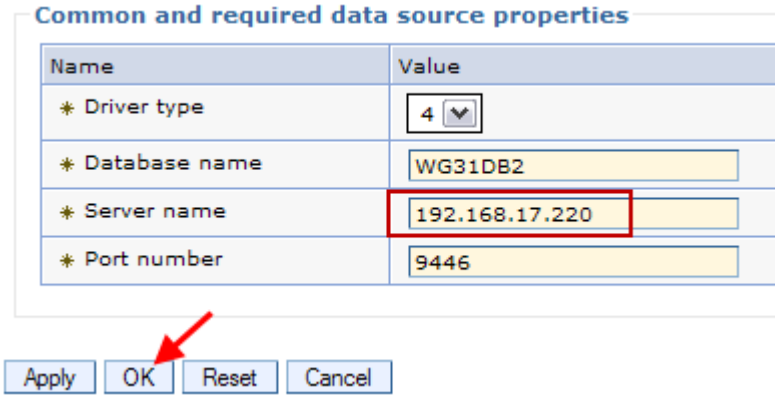
We only have one DB2 subsystem on each team's guest z/OS system. So what we'll do is modify the Type 4 data source and point it to a unused but operational workshop guest with DB2. Your work will flow there.

Automatic failover and failback

- ☐ In the Admin Console go to *Resources* ➔ *JDBC* ➔ *Data sources*. Set the "scope" equal to the `z9sr01a` server. You should see your two data sources displayed:



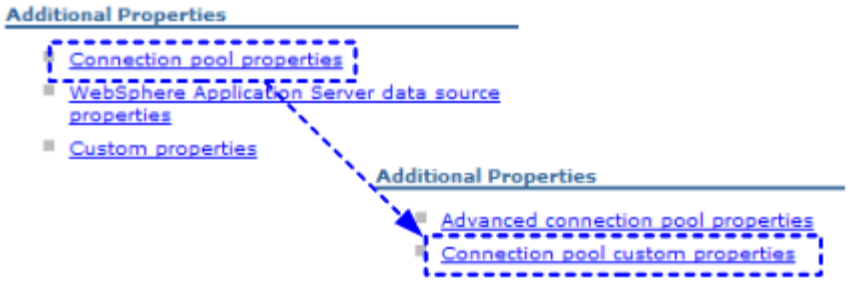
- ☐ Click on the `type4ds` data source. Scroll to the bottom and change the "Server Name" field so it points to **`192.168.17.220`** rather than `wg31.washington.ibm.com`:



Then click "OK" and save and synchronize the change.

??? We're about to have you kill your instance of DB2. We're simulating failover to another LPAR in a Sysplex, but in this case it's a different z/OS system. The "220 System" is one of the guest z/OS systems on our z/VM LPAR where DB2 is running.

- ☐ When synchronization completes, check the box next to `jdbc/type4ds` and then click on the "Test Connection" button. You should see the indication of success.
- ☐ Now click on the **`jdbc/type2ds`** data source link. You're going to add a few more connection pool custom properties.
- ☐ Just as you did before, drill into the custom properties:



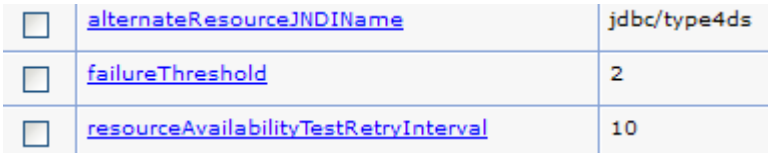
You should see your one custom property created earlier:

- ☐ Using the "New" button, create *two* new custom properties:

Name	*Value*	*Notes*
`failureThreshold`	**2**	1
`resourceAvailabilityTestRetryInterval`	**10**	2

Notes:
(1) Indicates how many failed attempts to the primary data source are needed to trigger the automatic failover to the alternate. Here you're indicate two failed attempts.
(2) Indicates how frequently (in seconds) WAS should poll the failed primary to see if it is back available. In this case you're indicating 10 seconds.

When you're done it should look like this:



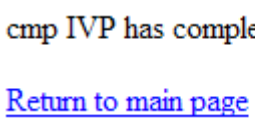
- ☐ Save and synchronize the changes.
- ☐ Stop and *restart* the `z9sr01a` server to pick up those custom properties.

> **Note:** We won't use JMeter for this because with so many reqeusts flowing so rapidly things go by too quickly. You'll miss the key messages indicating detection outage and detection of recovery. So we'll do things manually so we can observe these things in action.

- ☐ Point your browser at the following URL:

  `http://wg31.washington.ibm.com:10067/PolicyIVPV5Web/cebit.html`

- ☐ Enter some integer value for each policy number then click the "Submit" button. You should see an indication of success on the results screen:



At least one successful connection to the primary is required before failover will take place.

- From `=SDSF.LOG`, issue the following command to bring down DB2 quickly,

  `/-DSNX STOP DB2 MODE(FORCE)`

  This will simulate an abrupt failure of DB2. Give it a moment or two to come down.

- Go back to the browser and drive a request. You should get a failure indicator on the browser screen

  IVP has failed

  as well as stack traces in the servant region.

  This is failure number 1 within the threshold value you set of 2.

- Drive the browser request again. You'll get another failure on the browser and another set of stack traces in the servant output.

  This is failure number 2 within the threshold value you set of 2, which should trigger the automatic failover to the alternate JNDI. You should see this in the servant output:

```
Trace: 2011/11/04 17:42:32.018 02 t=9C8C68 c=UNK key=P8 tag= (13007004)
  SourceId: com.ibm.ws.rsadapter.spi.WSRdbDataSource
  ExtendedMessage: BBOO0222I: DSRA8208I: JDBC driver type  : 4
```

  That indicates it has automatically failed over to the alternate data source.

- Go back to the browser and drive another request. This time you should see success, both on your browser screen and in your servant output.

  Note: Your local DB2 is shut down. If you're seeing success then it must mean you're going somewhere for that JDBC connection. The only knowledge your WAS has of other DB2 resources is in your `jdbc/type4ds` data source. And that's what's named as the alternative JNDI resource when the local is deemed unreachable based on the threshold number.

- From `=SDSF.LOG`, issue the following command to restart your local DB2:

  `/-DSNX START DB2`

- Wait about 30 or 45 seconds for two things to happen: (1) DB2 to come back up, and (2) WAS to recognize that the local DB2 is back up. Recall you set the polling interval to 10 seconds.

  Watch the servant output for this message, indicating the local resource is back:

```
Trace: 2011/11/04 17:43:22.216 02 t=9CCA48 c=UNK key=P8 tag= (13007004)
  SourceId: com.ibm.ejs.j2c.DataSourceConnectionFactoryFailoverTimer
  ExtendedMessage: BBOO0222I: J2CA0682I: The configured resource with a
  JNDI name of jdbc/type2ds is available to process new requests for
  the resource with a JNDI name of jdbc/type2ds.
```

- From the browser drive another request. This should be successful and will be using your `jdbc/type2ds` local connection to the local DB2 resource.

Additional z/OS function related to availability enhancements

Note: What happens if **both** the primary *and* alternate data resources are unreachable? Then you need some actions of last resort. That's what we'll look at now ... three exclusive z/OS functions:

(1) Issue a message but take no other action -- useful with system automation tools

(2) Stop the listeners on the application servers that have applications that use resources that have failed -- useful so front-end routing function may then route around the problem

(3) Stop the applications that are using resource references where the backend resource has failed -- this allows other applications not using the failed resources to continue

- ☐ Go into the `jdbc/type4ds` data source and change the "Server name" value from `192.168.17.220` back to **`wg31.washington.ibm.com`**. This eliminates any viable backup DB2, which enables this "last resort" function to take effect.
- ☐ Go back to the `jdbc/type2ds` data source connection pool custom properties. *Add* the following property to those already present:

Name	*Value*
`failureNotificationActionCode`	**`1`**

- ☐ Save and synchronize the changes.
- ☐ Stop and restart the server to pick up the changes.
- ☐ *Important:* from the browser, invoke PolicyIVP once more to show that it works using `jdbc/type2ds` to the local copy of DB2.
- ☐ From `=SDSF.LOG`, issue the following command to bring down DB2 quickly:

  `/-DSNX STOP DB2 MODE(FORCE)`

- ☐ From the browser, invoke PolicyIVP twice (the threshold level). You'll see failures each time[36].
- ☐ Go to `=SDSF.ST`, set `PREFIX=Z9` and then put a question mark (`?`) next to the `Z9SR01A` ***controller*** region.
- ☐ Then put an **s** next to `HRDCPYDD` and hit enter. Scroll to the bottom and you should see:

```
BBOJ0130I: CONNECTION MANAGEMENT IN A SERVANT REGION DETECTED THAT THE
RESOURCE IDENTIFIED BY JNDI NAME jdbc/type2ds IS DISCONNECTED FROM SERVER
z9cell/z9nodea/Z9SR01/z9sr01a. ACTION TAKEN: NONE.
```

  Note: That's it ... just this message. Nothing else when the value of the action code is "1".

- ☐ Restart DB2 with this command:

  `/-DSNX START DB2`

- ☐ Go back to the `HRDCPYDD` output in the controller and after the 10 second polling interval you should see:

```
BBOJ0131I: RESTORATIVE ACTION IS BEING TAKEN FOR THE RESOURCE
IDENTIFIED BY JNDI NAME jdbc/type2ds ON SERVER
z9cell/z9nodea/Z9SR01/z9sr01a, REASON=1. ACTION TAKEN: NONE.
```

  Note: Again, that's it ... just a message. But this does indicate that WAS detected the resumption of DB2. With these messages and system automation you may program other actions to take in the event of an outage. Admittedly this is a very simple action code. More is coming next.

- ☐ Change the `failureNotificationActionCode` value from `1` to `2`.
- ☐ Save and synchronize
- ☐ Stop and restart the server.
- ☐ Important: successfully drive PolicyIVP once.
- ☐ Shut down DB2 with **`/-DSNX STOP DB2 MODE(FORCE)`**

36 You'd see a failure even on invocation 3 or beyond since we changed the `jdbc/type4ds` data source back to the local DB2, which is now down. The `failureNotificationActionCode` function only invokes if there is *no* access to a data resource. Had we left that data source pointing to `192.168.17.220` the failover would have been successful and what we're trying to achieve with this lab would never appear.

- ☐ Drive PolicyIVP twice with failures to trigger the theshold value of 2. In the `HRDCPYDD` of the *controller* region you should see:

```
BBOJ0130I: CONNECTION MANAGEMENT IN A SERVANT REGION DETECTED THAT THE
RESOURCE IDENTIFIED BY JNDI NAME jdbc/type2ds IS DISCONNECTED
FROM SERVER z9cell/z9nodea/Z9SR01/z9sr01a. ACTION TAKEN: PAUSING LISTENERS.
BBOO0222I: ZAIO0002I: z/OS asynchronous IO TCP Channel TCP_1 has stopped
listening on host * port 10065.
BBOO0222I: ZAIO0002I: z/OS asynchronous IO TCP Channel TCP_3 has stopped
listening on host * port 10066.
BBOO0222I: ZAIO0002I: z/OS asynchronous IO TCP Channel TCP_2 has stopped
listening on host * port 10067.
BBOO0222I: ZAIO0002I: z/OS asynchronous IO TCP Channel TCP_4 has stopped
listening on host * port 10068.
```

Note: Many front-end routing devices -- the WAS Plugin included -- will key off the lack of a listener port to conclude that work should no longer be routed there. That's the purpose of this function ... to make the server with the lost backend resource "go dark" so front-end routers may then begin routing *around* the problem.

- ☐ Open a TeraTerm or PuTTY session to your system and log with ID Z9ADMIN. Issue the following command:

  `netstat | grep Z9SR01A`

  You should see a listing of the ports for that server, but you will *not* see ports 10065 - 68 in the list:

```
Z9SR01A  0000CCD1 127.0.0.1..37692       127.0.0.1..10029       Establsh
Z9SR01A  0000CCBB 0.0.0.0..10070         0.0.0.0..0             Listen
Z9SR01A  0000CCBE 127.0.0.1..10069       0.0.0.0..0             Listen
Z9SR01A  0000CCD4 127.0.0.1..37693       127.0.0.1..10029       Establsh
Z9SR01A  0000CCAE 0.0.0.0..10064         0.0.0.0..0             Listen
Z9SR01A  0000CCC1 192.168.17.211..10070  192.168.17.211..37687  Establsh
Z9SR01A  0000CCAD 0.0.0.0..10063         0.0.0.0..0             Listen
Z9SR01A  0000CCBD 0.0.0.0..10062         0.0.0.0..0             Listen
Z9SR01A  0000CCC4 192.168.17.211..10070  192.168.17.211..37688  Establsh
Z9SR01A  0000CCCF 0.0.0.0..10039         *..*                   UDP
```

- ☐ Restart DB2 with **`/-DSNX START DB2`** ... give it 30 or so seconds then check the `HRDCPYDD` of the controller region. You should see:

```
BBOJ0131I: RESTORATIVE ACTION IS BEING TAKEN FOR THE RESOURCE IDENTIFIED
BY JNDI NAME jdbc/type2ds ON SERVER z9cell/z9nodea/Z9SR01/z9sr01a, REASON=1.
ACTION TAKEN: RESUMING LISTENERS.
BBOO0222I: ZAIO0001I: z/OS asynchronous IO TCP Channel TCP_1 is listening
on host * port 10065.
BBOO0222I: CHFW0019I: The Transport Channel Service has started
chain WCInboundAdmin.
BBOO0222I: ZAIO0001I: z/OS asynchronous IO TCP Channel TCP_2 is listening
on host * port 10067.
BBOO0222I: CHFW0019I: The Transport Channel Service has started
chain WCInboundDefault.
BBOO0222I: ZAIO0001I: z/OS asynchronous IO TCP Channel TCP_3 is listening
on host * port 10066.
BBOO0222I: CHFW0019I: The Transport Channel Service has started
chain WCInboundAdminSecure.
BBOO0222I: ZAIO0001I: z/OS asynchronous IO TCP Channel TCP_4 is listening
on host * port 10068.
```

Note: Upon detection of the return of DB2 WAS z/OS restarts the listeners. Front-end routing functions that heartbeat the ports will see the server as back and available and will begin routing to the server again.

- ☐ Change the `failureNotificationActionCode` value from `2` to `3`.
- ☐ Save and synchronize
- ☐ Stop and restart the server.
- ☐ Important: successfully drive PolicyIVP once.
- ☐ Shut down DB2 with **`/-DSNX STOP DB2 MODE(FORCE)`**
- ☐ Drive PolicyIVP twice with failures to trigger the theshold value of 2. In the `HRDCPYDD` of the controller region you should see:

```
BBOJ0130I: CONNECTION MANAGEMENT IN A SERVANT REGION DETECTED THAT THE
RESOURCE IDENTIFIED BY JNDI NAME jdbc/type2ds IS DISCONNECTED FROM SERVER
z9cell/z9nodea/Z9SR01/z9sr01a. ACTION TAKEN: STOPPING APPLICATIONS THAT
USE THIS RESOURCE.
```

- ☐ Go to *Applications* ➔ *Application Types* ➔ *WebSphere enterprise applications*. There you should see a listing of the deployed applications and the status of each. You should see something like this:

Note: The downside to `PAUSELISTENERS` is it affects all the applications in the server. This action seeks to remedy that by stopping only those applications that use the JNDI reference for the failed backend resource. In our case that's just `PolicyIVPV5`.

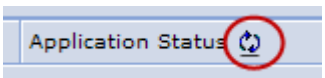
A note of caution -- many front-end routing functions do not "see" the Java EE state of the applications. So it's possible this will allow those routing functions to route work to a stopped application, which will result in error messages being returned. More intelligent front-end functions such as the WAS Proxy Server or the WAS On Demand Router *are* aware of the Java EE application state.

- ☐ Restart DB2 with **`-DSNX START DB2`** ... give it 30 or so seconds then check the `HRDCPYDD` of the controller region. You should see:

```
BBOJ0131I: RESTORATIVE ACTION IS BEING TAKEN FOR THE RESOURCE IDENTIFIED
BY JNDI NAME jdbc/type2ds ON SERVER z9cell/z9nodea/Z9SR01/z9sr01a,
REASON=1. ACTION TAKEN: STARTING APPLICATIONS THAT USE THIS RESOURCE.
```

- ☐ If you go back to the list of applications you'll see PolicyIVP restarted:

You may need to invoke the "refresh" twisty to see the indicator change to green arrow:

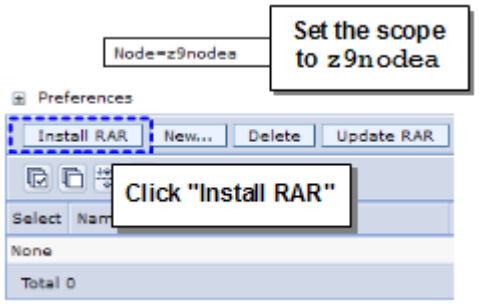


CICS

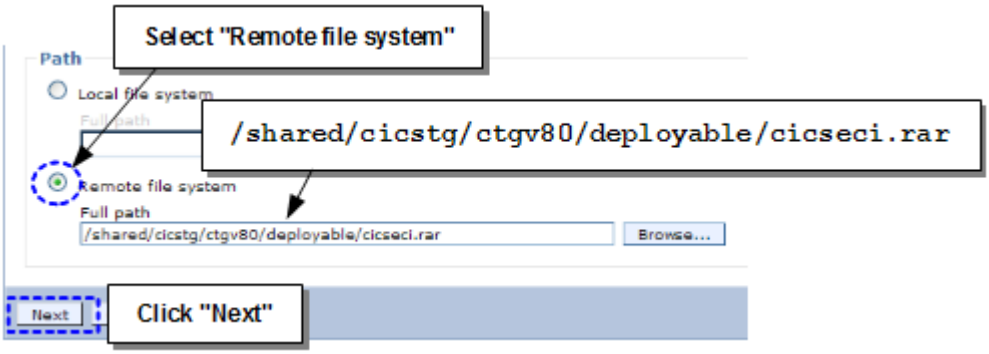
Note: The focus of the CICS-related labs is to show how to connect from WAS into CICS in three modes: (1) local EXCI; (2) using the gateway daemon task; and (3) using IPIC.

Install the resource adapter and define the connection factories

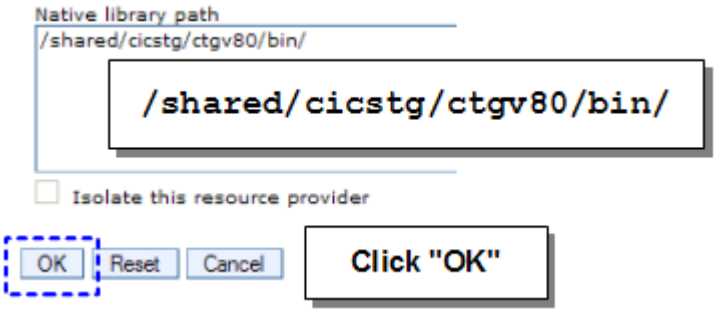
- ☐ In the Admin Console, go to *Resources* ➔ *Resource Adapters* ➔ *Resource adapters*
- ☐ Then do the following:



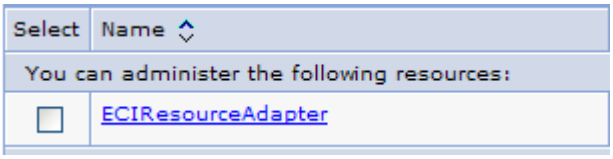
- ☐ Then point to where WAS should get the RAR file:



- ☐ On the next panel, update the "Native library path" and provide information about where the CTG shared object files are located:

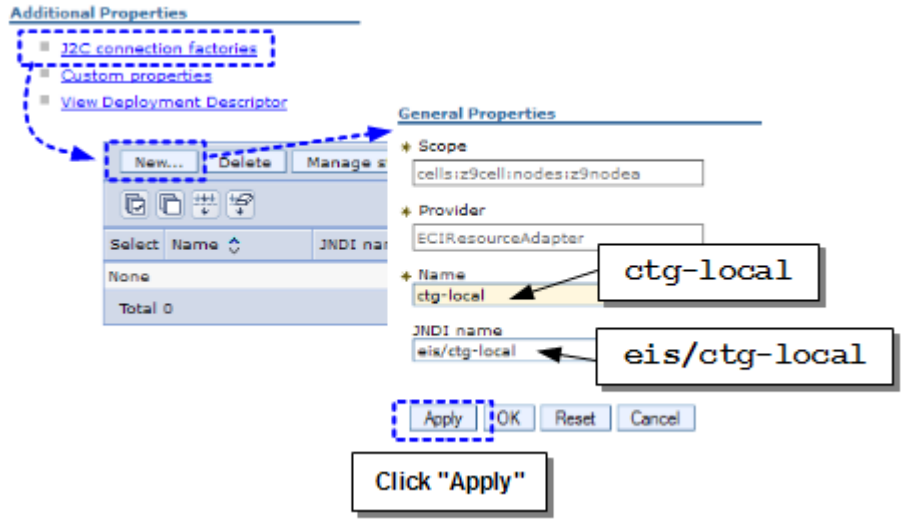


- ☐ You should the new resource adapter in the list:

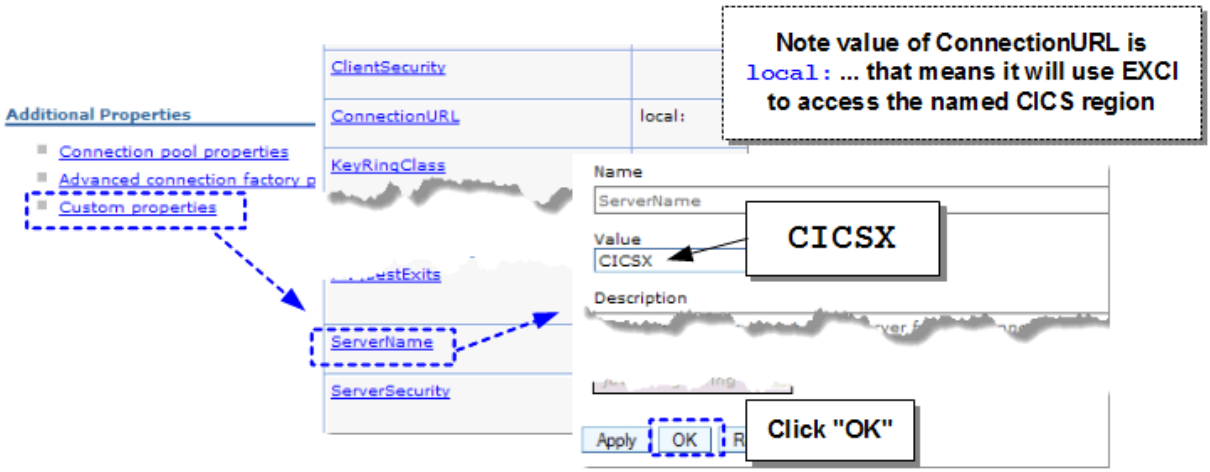


Click on that link to begin the process of defining the connection factories.

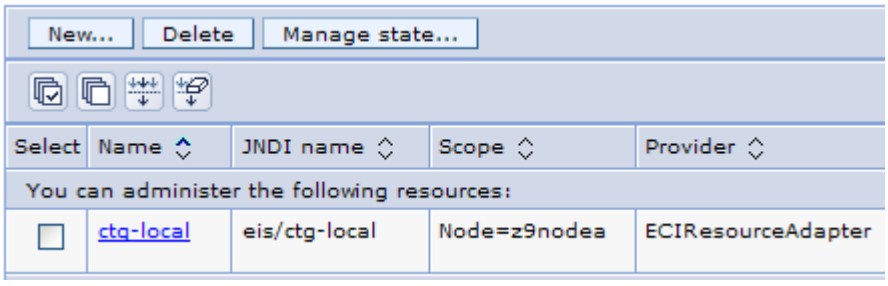
- ☐ Then do the following:



- ☐ That "Apply" un-grayed some links on that page. Now do the following to name the CICS region to connect to:

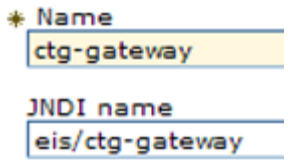


- ☐ Save and synchronize changes made to this point.
- ☐ Go back to *Resources* ➔ *Resource Adapters* ➔ *Resource adapters.*
- ☐ Click on the resource adapter link, then on "J2C connection factories"
- ☐ You should see the connection factory you just created:



Click on the "New" button to create *another* connection factory.

- ☐ Supply a name and JNDI name:



  and then click "Apply" to un-gray the links at the top right of the screen.
- ☐ Click on the "Custom properties" link
- ☐ Click on the "ConnectionURL" link and change the default value. When finished, click "OK."

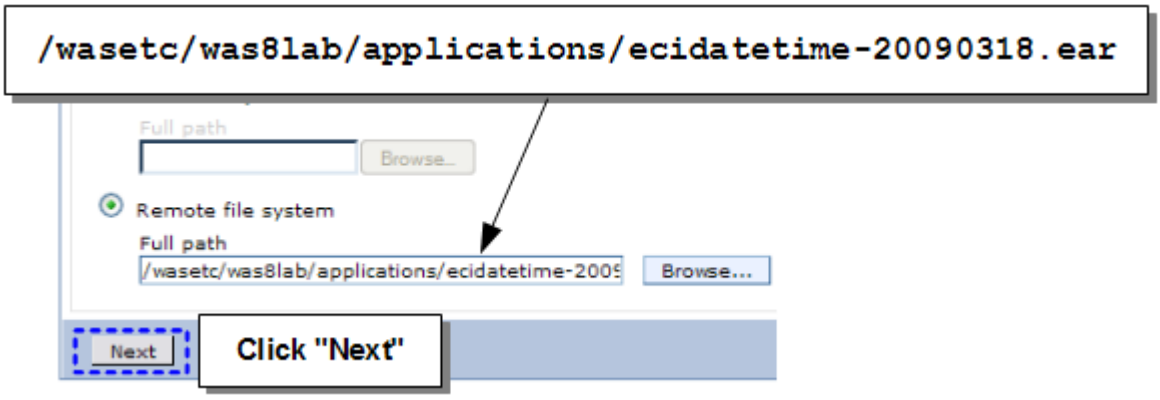
Before:	**`local:`**
After:	**`tcp://wg31.washington.ibm.com`**
Note: this tells the CICS resource adapter to send the request over TCP rather than using local EXCI.	

- ☐ Note the value of the "PortNumber" custom property. It should default to 2006. This is also the default port used by the CTG gateway daemon, which you'll start in just a moment.
- ☐ Save and synchronize all your changes.
- ☐ Go back and look at your list of connection factories under the resource adapter. You should now see both:

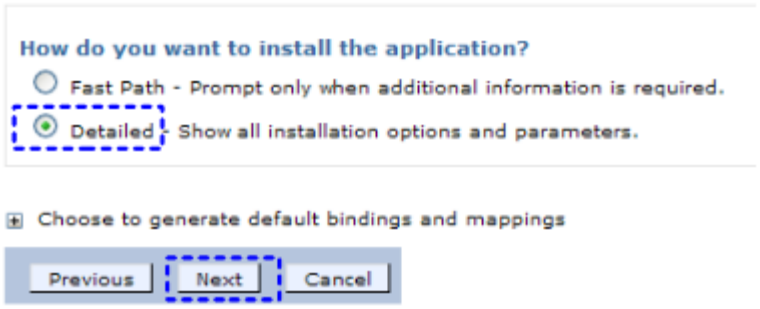
You can administer the following resources:				
☐	ctg-gateway	eis/ctg-gateway	Node=z9nodea	ECIResourceAdapter
☐	ctg-local	eis/ctg-local	Node=z9nodea	ECIResourceAdapter

Install the simple CICS sample application

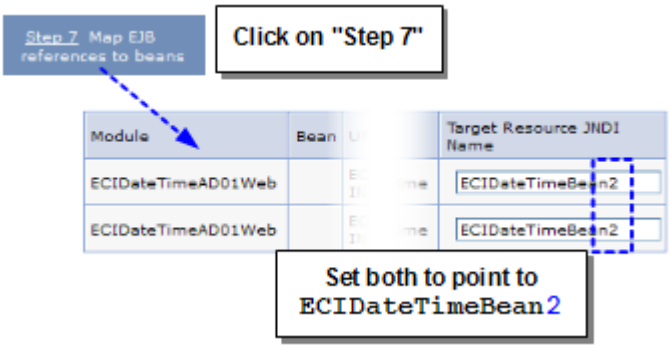
- ☐ In the Admin Console, go to *Applications* ➔ *New application* ➔ *New Enterprise Application*.
- ☐ Indicate where WAS should get the application EAR:



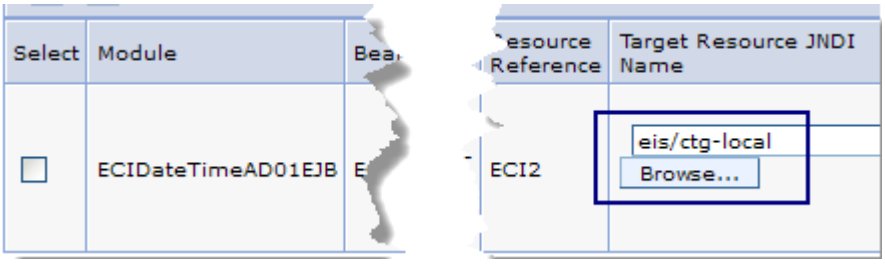
☐ Then select "Detailed" and click "Next:"



☐ Then do the following[37]:

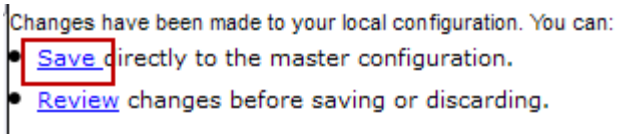


☐ Click on "Step 8" and then set the resource reference to the JNDI name of your local connection factory:



You may hand type the JNDI string or use the "Browse" button.

☐ Click on "Step 14 - Summary" then click the "Finish" button.



☐ Stop and restart the `z9sr01a` server to pick up the changes.

Test local EXCI connection to CICS

☐ Go to `=SDSF.LOG` and start the CICSX region with the z/OS command:

```
/S CICSX
```

Look for this message as an indication of success:

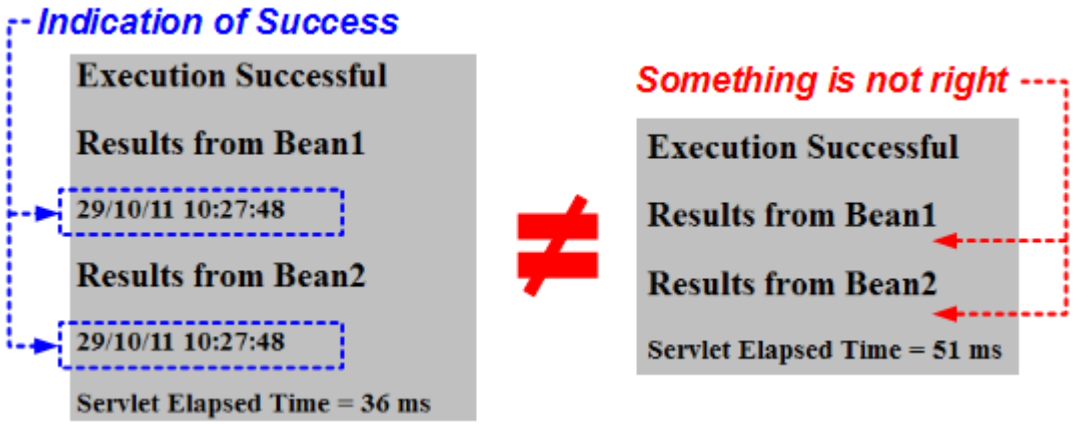
```
+DFHSI1517  CICSX   Control is being given to CICS.
```

37 An odd quirk with this application. The servlet has two EJB references and the application is *supposed* to have two target EJBs. But it really only has one. So we'll simply point both references to the same bean.

- ☐ By this time your `z9sr01a` server should be up[38]. Point your browser at:

  `http://wg31.washington.ibm.com:10067/ECIDateTimeWeb/index.html`

  Then click the "Submit" button the web page.
- ☐ If things work as they're supposed to, you should see today's date and time displayed twice:



That test used the local EXCI connection into the CICSX region. If you don't see the date and time stamps, then something isn't working properly. Check the servant output for clues.

Using IPIC

Overview The CICSX server has the necessary TCPIPSERVICE and IPCONN definitions. All you need to do is re-configure the connection factory to point to the CICS region and use it.

- ☐ Go back to the custom properties of your `ctg-local` connection factory.
- ☐ Modify the properties so the following five properties contain these values:

Applid	IPCONX
ApplidQualifier	IPCONNET
ConnectionURL	local:
PortNumber	2006
ServerName	tcp://wg31.washington.ibm.com:10099

- ☐ Save and synchronize the changes.
- ☐ Stop and restart your z9sr01a application server
- ☐ Point your browser at the application:

  `http://wg31.washington.ibm.com:10067/ECIDateTimeWeb/index.html`

  Then click the "Submit" button the web page. You should see the same sign of success you saw before.

38 Check to be sure: *Servers* ➔ *Server Types* ➔ *WebSphere application servers*. Check for green arrow.

Change application to use the gateway daemon and test that connection

- ☐ Stop the CICSX region by issuing the following command from `=SDSF.LOG`:

  `/F CICSX,CEMT PERFORM SHUTDOWN`

  Why? We wish to "prove" your application is using the `ctg-gateway` connection factory. The application is so simple no footprints are left to see where the request actually went. By bringing down `CICSX` we know the `ctg-local` CF would fail. You'll see that `ctg-gateway` works because the gateway is coded to talk to CICSY.

- ☐ When CICSX comes down, start CICSY:

  `/S CICSY`

- ☐ Start the CICS Transaction Gateway Daemon by submitting the JCL found at **`USER1.WAS.CNTL(CTGJOB)`**.

- ☐ Go to `=SDSF.LOG` and you should see the following messages:

```
CTG6400I CTGAPPLD CICS TRANSACTION GATEWAY IS STARTING
CTG6512I CTGAPPLD CICS TRANSACTION GATEWAY INITIALIZATION COMPLETE
```

- ☐ Go to `=6` and issue the command **`NETSTAT`**. You should see in that list of TCP ports the following:

```
EZZ2587I CTGJOB   00001F78 0.0.0.0.2006
```

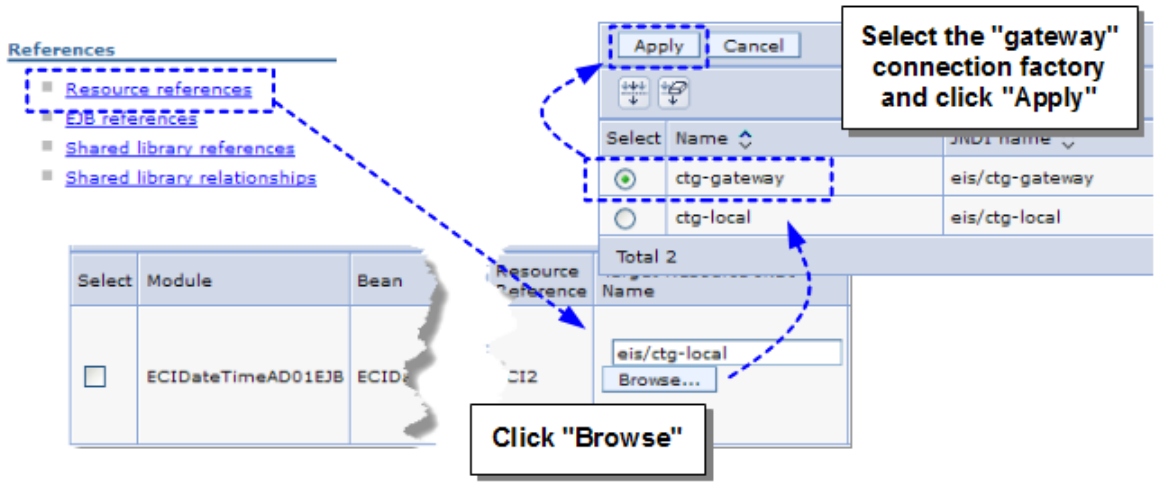
That shows that the Gateway Daemon is listening on its default port of `2006`. And if you have a *really good memory* ☺ you'll recall the following default custom property in the connection factories you built:

PortNumber	2006

Next you're going to change the application to use the ctg-gateway connection factory, which has `ConnectionURL=tcp://wg31.washington.ibm.com` and `PortNumber=2006`. In other words, the application will communicate with the Gateway Daemon to access CICS.

- ☐ In the Admin Console, go to *Applications* ➔ *Application Types* ➔ *WebSphere enterprise applications*. Click on the `ECIDateTimeAD01` application link.

- ☐ Then do the following:



- ☐ You should then see the resource reference panel again, but this time updated with JNDI name for the "gateway" connection factory:

- ☐ Save and synchronize the change. WAS will automatically stop and restart the *application* to pick up the change.
- ☐ In your browser, re-invoke the URL:

  `http://wg31.washington.ibm.com:10067/ECIDateTimeWeb/index.html`

  and again, click the "Submit" button.
- ☐ You should see the same success as before -- the date and time stamps. But this time the access path was through the CTG Gateway Daemon and into CICSY.

Cleanup

- ☐ Go to `=SDSF.LOG` and issue the following commands to clean up the CICS environment:

  `/F CICSY,CEMT P SHUTDOWN`

  `/P CTGJOB`

MQ

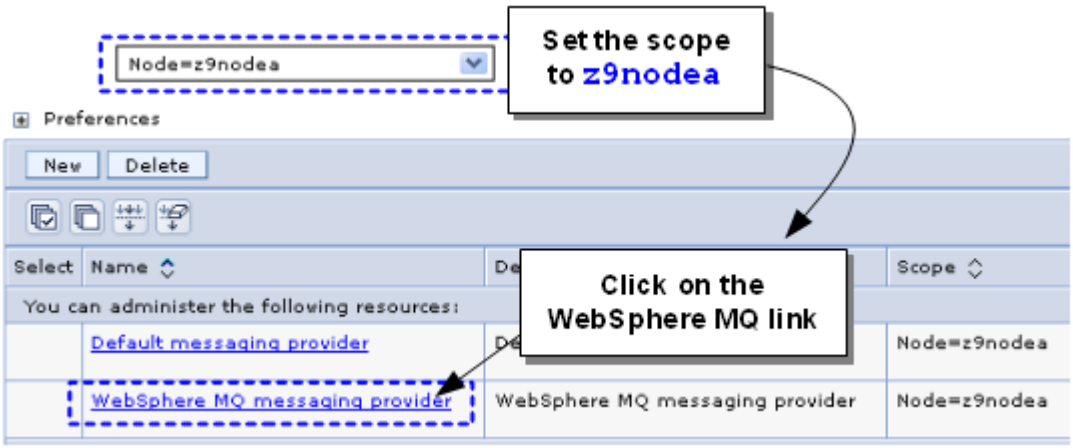
Note: This will be a relatively short lab section -- create the JMS definitions to connect to the local MQ queue manager, then drive a simple application that puts message on the MQ queue, then pulls it back. This demonstrates connectivity to MQ using WAS's JMS support.

Start MQ

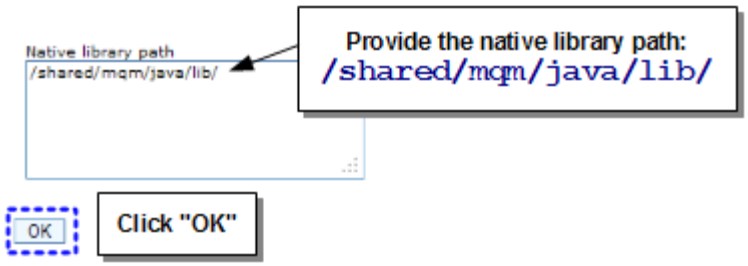
- ☐ From `=SDSF.LOG`, enter the command **`/-MQS1 START QMGR`**

Create JMS definitions for MQ queue manager and MQ queue

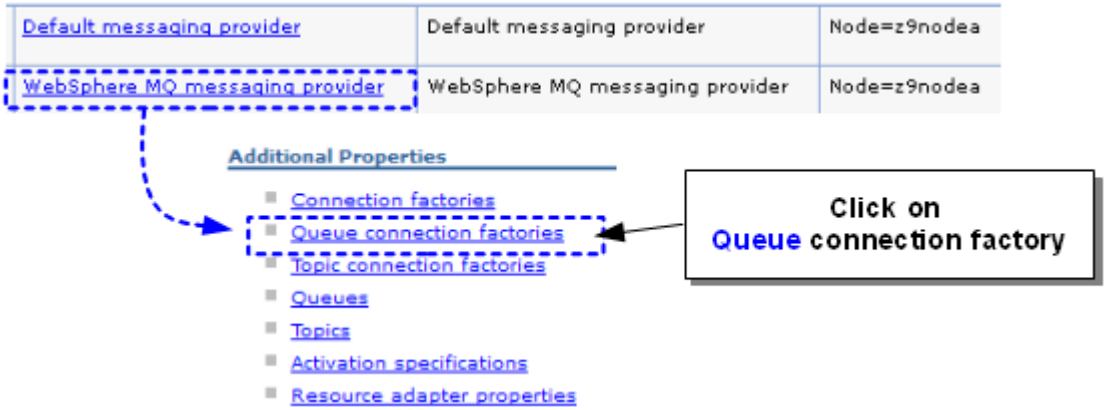
- ☐ In the Admin Console, go to *Resources* ➔ *JMS* ➔ *JMS providers*.
- ☐ Then:



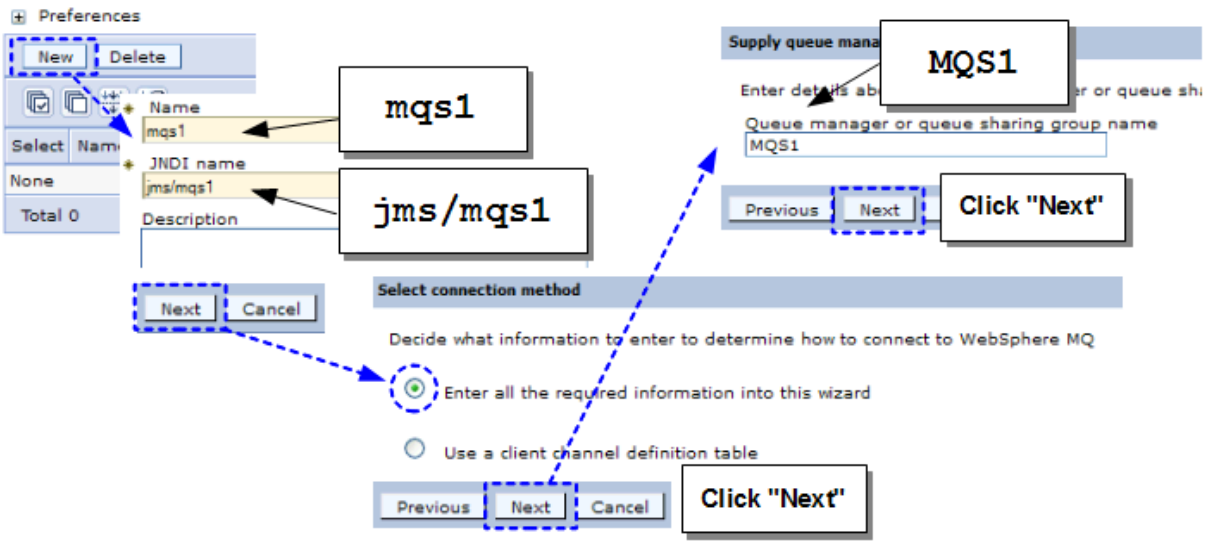
- ☐ Then do the following:



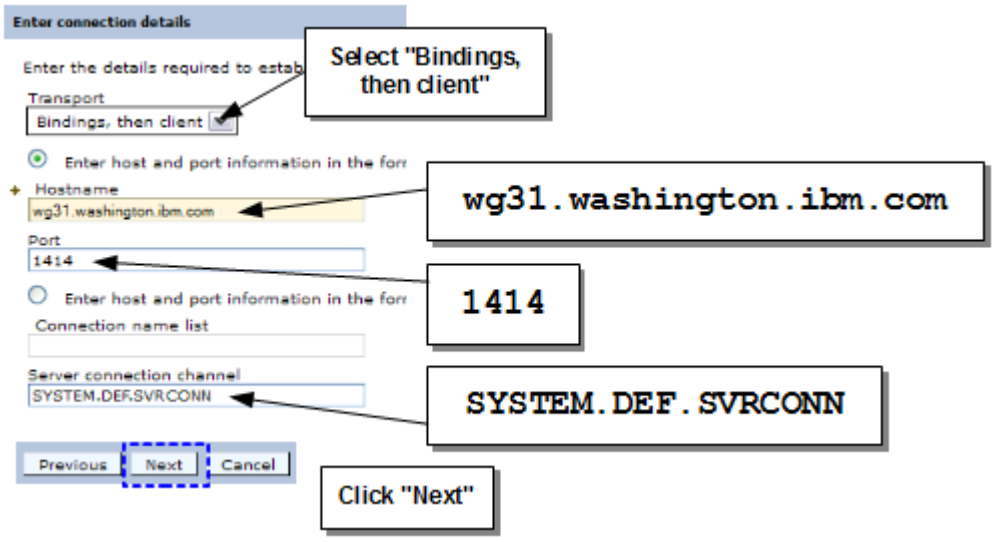
- ☐ That'll take you back to the list of JMS providers at the node level. Do the following:



- ☐ Then:



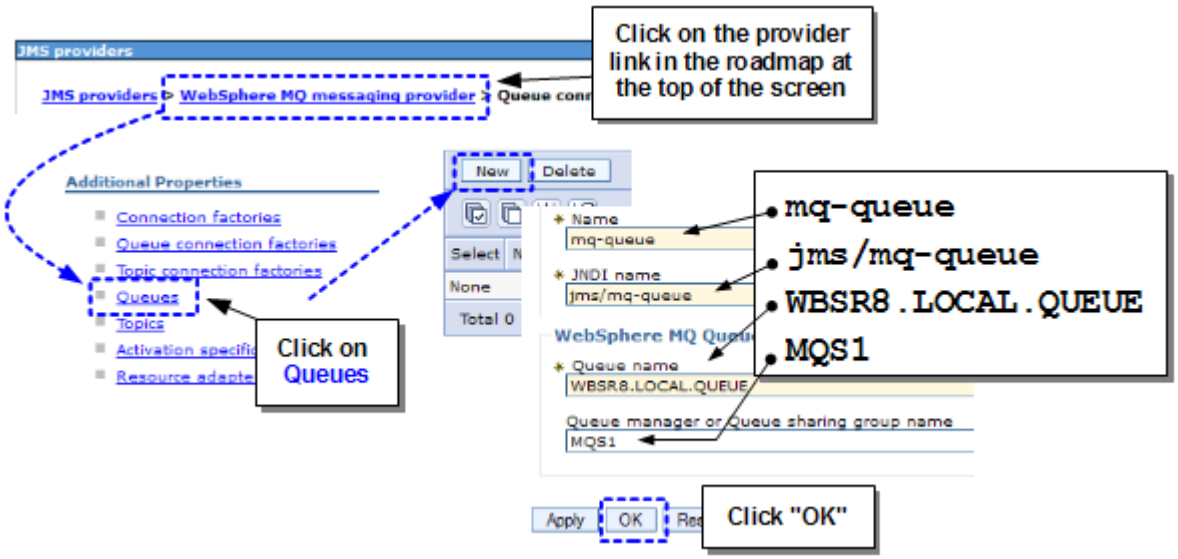
- ☐ Then provide information about how to access the MQS1 queue manager:



- ☐ The next panel has a "Test Connection" button. Give it a try.
- ☐ Click "Next" and "Finish" to complete the wizard. Save and synchronize.

> **Note:** The "Queue Connection Factory" defines the connection to the MQ queue manager. You have provided the information needed for both `BINDINGS` as well as `CLIENT` to the `MQS1` queue manager on your lab system.

- ☐ Next, do the following:



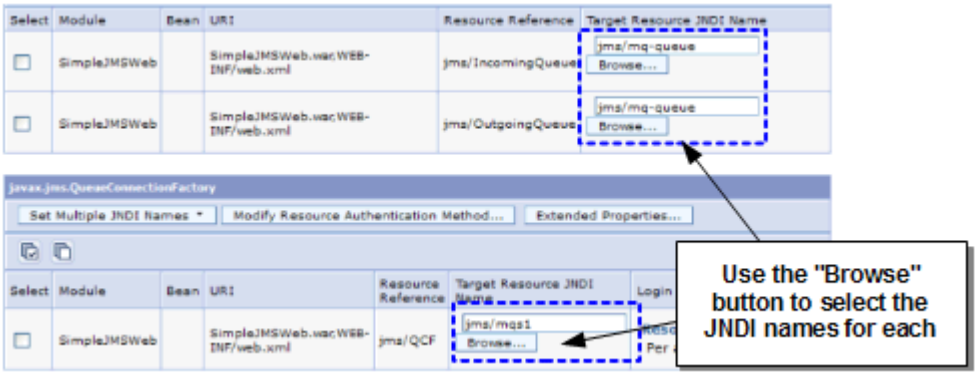
> **Note:** The "Queue" definition provides an abstraction of the real MQ queue name.

- ☐ Save and synchronize all the changes.

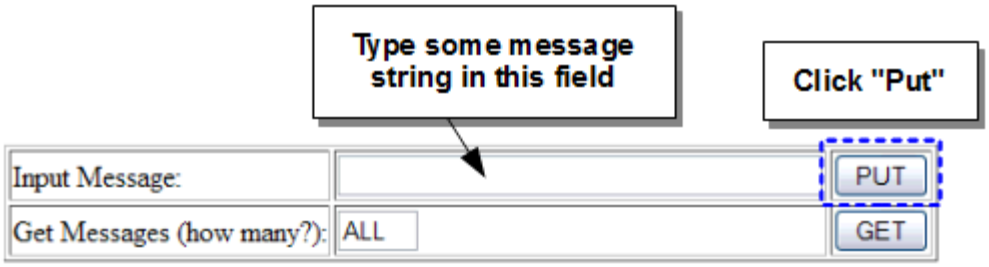
Install the sample JMS application and test connection to MQ

- ☐ In the Admin Console, go to *Applications* ➔ *New application* ➔ *New Enterprise Application*.
- ☐ Location of EAR file to install:
  `/wasetc/was8lab/applications/SimpleJMS.ear`
- ☐ Take the "Fast Path"

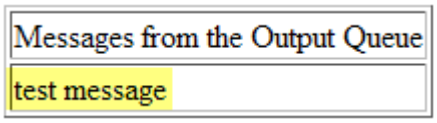
- For "Step 3" you'll see WAS calling for resolution of the resource references. The top two are for the input and output queues, which you'll map to the one queue you defined. The bottom reference is for the queue manager.



- Finish the installation and save / synchronize.
- Stop and restart the `z9sr01a` server to pick up the new definitions to MQ.
- Point your browser to the following URL:
  `http://wg31.washington.ibm.com:10067/SimpleJMS/getMessages`
- Then enter some test message into the input field:



- Then click the "Get" button and you should see:



- Try "PUT" multiple times, adding a different message each time. Then click on "GET."

Stop MQ

- From `=SDSF.LOG`, enter the command **`/-MQS1 STOP QMGR`**

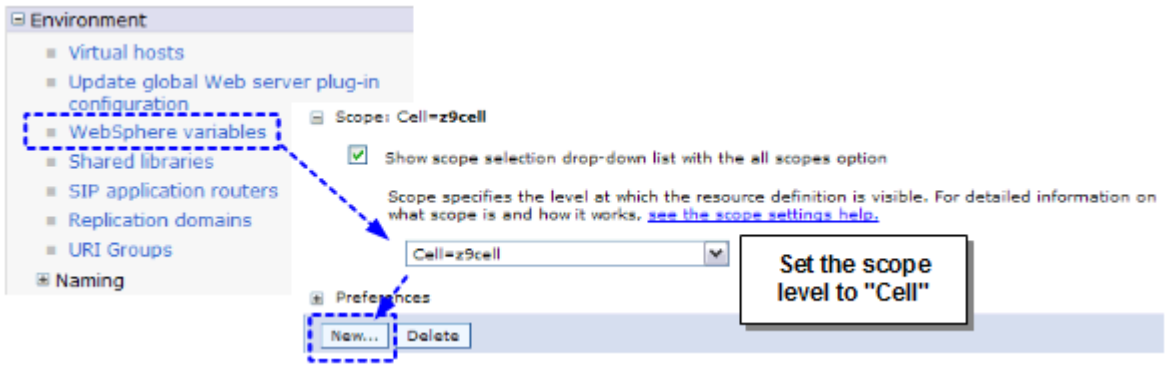
End of Unit 4 Lab

Unit 6 Lab - WebSphere Optimized Local Adapters

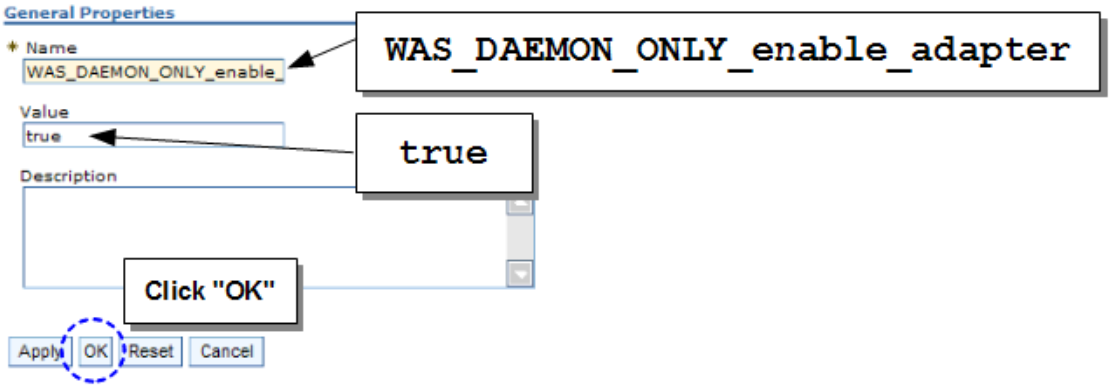
Setup of WOLA in WAS runtime

Note: This is a relatively simple process -- creation of two cell-level variables and the installation of the WOLA RAR file.

- ☐ In the Admin Console go to the environment variable panel and set the scope to "cell" and then click "New":



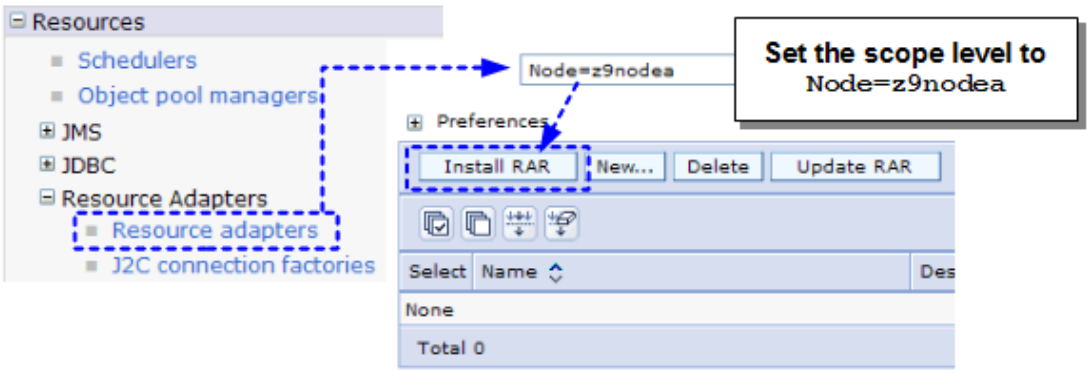
- ☐ Create the first environment variable:



- ☐ Then create the second:

Name	`ola_cicsuser_identity_propagate`
Value	`1`

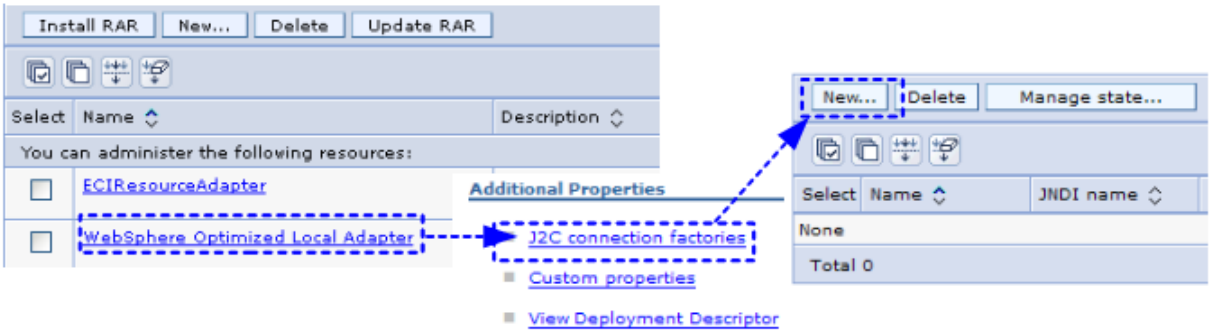
- ☐ Save and synchronize the changes.
- ☐ Begin the process of installing the resource adapter:



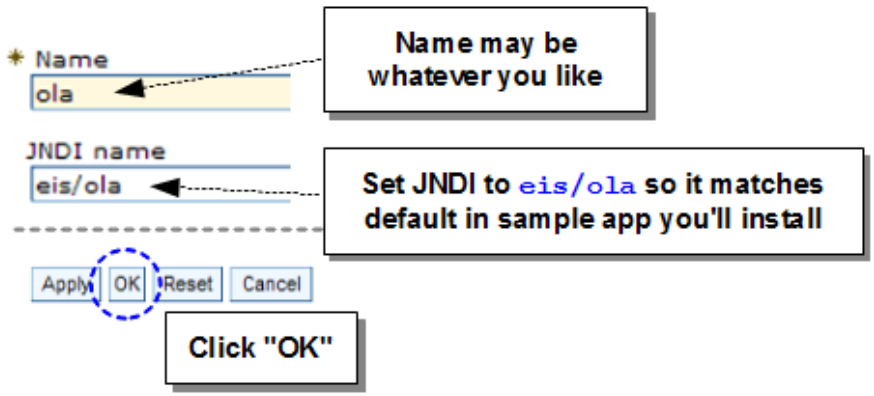
- Then ...

- On the "General Properties" panel just click OK:

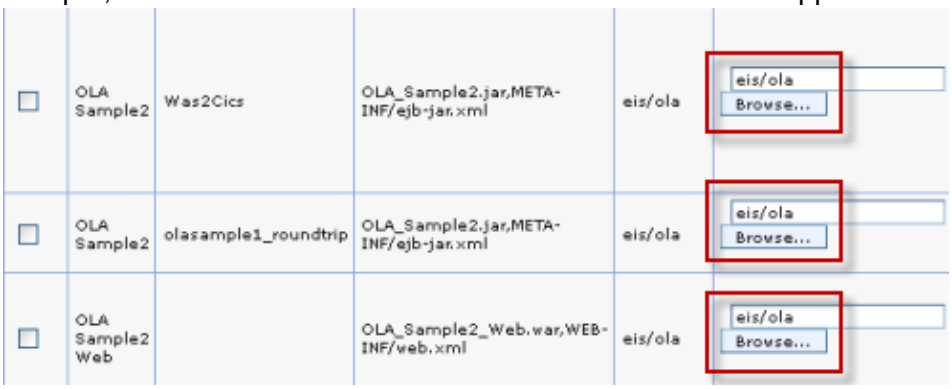
- Then:



- ☐ And on the General Properties provide a name and JNDI name[39] and click OK:



- ☐ Install the following application into your first server:
  `/shared/zWebSphere/V8R5FP02/util/zos/OLASamples/OLASample2.ear`
  - ○ Take the "Fast Path"
  - ○ For "Step 2" make sure application is mapped to server "z9sr01a" (which is default)
  - ○ For "Step 3," make sure all module "resource references" are mapped to `eis/ola`



  - ○ Take all the other defaults and finish up the application installation
- ☐ Save and ***synchronize*** your changes.
- ☐ **Important!** Shut down your *entire WAS cell* with the following MVS command: **`/P Z9DEMN`**
  This stops the Daemon, which will stop every other server in the cell on this LPAR

Setup of WOLA in CICS region CICSX

Note: We've done a little of the 3270 green screen work for you ahead of time.

- ☐ Take a look at the contents of the **`USER1.WAS8.WOLA.LOADLIB`** data set. The members in that data set are the WOLA native modules.
  We populated that `LIBRARY` data set ahead of time by running the `copyZOS.sh` shell script[40]. That shell script resides in the node's `/profiles/default/bin` directory.
- ☐ Now look at the **`USER1.WAS8.WOLA.SAMPLES`** data set. The members in that data set are the WOLA samples, including sample JCL you'll run to update CICS.
  We populated that `FB 80` data set ahead of time by running the `copyZOS.sh` shell script as well[41].

39 The JNDI may be any string. The default is `eis/ola` and we'll take that so the sample app deploy may take defaults.
40 Command: `./copyZOS.sh OLAMODS 'USER1.WAS8.WOLA.LOADLIB'`
41 Command: `./copyZOS.sh OLASAMPS 'USER1.WAS8.WOLA.SAMPLES'`

- ☐ The first job you'll run[42] is **CSDUPDAT** which updates the CICSX region's CSD. *Browse* the member, and when ready submit and insure RC=0 before continuing on.
- ☐ The next job you'll run is **DFHPLTOL** which creates a PLTPI program to start the Task Related User Exit when CICS starts. Browse the member and when ready submit and check for successful completion.
- ☐ Browse the **OLAMAP** member. This is the BMS map for the "OLAUTIL" 3270 sample application. Submit and insure successful completion.
- ☐ Browse the **OLAUTIL** member. This is the sample application that uses the OLAMAP BMS. Submit and insure successful completion.
- ☐ Browse the **OLACB01** member. This is a simple echo program that runs in CICS. **Submit** and insure successful completion.
- ☐ *Edit* the **CICSX.CICS42.SYSIN(CICSXSIP)** member and add the following line:

```
000027 SYSIDNT=CICX
000028 TCT=NO
000029 PLTPI=OL
000030 .END
```

- ☐ Edit the SYS1.PROCLIB(CICSX) member and add the following:

```
004300 //DFHRPL   DD DSN=&CICSDS..SDFHLOAD,DISP=SHR
004400 //          DD DSN=SYS1.LEMVS.SCEECICS,DISP=SHR
004500 //          DD DSN=SYS1.LEMVS.SCEERUN2,DISP=SHR
004600 //          DD DSN=SYS1.LEMVS.SCEERUN,DISP=SHR
004700 //          DD DSN=SYSS.CICS.LOADLIB,DISP=SHR
004800 //          DD DSN=SYS1.XML.SIXMLOD1,DISP=SHR
004900 //          DD DSN=CTS420.CICS.SDFHWSLD,DISP=SHR
005000 //          DD DSN=USER1.WAS8.WOLA.LOADLIB,DISP=SHR
```

- ☐ Finally, issue the following two commands from TSO Option 6 to grant the CICS region ID access to the WAS runtime:

```
PERMIT CB.BIND.Z9* CLASS(CBIND) ID(CICSX) ACCESS(READ)
SETROPTS RACLIST(CBIND) REFRESH
```

Restart the environment and validate

- ☐ Start the WAS deployment manager:

```
S Z9DCR,JOBNAME=Z9DMGR,ENV=Z9CELL.Z9DMNODE.Z9DMGR
```

- ☐ When the DMGR has initialized, check the Daemon (Z9DEMN) held output and verify the following is present[43]:

```
BBOM0001I              daemonName: Z9CELL.
BBOM0001I default_internal_work_transaction_class: NOT SET.
BBOM0001I          enable_adapter: 1.
BBOM0001I    iiop_max_msg_megsize: NOT SET, DEFAULT=0.
```

42 We updated the job with the proper high-level qualifiers for this system simply to reduce your typing.
43 This validates the cell level WAS environment variable "enable_adapter" is in effect.

- ☐ Check the DMGR's controller region (`Z9DMGR`) held output and verify the following is present[44]:

  ```
  BBOM0001I odr_tclass_propogation_enabled: NOT SET, DEFAULT=1.
  BBOM0001I ola_cicsuser_identity_propagate: 1.
  ```

- ☐ Start the node agent:

  ```
  S Z9ACRA,JOBNAME=Z9AGNTA,ENV=Z9CELL.Z9NODEA.Z9AGNTA
  ```

- ☐ In the Admin Console go to System *Administration* ➔ *Nodes*. Make sure the synchronization status for the z9nodea node shows a green circle, meaning "synchronized." If it shows a broken red circle (not synchronized), then check the node checkbox and click "synchronize."
- ☐ From the Admin Console, start the first server.
- ☐ In the Admin Console, verify the `OLASample2` application is started.
- ☐ Now start CICS:

  `/S CICSX`

- ☐ Give it a few moments, then check the CICS region's output for the following[45]:

  ```
  +BBOA9920I WAS z/OS OLA CICS PLT init start.
  +BBOA9921I WAS z/OS OLA CICS TRUE enabled.
  +BBOA9925I WAS z/OS OLA CICS PLT init ending.
  +BBOA9940I WAS z/OS OLA CICS PLT init 2 start.
  ```

  and[46]:

  ```
  Resource definition for BBOACALL has been added.
  ```

Start Link Server Task, Register into WAS and perform basic tests

Note: You'll make use of the sample application in WAS (`OLASample2.ear`), the compiled echo program in CICS (`OLACB01`) and the 3270 sample application (`OLAUTIL`).

- ☐ Click on the icon to launch the 3270 screen, then right-click the mouse and select "Clear":

- ☐ On the now blank screen, issue:

  `LOGON APPLID(CICSX)`

- ☐ At the CICS screen right mouse click and select "Clear" again.

44 This validates the cell level WAS environment variable allowing the CICS ID to propagate inbound is present.
45 These messages in CICS verifies the PLTPI (job `DFHPLTOL`)and update to the SIP member took effect.
46 The first of many messages verifying the update of the CSD (job `CSDUPDAT`)

☐ On the now blank screen, issue the following command *all on one line*[47]:

BBOC START_SRVR RGN=CICSXREG DGN=Z9CELL NDN=Z9NODEA SVN=Z9SR01A SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y

You should see:

```
BBOA8000I Start server completed successfully.
```

Note: If you get RC and RSN numbers, go to the WAS V8.5[48] InfoCenter and search on `cdat_olaapis` and look under the `BBOA1REG` API for the RC/RSN values.

If you left some of those parameter off defaults would be taken. It would still work, but you would see "Start server completed with warnings."

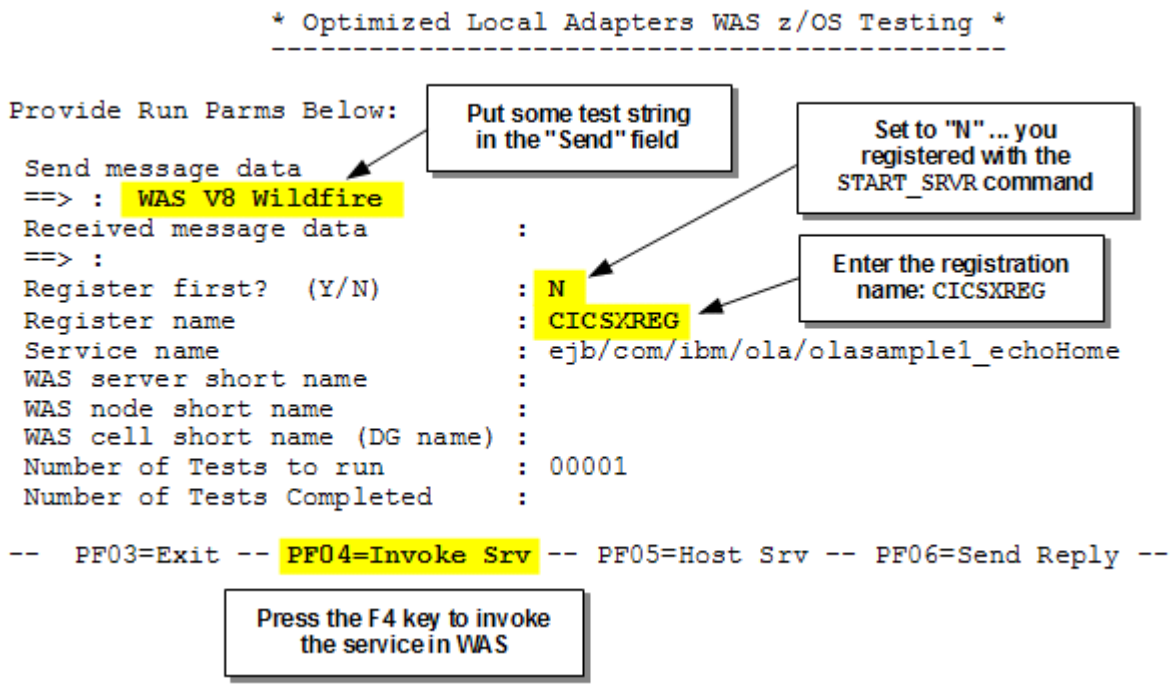
☐ Go back to the MVS console session and issue the following command:

F Z9SR01A,DISPLAY,ADAPTER,DAEMONRGES

You should see:

```
F Z9SR01A,DISPLAY,ADAPTER,DAEMONRGES
BBOA0007I: SHOWING REGISTRATIONS FOR DAEMON GROUP:
BBOA0003I: Name          Jobname  SWT TL Min  Max  Act  State
BBOA0004I: CICSXREG      CICSX    000 00 0001 0010 0001 00
BBOA0004I: $WASDEFAULT$ CICSX    000 00 0000 0001 0000 00
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,ADAPTER,DAEMONRGES
```

☐ The first test will be *inbound*, CICS ➔ WAS. Clear the CICS session screen one more time and issue the command **OLAUTIL**. When the `OLAUTIL` panel comes up, do the following:



47 Syntax documented in InfoCenter, search string `rdat_cics`. Hint: compose in Notepad first, then copy/paste into the 3270 session.

48 http://publib.boulder.ibm.com/infocenter/wasinfo/v8r5/index.jsp

You should see:

```
Received message data
==> :  WAS V8 Wildfire
```

and

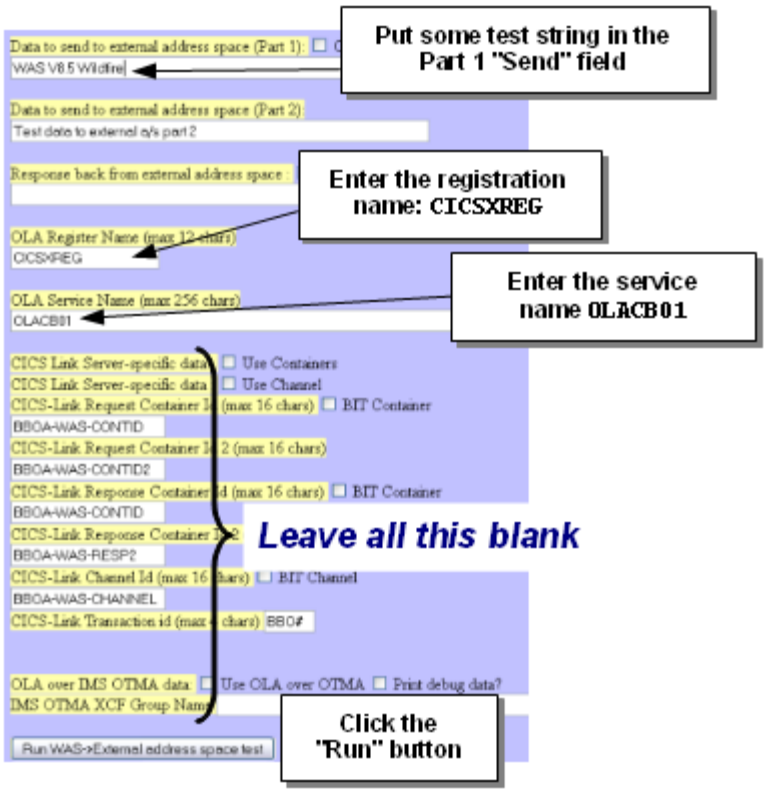
```
Invoke requested ... calls completed ok
 Start time: 10102011 220446
 Stop time : 10102011 220446 DONE Count: 00000001
```

Note: That was a single invocation of the OLASample2.ear application in WAS.

- ☐ The next test will be outbound, WAS ➔ CICS. From a browser, go to:

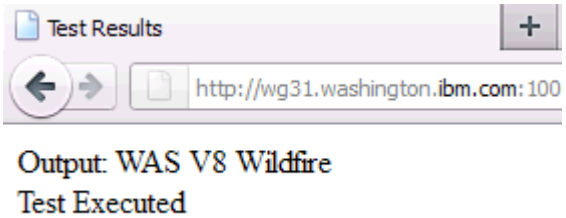
  `http://wg31.washington.ibm.com:10067/OLA_Sample2_Web/`

- ☐ From the web page, do the following:



Note: `OLACB01` was the sample COBOL program you compiled a few steps back. That sample has no WOLA APIs at all ... it's just a simple `COMMAREA` echo program. WOLA is "hidden" from `OLACB01` by the WOLA Link Server Task.

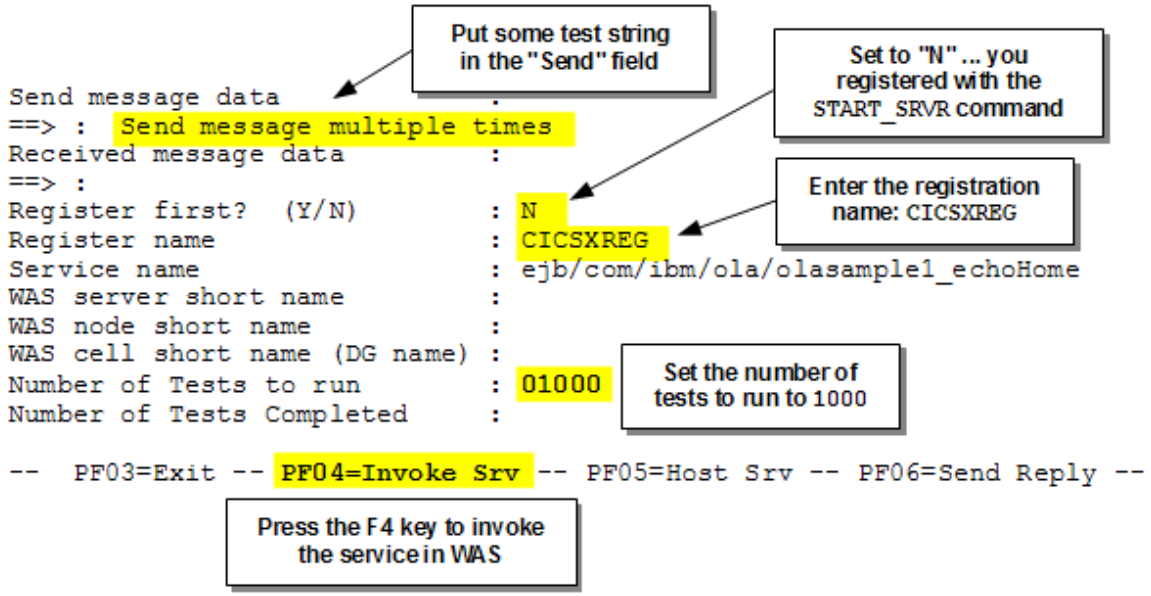
- ☐ If successful, the output back should show the data you sent:



Inbound: CICS OLAUTIL application to WAS

Note: Here you'll start to make use of the WOLA APIs in COBOL. When inbound from WAS to CICS the Link Server Task is not used. Use of APIs is required somewhere in the mix.

- ☐ Before we move on to compiling batch jobs that use the WOLA APIs, use the `OLAUTIL` 3270 application in CICS one more time to drive *multiple messages* into WAS:
  - ○ From the CICS 3270 session, get back to the `OLAUTIL` application
  - ○ Do the following:



  - ○ After a moment you should see something like this:

```
Invoke requested ... calls completed ok
 Start time: 10112011 110425
 Stop time : 10112011 110430 DONE Count: 00001000
```

Note: That drove 1,000 messages into WAS. What elapsed time do you see?
That program did not use the Link Server Task. If you were to look at the `OLAUTIL` sample you compiled earlier you'd see it makes extensive use of the WOLA APIs, one of which is `BBOA1INV`. Next you'll compile and run programs to use the APIs to drive work from a batch program into WAS.

- ☐ From the CICS 3270 screen issue the following command[49]:

`BBOC STOP_SRVR RGN=CICSXREG`

This will de-register from the WAS server. You should see the following message:

```
BBOA8000I Stop server completed successfully.
```

49 Just cleaning up the registration used by `OLAUTIL`. The batch COBOL will register using the `BBOA1REG` API.

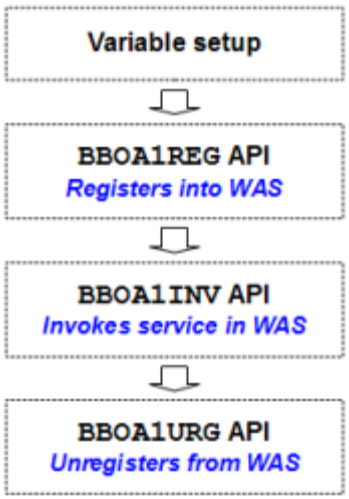
Inbound: batch applications to WAS

Note: Here you'll start to make use of the WOLA APIs in COBOL. When inbound from WAS to CICS the Link Server Task is not used. Use of APIs is required somewhere in the mix.

☐ Now browse the `USER1.WAS8.WOLA.CNTL` data set. You'll see the following[50]:

COMPILE	*Simple JCL to compile the COBOL into executable module*
EXER2A	*Simple BBOA1REG, BBOA1INV and then BBOA1URG*
EXER2B	*The same program as EXER2A except with a loop structure around BBOA1INV*
EXER2D	*Program that illustrates use of "advanced" APIs*
IGYWCL	*COBOL compile proc called by COMPILE JCL*
RUNPROG	*Simple JCL to run executable*

☐ *Browse* the `EXER2A` member and note the structure of the COBOL code:



☐ Now *edit* the `EXER2A` member and customize[51] the COBOL with the following:

```
000029       * Working Variables
000030        01 text-msg                                    0) VALUE SPACES.
000031
000032       * Procedures Section
000033        PROCEDURE DIVISION.
000034        MAINLINE SECTION.
000035            MOVE 'EXER2A'                              TO register-name.
000036            MOVE 'ccccc'                               TO daemongroup.
000037            MOVE 'nnnnnnn'                             TO node-name.
000038            MOVE 'sssssss'                             TO server-name.
000039            MOVE 'This is a test message'              TO text-msg.
000040            MOVE 'ejb/com/ibm/ola/olasample1_echoHome'
000041                 TO service-name.
```

Cell SHORT *uppercase*

Node SHORT *uppercase*

Server SHORT *uppercase*

Save the file.

50 These exercises are from the "WOLA Primer" found in the WP101490 document at `ibm.com/support/techdocs`

51 Cell = `Z9CELL`, Node = `Z9NODEA`, Server = `Z9SR01A`

- ☐ *Edit* the `COMPILE` member and make the following customization:

```
000008 //*****************************************************************
000009 //* Update the member name to point to your COBOL source          *
000010 //*****************************************************************
000011 //COBOL.SYSIN DD DSN=USER1.WAS8.WOLA.CNTL(aaaaaaaa),DISP=SHR
000012 //*
000013 //LKED.SYSLIB DD DSN=SYS1.LEMVS.SCEELKED,DISP=SHR
000014 //            DD DSN=USER1.WAS8.WOLA.LOADLIB,DISP=SHR
000015 //*****************************************************************
000016 //* Update the member name AND the module name to match the name  *
000017 //* of the module you are creating                                *
000018 //*****************************************************************
000019 //LKED.SYSLMOD DD DSN=USER1.WAS8.WOLA.LOADLIB(aaaaaaaa),DISP=SHR
000020 //LKED.SYSPRINT DD SYSOUT=*
000021 //LKED.SYSIN   DD  *
000022   NAME aaaaaaaa(R)
000023 /*
```

Change these three instances to EXER2A, which is the source you just edited and customized

Save the file.

- ☐ Submit the `COMPILE` member to compile the program.
- ☐ *Edit* the `RUNPROG` member and make the following customization:

```
000001 //RUNPROG   JOB (),'USER1',REGION=0M,MSGCLASS=H,NOTIFY=&SYSUID
000002 //*****************************************************************
000003 //* CHANGE PGM= TO THE NAME OF THE WOLA PROGRAM MODULE TO RUN      *
000004 //*****************************************************************
000005 //WOLA      EXEC PGM=aaaaaaaa
000006 //*
000007 //STEPLIB   DD DSN=USER1.WAS8.WO
000008 //SYSPRINT  DD SYSOUT=*
000009 //SYSUDUMP  DD SYSOUT=*
```

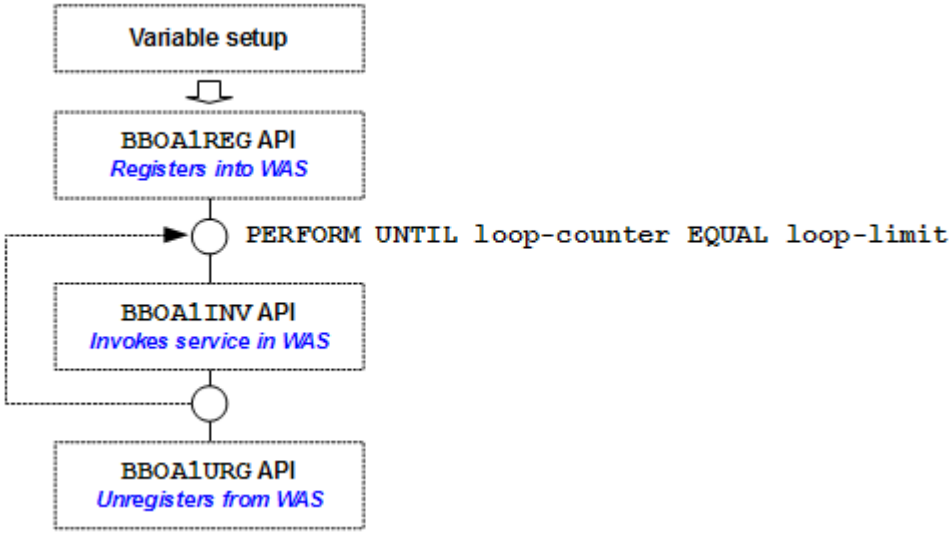
Change this instances to EXER2A, which is the name of the executable module

Save the file.

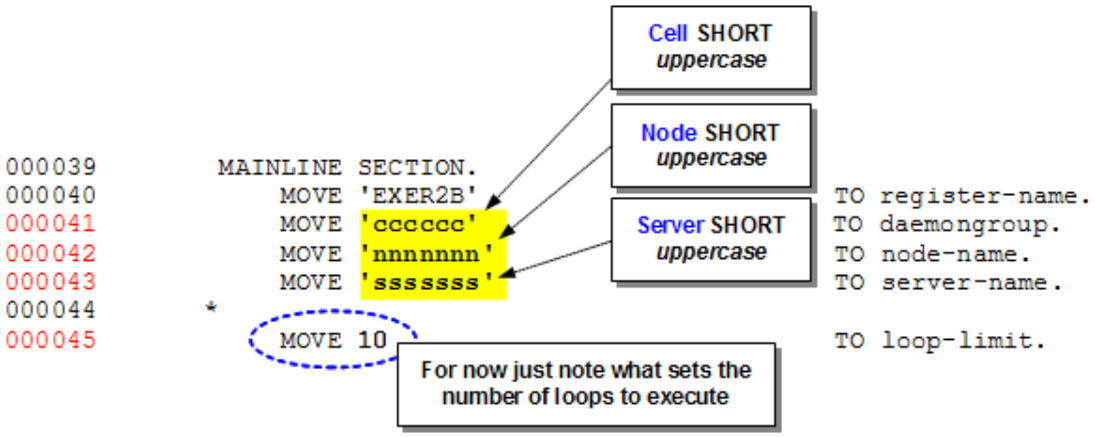
- ☐ Submit `RUNPROG` to run the program. Look for RC=0. You should see the following in the held output for the job:

```
Successfully registered into Z9CELL
Message sent: This is a test message
Message back: This is a test message
Successfully unregistered from Z9CELL
```

- ☐ Browse the `EXER2B` member and note the structure of the COBOL, which includes a loop structure around the `BBOA1INV` API (line 75 is the `PERFORM UNTIL` statement):



- ☐ Edit `EXER2B` and customized much like you did for the first job:



- ☐ Compile that program using the `COMPILE` job. Update all three references to the program name.
- ☐ Update the `RUNPROG` member and run the batch job. This will drive 10 message into WAS and the output should look like this:

```
Successfully registered into Z9CELL
Message sent: 0001 This is a test message
Message back: 0001 This is a test message
Message sent: 0002 This is a test message
Message back: 0002 This is a test message
Message sent: 0003 This is a test message
Message back: 0003 This is a test message
Message sent: 0004 This is a test message
Message back: 0004 This is a test message
Message sent: 0005 This is a test message
Message back: 0005 This is a test message
Message sent: 0006 This is a test message
Message back: 0006 This is a test message
Message sent: 0007 This is a test message
Message back: 0007 This is a test message
```

```
Message sent: 0008 This is a test message
Message back: 0008 This is a test message
Message sent: 0009 This is a test message
Message back: 0009 This is a test message
Message sent: 0010 This is a test message
Message back: 0010 This is a test message
Successfully unregistered from Z9CELL
```

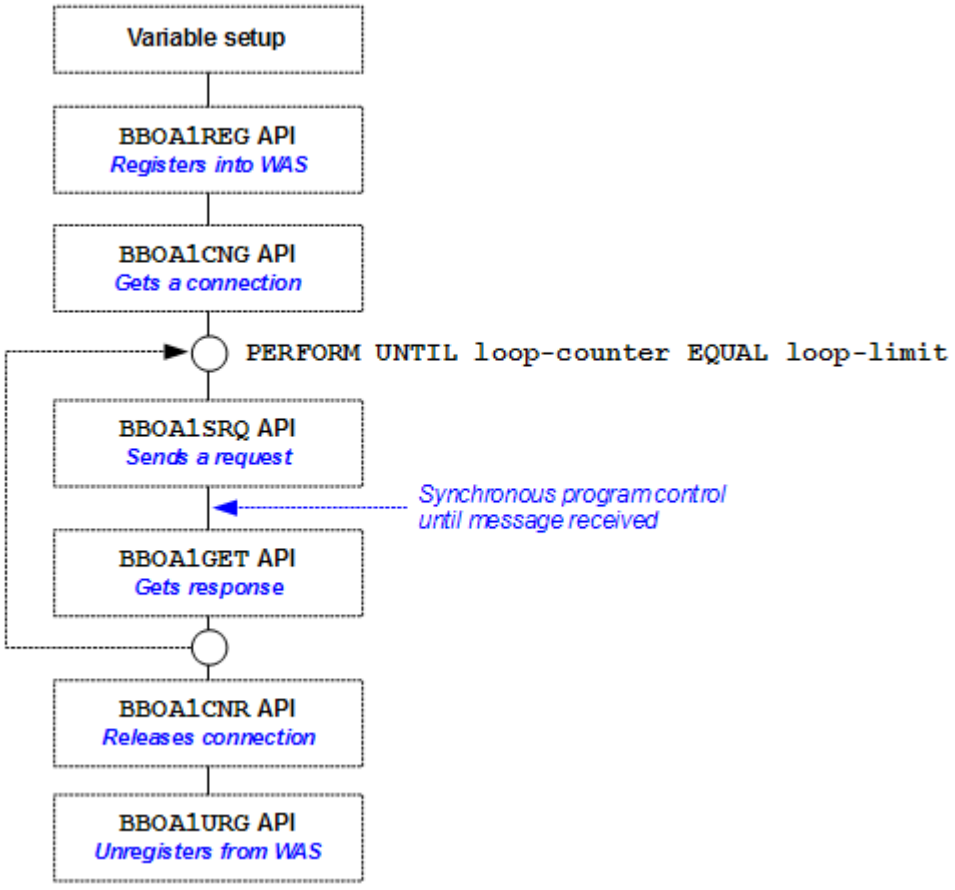
- ☐ Go back in the `EXER2B` source and change the loop count value from `10` to `5000`. Compile and submit. Look in the held output for the elapsed time and CPU time spent:

```
+-------------------------------------------------------------------------+
|Elapsed Time: 00:00:05.79  CPU Time(TCB): 00:00:00.77  (SRB): 00:00:00.03  |
|RCT CPU Time: None          I/O Int. Time: None         ICSF Count: None    |
|Service Units: CPU=   21,871  SRB=      941  I/O=      111  MSO=        0  |
|Transactions: ended:None   Active for: 00:00:05.78 Resident for: 00:00:05.78 |
|WLM: Workload: BAT_WKL  Service Class:BAT_MED   Resource Group: None       |
```

What's the elapsed time per message for the 5,000 messages sent and received?

Note: The CPU time does not include WAS. It is just the batch job itself. And the work being exercised in WAS is a trivial echo program.

- ☐ Now go browse the `EXER2D` member and note the structure, which looks like this:



Note: The combination of `CNG`, `SRQ`, `GET` and `CNR` is equivalent to the simpler `BBOA1INV`. The "advanced" APIs offer greater control. Here we're showing the re-use of the connection rather than returning it to the pool each invocation like `BBOA1INV` does. The advanced APIs also provide *asynchronous* program control, though this example is still using *synchronous*.

- ☐ Edit the `EXER2D` member and update the cell, node and server short names (lines 44-46) and change the loop count variable (line 48) to **`5000`**.
- ☐ Compile and run the program. Look at the elapsed and CPU time and compare to the `EXER2B` run which used `BBOA1INV`.

> **Note:** There's a degree more efficiency in `EXER2D` than `EXER2B` because we're re-using the connection. The numbers probably won't be dramatically different. And we're running "only" 5,000 invocations, which is relatively small for some batch processes. Small gains in efficiency accumulate when repeated tens or hundreds of thousands of times.

Outbound: WAS to CICS using Link Server Task

> **Note:** When going outbound from WAS to CICS you may use the Link Server Task to "hide" WOLA from the target program, or bypass the Link Server Task and "Host a Service" directly. In this lab you'll use the Link Server and drive the `OLACB01` program, which has no WOLA knowledge at all. The Link Server Task hides WOLA from the simple echo program.

- ☐ Open a 3270 session, clear the screen, then issue the command:

  `logon applid(CICSX)`

- ☐ Clear the CICS logo screen and issue the following command *all on one line*:

  `BBOC START_SRVR RGN=CICSXREG DGN=Z9CELL NDN=Z9NODEA SVN=Z9SR01A SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y`

  You should see:

  ```
  BBOA8000I Start server completed successfully.
  ```

- ☐ Check the WOLA registration into the server ... issue the following command from the MVS console:

  `F Z9SR01A,DISPLAY,ADAPTER,DAEMONRGES`

  If the registration is still in place you should see:

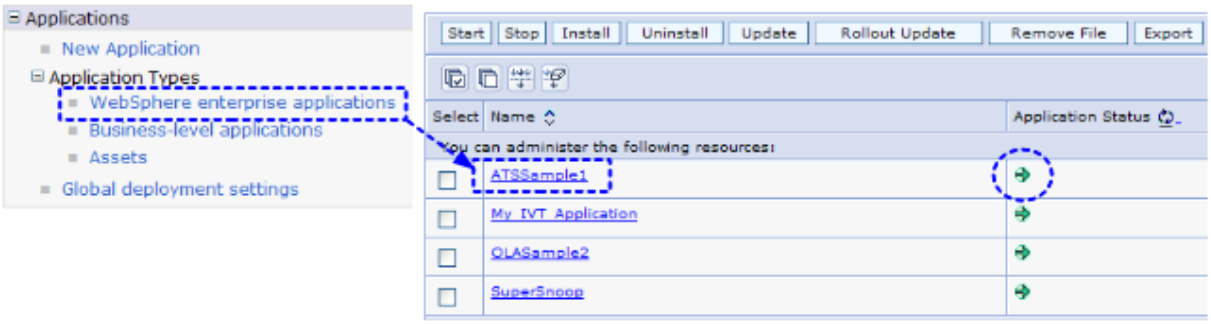
  ```
  BBOA0007I: SHOWING REGISTRATIONS FOR DAEMON GROUP:
  BBOA0003I: Name          Jobname  SWT TL Min  Max  Act  State
  BBOA0004I: CICSXREG      CICSX    000 00 0001 0010 0002 00
  BBOA0004I: $WASDEFAULT$ CICSX    000 00 0000 0001 0000 00
  BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,ADAPTER,DAEMONRGES
  ```

- ☐ Install into the first server the following application:

  `/wasetc/was8lab/applications/ATSSample1-new.ear`

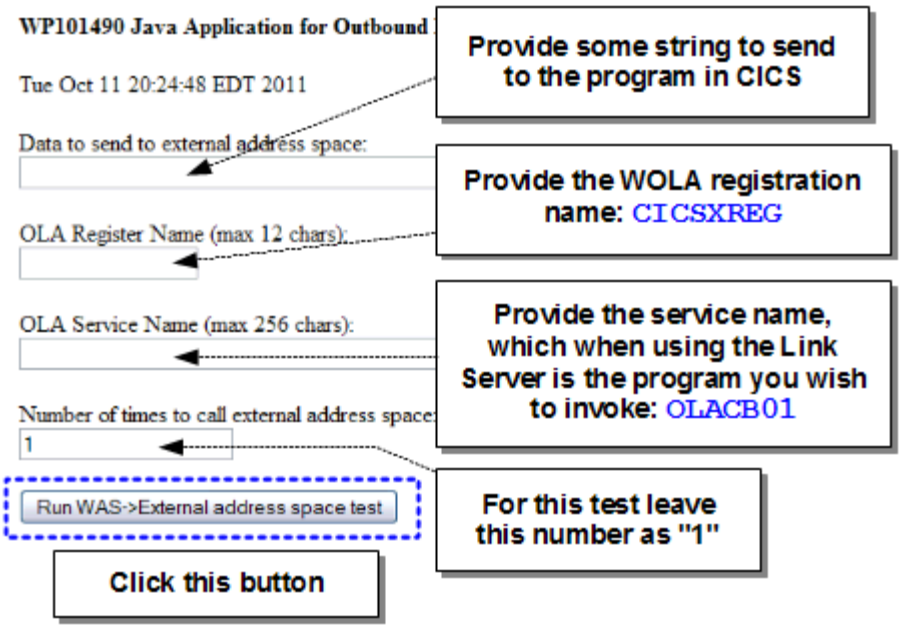
  Map the unresolved resource reference to the WOLA JNDI of `eis/ola`. Take all the other defaults.

- ☐ Check to see if the application is started:

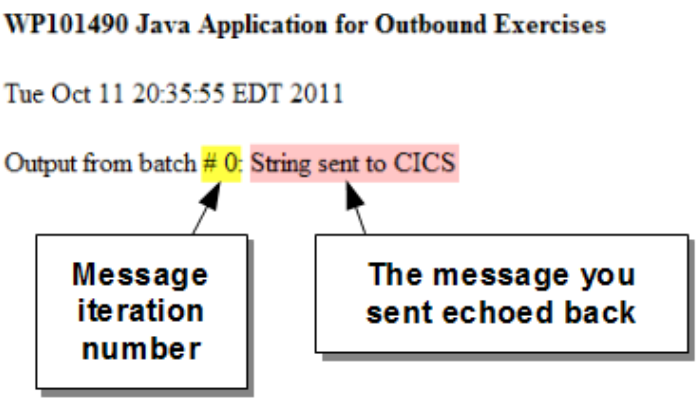


If not started, select the checkbox next to the application and click "Start."

- ☐ Point your browser at the following URL:
  `http://wg31.washington.ibm.com:10067/ATSSample1Web/`
- ☐ Fill in the form as follows:



- ☐ You should see something like this:



- ☐ Use the browser "back" button to return to the main application panel. Change the "Number of times to call external address space:" value to **`100`**. Click the button again. You should se the response panel again, but this time with 100 echoed messages.
- ☐ Do it again but this time with **`1000`** for the value of the messages to send.

Outbound: WAS to CICS with "alternative JNDI failover"

Note: Next we'll illustrate how to use the V8 function that allows for failover to a backup CICS region. In this lab that backup CICS region will be `CICSY`. That means we first must to do a *little* bit of setup work for `CICSY` so WOLA is enabled there.

- ☐ Close the registration from `CICSX` into WAS. From the CICS 3270 session[52] issue the following command:
  `BBOC STOP_SRVR RGN=CICSXREG`

52 Open a 3270 session, clear the screen, issue `LOGON APPLID(CICSX)`, then clear the CICS screen and issue command.

- ☐ Clear the CICS screen and issue the command **`CESF LOGOFF`**.
- ☐ Stop CICSX -- **`/F CICSX,CEMT PERFORM SHUTDOWN`**
- ☐ *Restart* CICSX[53] -- **`/S CICSX`**
- ☐ *Edit* the **`CICSY.CICS42.SYSIN(CICSYSIP)`** member and add the following line:

```
000027 SYSIDNT=CICY
000028 TCT=NO
000029 PLTPI=OL
000030 .END
```

- ☐ Update the `SYS1.PROCLIB(CICSY)` member and add the following line to `DFHPRL`:

```
004300 //DFHRPL   DD DSN=&CICSDS..SDFHLOAD,DISP=SHR
004400 //          DD DSN=SYS1.LEMVS.SCEECICS,DISP=SHR
004500 //          DD DSN=SYS1.LEMVS.SCEERUN2,DISP=SHR
004600 //          DD DSN=SYS1.LEMVS.SCEERUN,DISP=SHR
004700 //          DD DSN=SYSS.CICS.LOADLIB,DISP=SHR
004800 //          DD DSN=SYS1.XML.SIXMLOD1,DISP=SHR
004900 //          DD DSN=CTS420.CICS.SDFHWSLD,DISP=SHR
005000 //          DD DSN=USER1.WAS8.WOLA.LOADLIB,DISP=SHR
```

- ☐ Once again, issue the following two commands from TSO Option 6 to grant the CICS region ID access to the WAS runtime, this time for the `CICSY` ID:

```
PERMIT CB.BIND.Z9* CLASS(CBIND) ID(CICSY) ACCESS(READ)
SETROPTS RACLIST(CBIND) REFRESH
```

- ☐ Issue the command **`/S CICSY`** and then look for the following in the region's output for validation that the PLTPI initialized the Task Related User Exit:

```
+BBOA9920I WAS z/OS OLA CICS PLT init start.
+BBOA9921I WAS z/OS OLA CICS TRUE enabled.
+BBOA9925I WAS z/OS OLA CICS PLT init ending.
+BBOA9940I WAS z/OS OLA CICS PLT init 2 start.
```

- ☐ In the Admin Console go back to your WOLA resource adapter under *Resource* ➔ *Resource Adapters*. Click on the link that represents that WOLA resource adapter.
- ☐ Click on "J2C connection factories" and create a second connection factory so that you have two CFs that look like this:

Select	Name	JNDI name	Scope	Provider
You can administer the following resources:				
☐	ola	eis/ola	Node=z9nodea	OptimizedLocalAdapter
☐	ola2	eis/ola2	Node=z9nodea	OptimizedLocalAdapter

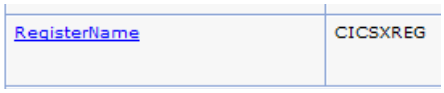
- ☐ Click on your *new* (alternate) connection factory link, then "Custom Properties." Update the "RegisterName" property[54] so it has the WOLA registration name from CICSY:

RegisterName	CICSYREG

53 There's a bunch of messages in the CICSX held output we need to clear so the failover test (next) is easier to see.

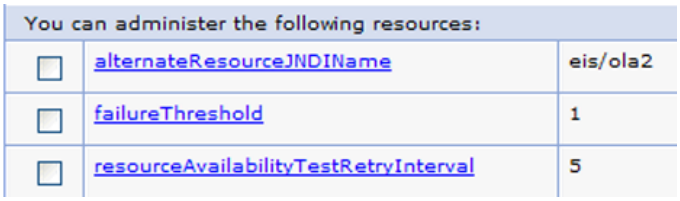
54 This relieves the application from having to use `setConnection()`, and allows the high availability failover function to work. Both connection factories will have `RegisterName` set.

- Click on your *original* (primary) connection factory ("ola" with JNDI name of "eis/ola"), and then under "Additional Properties" click on "Custom Properties." Then update the "registerName" property so it includes the WOLA registration name from `CICSX`:



RegisterName	CICSXREG

- Go back one screen, then click on "Connection Pool Properties," then "Connection Pool Custom Properties" for your *primary* eis/ola CF. Create three custom properties so the properties under your primary connection factory looks like this:



You can administer the following resources:	
alternateResourceJNDIName	eis/ola2
failureThreshold	1
resourceAvailabilityTestRetryInterval	5

Reminder: `alternateResourceJNDIName` provides the JNDI name of the alternate connection factory

`failureThreshold` indicates how many `getConnection()` failures must be seen before invoking the failover processing.

`resourceAvailabilityTestRetryInterval` indicates the polling interval WAS uses to determine when the primary resource is back and available.

- Save and synchronize all changes.
- Stop and restart the `z9sr01a` server.
- In a 3270 session, clear the screen and issue command `logon applid(CICSX)`. Clear the screen again, then issue the following command as one long command:

```
BBOC START_SRVR RGN=CICSXREG DGN=Z9CELL NDN=Z9NODEA SVN=Z9SR01A
                                    SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y
```

- Clear the screen, then issue the command `CESF LOGOFF`.
- From the 3270 menu bar, select *Communication* ➔ *Connect*. Clear the screen and issue:

```
logon applid(CICSY)
```

clear the screen again and then issue the command:

```
BBOC START_SRVR RGN=CICSYREG DGN=Z9CELL NDN=Z9NODEA SVN=Z9SR01A
                                    SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y
```

- Back in your TSO session issue the MVS command:

```
F Z9SR01A,DISPLAY,ADAPTER,DAEMONRGES
```

You should see *two* registrations into the server:

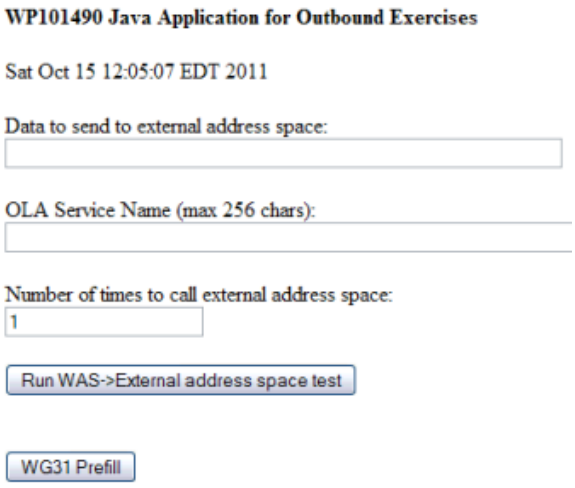
```
BBOA0007I: SHOWING REGISTRATIONS FOR DAEMON GROUP:
BBOA0003I: Name          Jobname  SWT TL Min  Max  Act  State
BBOA0004I: CICSXREG      CICSX    000 00 0001 0010 0001 00
BBOA0004I: $WASDEFAULT$ CICSX    000 00 0000 0001 0000 00
BBOA0004I: CICSYREG      CICSY    000 00 0001 0010 0004 00
BBOA0004I: $WASDEFAULT$ CICSY    000 00 0000 0001 0000 00
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,ADAPTER,DAEMONRGES
```

- ☐ You're almost ready to test the failover capability. One final thing remains: installing an application that does not use `setConnection()` and rather relies on the `RegisterName` property being set on the connection factory.

  So, install the application `/wasetc/was8lab/applications/ATSSample2.ear` into the `z9sr01a` server, resolving the resource reference to `eis/ola` and taking the defaults for everything else.
- ☐ Start the ATSSample2 application.
- ☐ Point your browser at the following URL:

  `http://wg31.washington.ibm.com:10067/ATSSample2Web/`

  You should see:

- ☐ Put some data string in the first field, type in `OLACB01` in the service name field and then click the "RUN WAS->External address space test" button.
- ☐ Look in the `CICSX` region held output and search on the string `WBSR8`. You should see a message indicating the request flowed over the primary connection factory and into CICSX[55].
- ☐ From the MVS console, issue the command **`/c CICSX`**. This will kill the CICSX region and make any `getConnection()` across the primary connection factory fail.
- ☐ On the browser use the "back button" and click the "Run" button again. You should see:

  An error has occurred!
  Please try your request again.

  This indicates a `getConnection()` failure across the primary connection factory[56].
- ☐ Since `failureThreshold` is set to `1`, that one failure is all that's needed to tell WAS to failover. Use the browser back button, then invoke the service again. This time you should see success at the browser.
- ☐ Look in the `CICSY` region held output and search on the string `WBSR8`. You should see the message, indicating the request flowed over the *alternate* connection factory and into the `CICSY` region.
- ☐ Use the browser to invoke the service multiple times into CICSY.

55 The string will look garbled ... that's because the Java program sent it in ASCII and no code-page conversion was done.
56 Which makes sense ... you just canceled that region. There's nobody there.

- Restart the `CICSX` region. Then open a 3270 screen, clear the screen, issue command `logon applid(CICSX)`. Clear the screen again, then issue the following command as one long command:

```
BBOC START_SRVR RGN=CICSXREG DGN=Z9CELL NDN=Z9NODEA SVN=Z9SR01A
                                        SVC=* MNC=1 MXC=10 TXN=N SEC=N REU=Y
```

- If you invoke the service again before the `resourceAvailabilityRetryInterval=5` timer pops, you'll see the request flow over to `CICSY`. But if after that five seconds it'll revert back to the primary and go into `CICSX`.

End of Unit 6 Lab

Answers, Hints and Tips

Brief tutorial on 3270 and MVS usage

ISPF panel shortcuts

=3.4	Data Set *List* Utility -- allows you to find and display data sets using the full name of the data set or high level qualifiers as wildcards.
=SDSF.DA	The Display *Active* panel. Shows jobs and tasks that are currently active. See "=SDSF, ST, DA and PREFIX" on page 83 for instructions on limiting what is displayed.
=SDSF.ST	The Display *Status* panel. Shows jobs and tasks both active and completed. See "=SDSF, ST, DA and PREFIX" on page 83 for instructions on limiting what is displayed.
=SDSF.LOG	The MVS system console. Shows output and activity for the entire z/OS system. See "Entering long commands" on page 82 for instructions on how to enter very long commands into MVS.
=SDSF.ENC	The WLM enclave display panel.

Entering short commands

On the `=SDSF.DA`, `=SDSF.ST` or `=SDSF.LOG` panels you may enter *short* commands at the "Command Input" field: `COMMAND INPUT ===>` A leading `/` is required.

Entering long commands

The "Command Input" field may not be long enough for some of the commands required by this workshop.

A single slash (`/`) entered in the command input opens up the "command extension":

```
Type or complete typing a system command, then press Enter.

===> _
```

That field wraps and allows commands up to 126 characters. A leading slash is *not* needed.

The command extension remembers commands entered:

```
Place the cursor on a command and press Enter to retrieve it.

=>   S Z9ACRA,JOBNAME=Z9AGNTA,ENV=Z9CELL.Z9NODEA.Z9AGNTA
=>   S Z9DCR,JOBNAME=Z9DMGR,ENV=Z9CELL.Z9DMNODE.Z9DMGR
=>
```

You may place your cursor on a previously entered command to retrieve it to the command line. There you may enter it again or modify it and enter it again.

Scrolling up and down in browse and edit mode

Scroll amount	Look in the upper right of the screen. You should see one `SCROLL ===> CSR` or `SCROLL ===> PAGE` `CSR` = scroll to the location indicated by the placement of the cursor `PAGE` = scroll one full page amount You may change the scroll value by overtype the field.
F7	Scroll up one scroll unit (CSR or PAGE)
F8	Scroll down one scroll unit (CSR or PAGE)
M + F7	`COMMAND INPUT ===> M` then F7 scrolls to the top
M + F8	`COMMAND INPUT ===> M` then F8 scrolls to the bottom

=SDSF, ST, DA and PREFIX

The `PREFIX` function allows you to limit what is displayed with `=SDSF.ST` and `=SDSF.DA`

Syntax[57]: `PREFIX <prefix string and asterisk>` then host enter (right Ctrl key)

Example: `COMMAND INPUT ===> PREFIX Z9*` limits display to your `Z9CELL` regions.

Result:

```
COMMAND INPUT ===>
PREFIX=Z9*  DEST=(ALL)  OWNER=**  SORT=JOBNA
NP   JOBNAME  StepName JobID     CPU% Real
     Z9AGNTA  Z9AGNTA  STC00469  0.08  49T
     Z9DEMN   Z9DEMN   STC00467  0.00  385
     Z9DMGR   Z9DMGR   STC00465  0.10  81T
     Z9DMGRS  Z9DMGRS  STC00468  0.07 198T
     Z9SR01A  Z9SR01A  STC00683  0.08  61T
     Z9SR01AS Z9SR01AS STC00688  0.10  91T
```

An **S** next to `JOBNAME` will open up the held output for that job.

Searching for strings in job output

The command `FIND` may be used to search for output from the cursor location down.

If the string has no spaces, it may be entered like this:

```
SDSF OUTPUT DISPLAY Z9SR01AS STC00688  DSID
COMMAND INPUT ===> FIND string
```

But if there's a blank in the string being sought then enclose in single quotes:

```
SDSF OUTPUT DISPLAY Z9SR01AS STC00688  DSID
COMMAND INPUT ===> FIND 'string string'
```

If multiple instances of the string are present, **F5** will find next.

If you wish to search from the cursor location up, then use the `PREV` keyword:

```
SDSF OUTPUT DISPLAY Z9SR01AS STC00688  DSID
COMMAND INPUT ===> FIND string PREV
```

57 The shortcut `PRE` works as well. The asterisk may not be required depending on your system. Use of the asterisk is a better practice to insure proper results until you understand the capabilities of your specific system.

Narrowing on a specific DD in held output

Placing a question mark (`?`) next to a job in `=SDSF.DA` or `=SDSF.ST` displays the individual held DDs within the job. Then you may use `S` to see only that specific output:

System inventory answers

☐ Go under *System Administration* ➔ *Nodes*. Take inventory of what you see:

Node Name	*What type of node?*	*Synchonization Status?*
z9dmnode	☒ **Deployment Manager** ☐ Managed Node	☒ **Green circle synchonized** ☐ Red symbol not synchronized
z9nodea	☐ Deployment Manager ☒ **Managed Node**	☒ **Green circle synchonized** ☐ Red symbol not synchronized

☐ Go under *System Servers* ➔ *Server Types* ➔ *WebSphere Application Servers.* Take inventory of what you see:

Server Name	*Which node?*	*Synchonization Status?*
z9sr01a	☐ Deployment Manager node ☒ **Managed Node**	☐ Green arrow started ☒ **Red X not started**

☐ Click on the link representing the server in the list and then *Server Infrastructure* ➔ *Java and Process Management* ➔ *Server instance.* Take inventory:

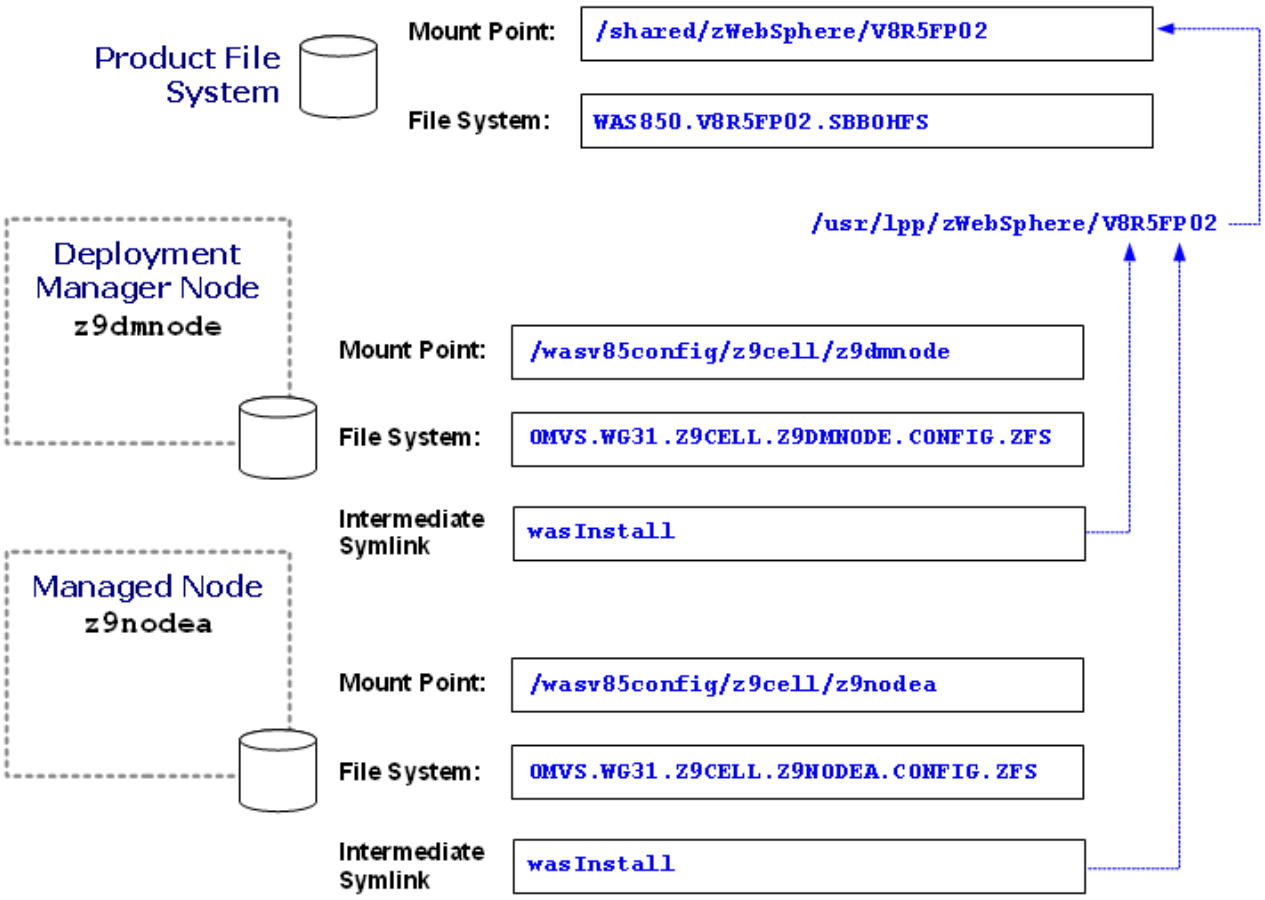
Multiple Instances Enabled	☐ Yes ☒ **No**
Minimum Number of Instances	☒ **1** ☐ 2 ☐ other: _____
Maximum Number of Instances	☒ **1** ☐ 2 ☐ other: _____

☐ Go under *Resources* ➔ *JDBC* ➔ *JDBC Providers.* Is there anything there that suggests the IBM DB2 JDBC driver is installed? **You should see one JDBC resource provider, but it won't mention DB2 in its name.**

☐ Go under *Resources* ➔ *Resource Adapters* ➔ *Resource adapters.* Is there anything there that suggests either the CICS JCA resource adapter or the WOLA resource adapter is installed? **You should see an empty list ... no resource adapters installed yet.**

Configuration file system lab answers

Your picture should look like this:

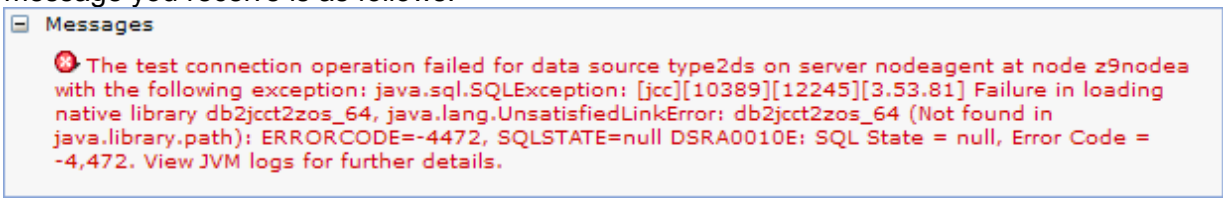


WSADMIN SuperSnoop Installation Mini-Quiz Answers

#	*Question*	*Your Answer*
Q1	What WSADMIN command object *and method* is used to **uninstall** SuperSnoop if it is found?	`AdminApp.uninstall(application)` *Where "application" is a variable that carries the application name to be uninstalled.*
Q2	After the application has been uninstalled, what command object *and method* is used to save the changes?	`AdminConfig.save()`
Q3	How many lines in the script are used to construct the `AdminApp.install()` option list?	There are four "appopts" lines. The first establishes the variable appopts and seeds the first string into it, the rest concatenate the previous with new input.
Q4	How many option parameters are in the application installation options list?	Two: `-appname` and `-MapModulesToServers`. The other values in that string are *values* that apply to the two option parameters.
Q5	What's the purpose of the `import sys` line at the top and the `splitlines()` used on `AdminApp.list()`?	`Import sys` brings in a set of useful utility class libraries. `splitlines()` serves to break the result of `AdminApp.list()` into a list that can be parsed as discrete units rather than one long string.

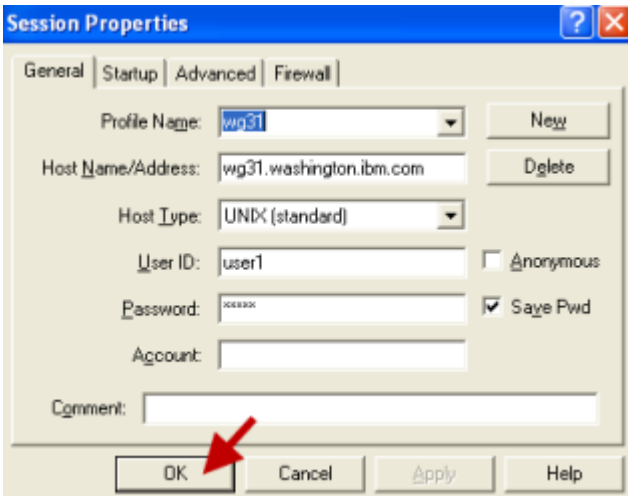
Example of Test Connection Failure when Scope=Node

If your JDBC provider and data source is scoped at the node level, the Test Connection attempts to run from the Node Agent. But there's no servant region there so the attempt fails. The message you receive is as follows:

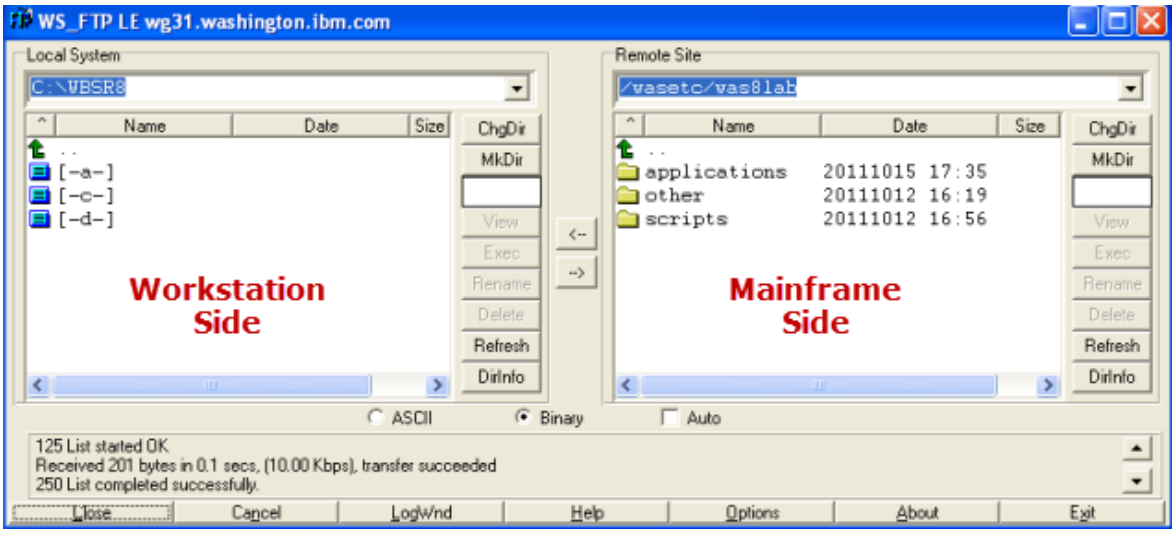


Using the WS-FTP client

- ☐ On the desktop, locate the WS-FTP icon and click on it.
- ☐ It will bring up a "Session Properties" panel that is all filled out for this workshop. You simply click "OK" to connect to your team's host system:



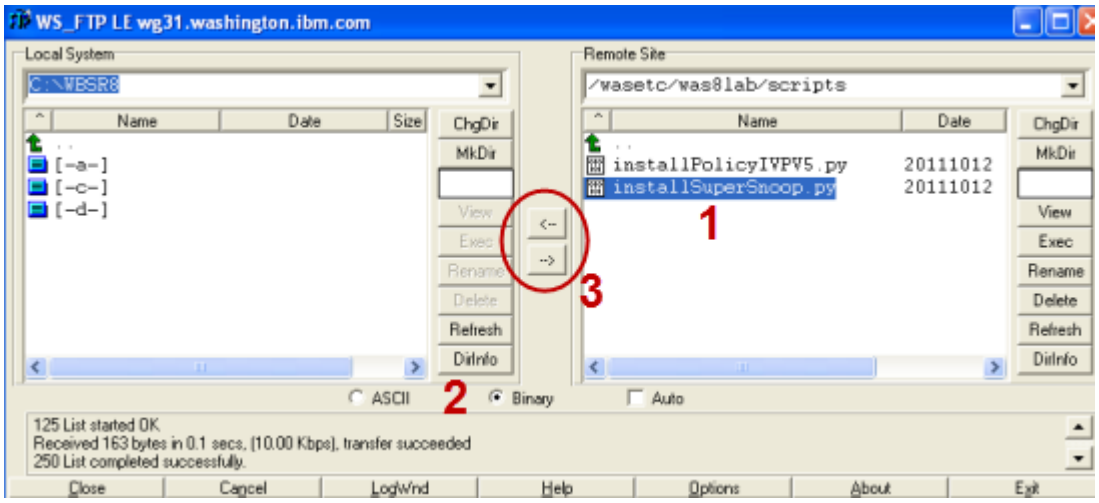
- ☐ You should then see:



The WS-FTP client has been preconfigured so it starts up with `C:\WBSR8` as its workstation home folder and `/wasetc/was8lab` as its host-side directory.

You can navigate up or down the folder or directory structure using the folder and "up-arrow" symbols on either side. Or you may type the path in the fields at the top and press enter.

- To download a file, do the following:



1 - Select the file

2 - Select ASCII or Binary for the download conversion to take effect

3 - Click the left arrow to download

- To upload a file do the reverse of download -- select PC file, then select ASCII or Binary, navigate to the desired target on the host side and then click the right arrow to upload.
- To view and update a workstation-side file, select the file on the left side of the WS-FTP panel and click "View." This will open Notepad and display the file. Make whatever changes you desire and save as usual.
- To view a host file select the file on the right side of the panel then select whether the file is to be converted before displaying -- ASCII means the file will go through EBCDIC-to-ASCII conversion before Notepad displays the file; Binary means the file is transferred to the PC and displayed without conversion.

End of Document