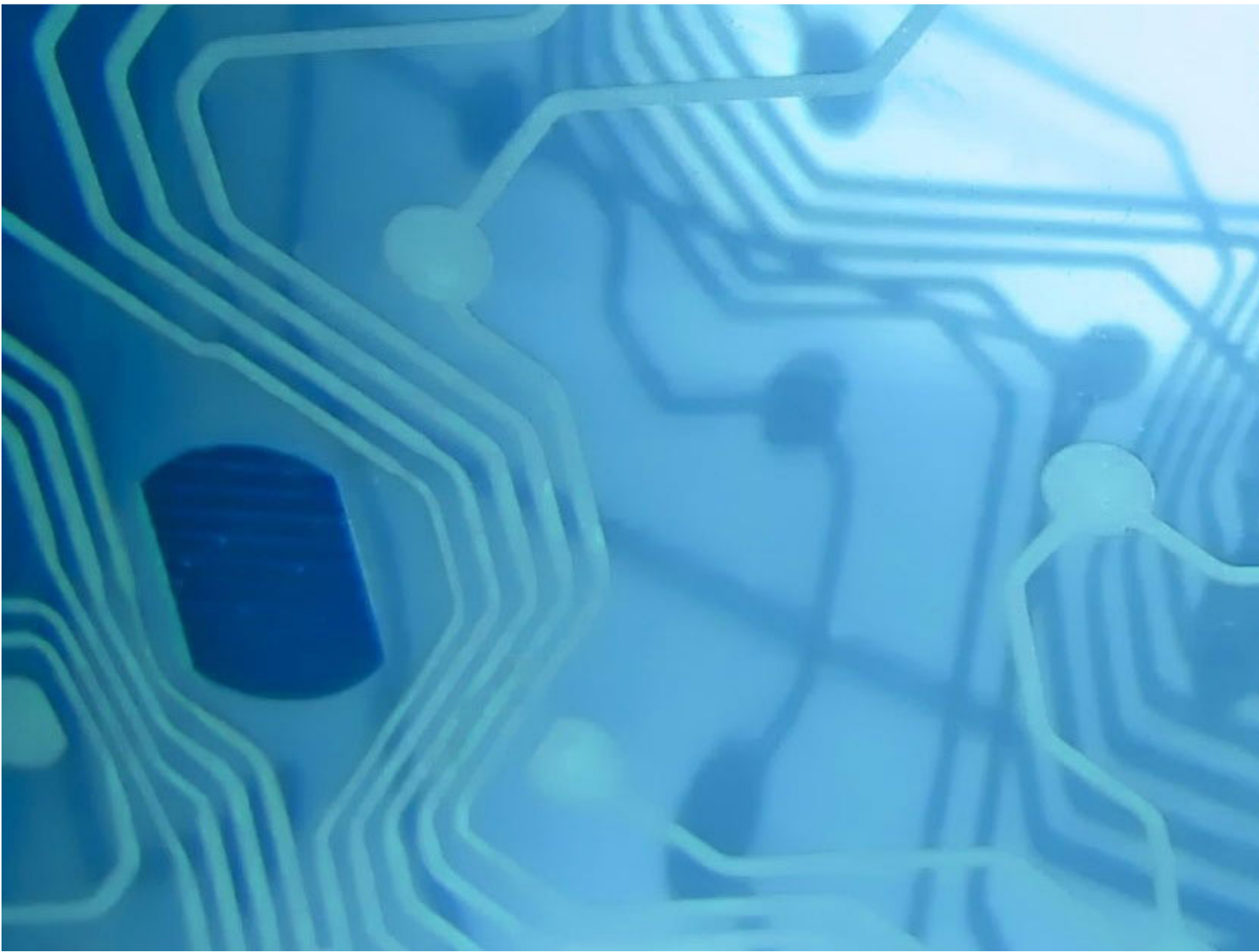
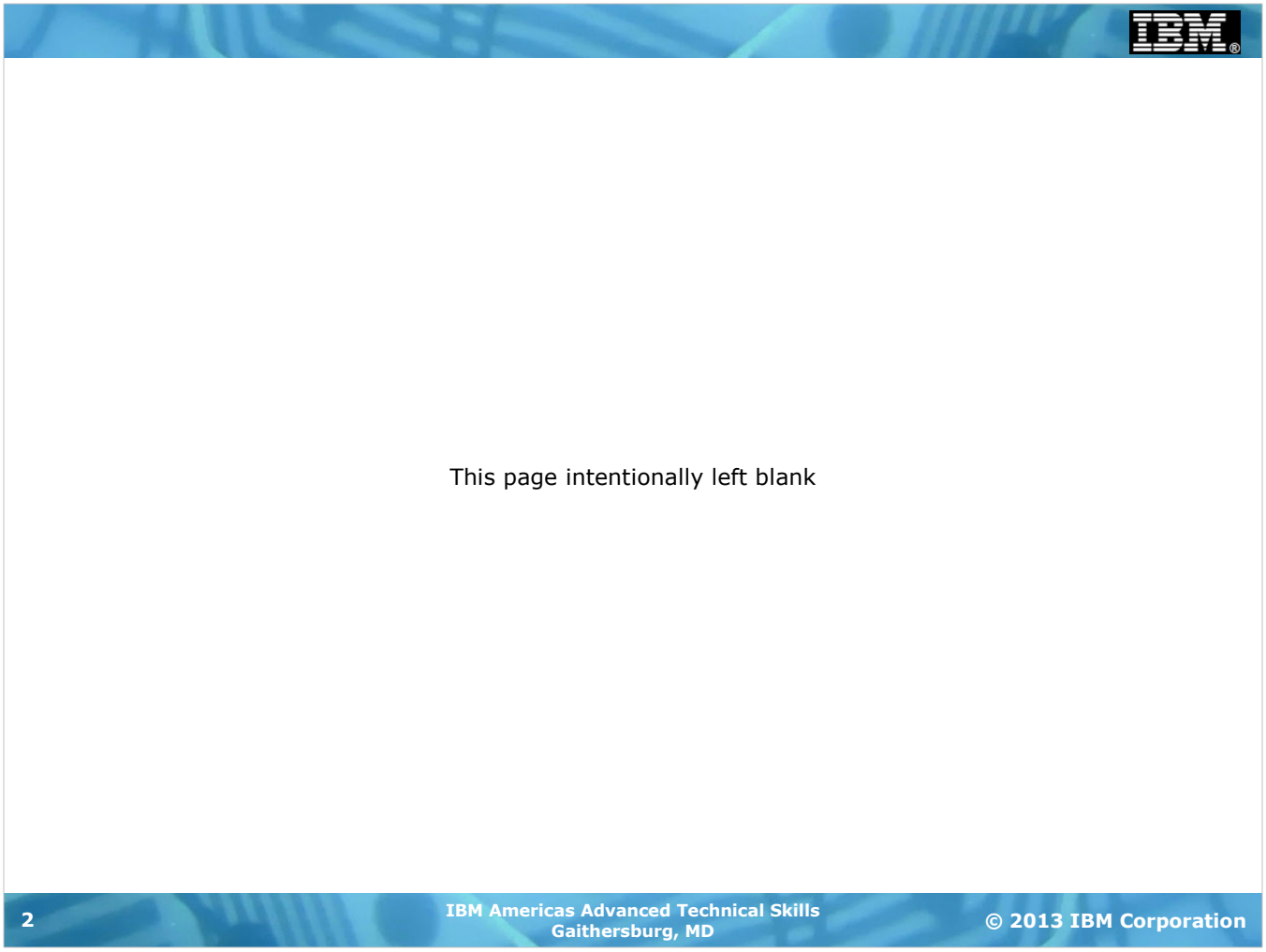
WebSphere Application Server V8.5 for z/OS

WBSR85

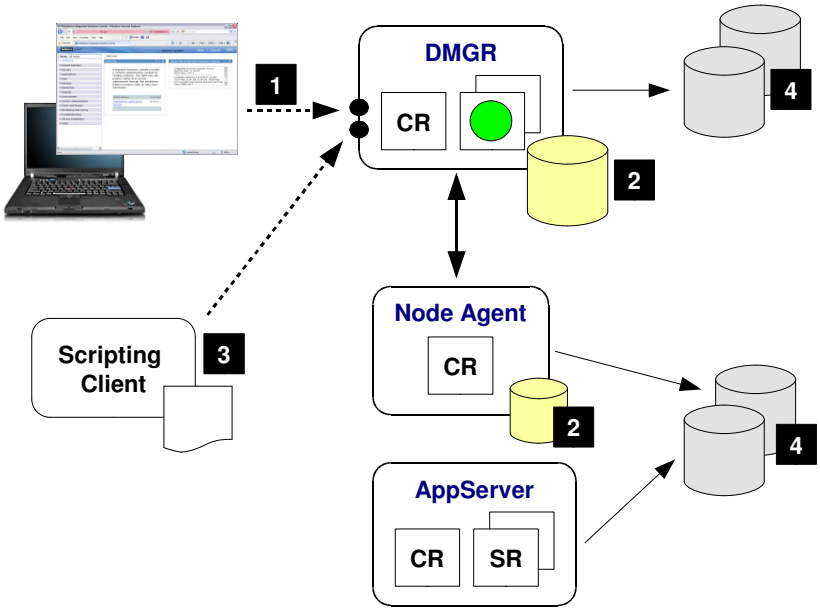
Unit 2 – Administrative Model




This page intentionally left blank
2
IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

High-Level Conceptual Picture

This provides the framework of our focus areas this unit:



1. **Administrative Console**
2. **Configuration File Systems for each Node**
3. **WSADMIN scripting interface**
4. **Logging and tracing and HPEL**

These are the topics we'll cover in this unit

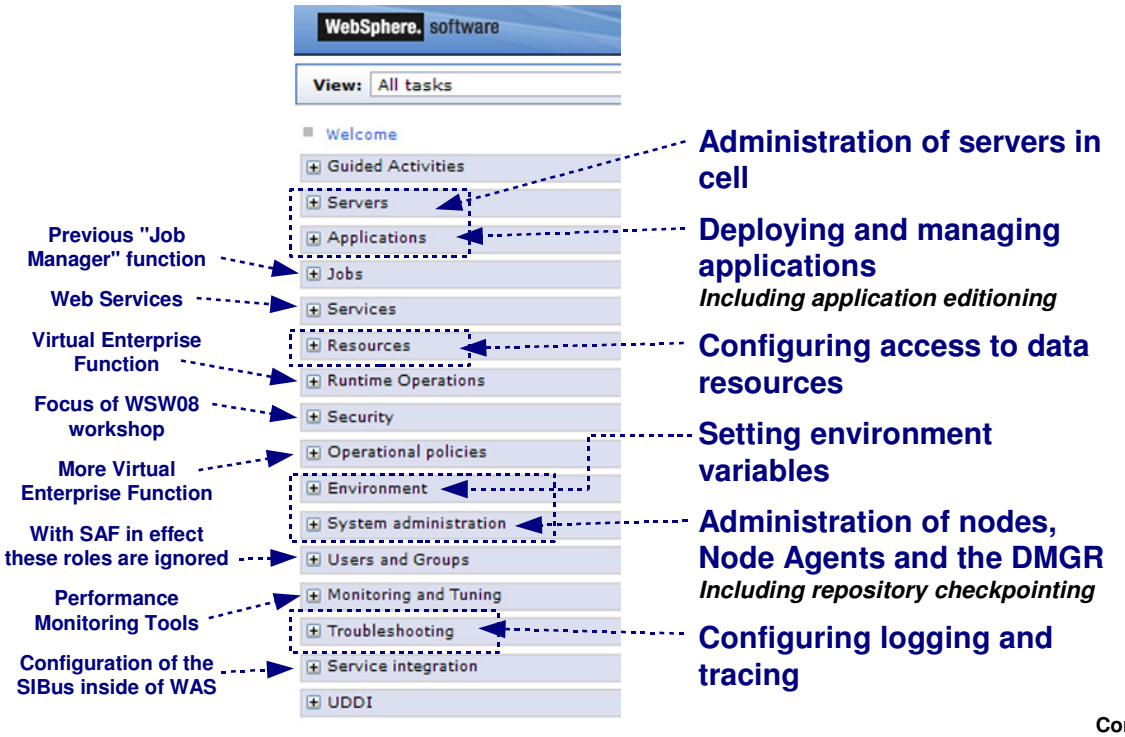
The administrative model focus we'll focus on in this unit spans four things:

1. **Administrative Console** -- the browser-based interface to the administrative application that runs in the Deployment Manager. Our objective will be to provide a quick review and offer insights into where z/OS-specific things begin to surface.
2. **Configuration file systems** -- each node has a configuration file system, and there's a relationship between the configuration file system and the product file system. We'll explore the structure of the configuration file system and review the concept of "intermediate symbolic links," which provide a degree of isolation between nodes for the purposes of maintenance.
3. **WSADMIN scripting** -- WAS provides a programmatic interface to the administrative function. It is called WSADMIN. The scripting language it supports is Jython (JACL as well, but our focus will be Jython).
4. **HPEL logging** -- we mentioned HPEL in the first unit. We'll go a little deeper here and explore what it looks like, how you configure it, and how you can use it.

Admin Console

Left-Side Navigator Bar for V8.5 Admin Console

At this level the Admin Console is common across all platforms. The areas of focus for us in this workshop are indicated below:



Commonality …

The Administrative Console, or "Admin Console" for short, is a browser interface to the administrative function. It is in many ways common across all platforms, though z/OS specifics do surface at certain points.

The browser interface is frames-based, with an expandable navigation tree down the left side and a main body frame to the right. A bitmap of the V8.5 Admin Console navigation tree is shown above with what we'll cover outlined with the dashed boxes and described to the right, and those things we won't cover briefly described to the left.

Degree of Commonality Across Platforms

Is actually fairly high ...

Servers and Clustering
Very common until you get to things like server short names and the Multi-JVM model

Applications
Identical across platforms

Web Services
Identical across platforms

Security
Fairly common with the exception of definitions to allow use of SAF

Environment
Interface is identical. Some of the variables you enter may be z/OS-specific.

Resources
Very common until you get to the definition of local adapters and native libraries

System Administration
Very common with the exception of the "z/OS location server" under "Node Groups"

SIBus
Identical across platforms

Platform specifics ...

The Admin Console has a high degree of commonality across platforms. The chart above attempts to give a sense for the degrees of commonality by functional area of the Admin Console.

Where z/OS-specific things surface it is because there's some element of WAS configuration that requires a z/OS-specific piece of information. For example, with servers and clustering WAS z/OS surfaces "short" names. (Short names are a way to accomodate the decades-old restriction z/OS has with 8-character names. Short names provide a way to map WAS elements down onto z/OS 8-character names.) The multi-JVM model of WAS z/OS also comes to light when you drill down into the server JVM structure on WAS z/OS.

But applications and Web Services are entirely common across platforms.

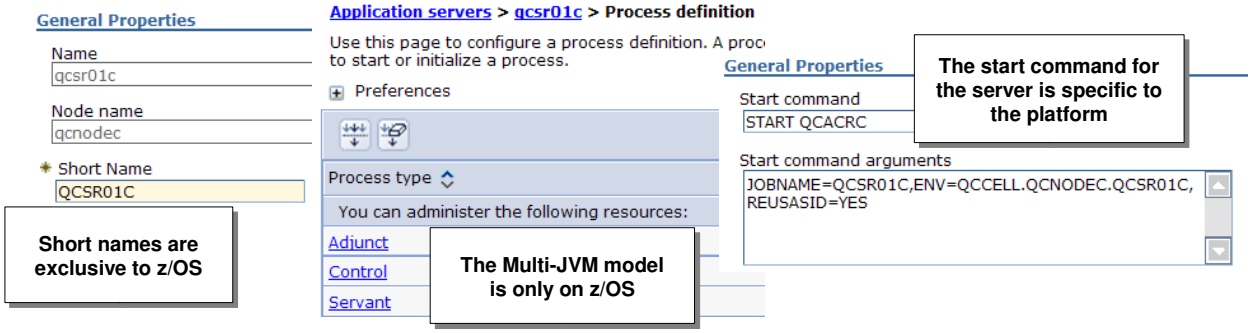
Security brings some z/OS things to the surface.

The point here is that WAS administrators familiar with the Admin Console on other platforms will find considerable comfort when they see the Admin Console on z/OS. It looks and operates in very much the same way.

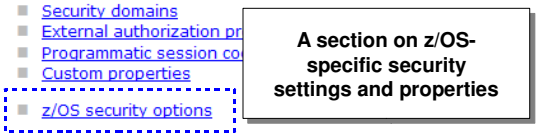
Examples of Platform Specifics Surfacing

A brief sampling of where some z/OS platform specifics surface in the Administration Console:

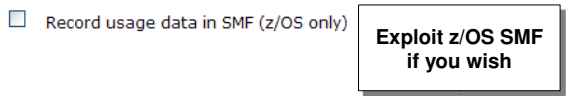
Under a given application server:



Under Global Security:



Under the integrated Java Batch configuration:



Point here is that while the Admin Console has a great deal of commonality, you can find differences the closer to the platform you get

Updating XML ...

This chart shows some examples of where z/OS specific things come to light. Under the application server the "General Properties" panel will have the name of the server (which is common across platforms, but on z/OS we call it the "long" name to differentiate it from the short name), as well as the short name we discussed on the previous panel. The short name is limited in length as an accomodation to length limitations in z/OS. The short name also maps to key z/OS constructs such as RACF profile names.

The multi-JVM model shows up when you drill down to the "Process definition" of a server. On distributed you would see one JVM, but on z/OS you see three -- Adjunct, Control and Servant. (The Adjunct server is used when the SIBus is configured for messaging. The controller and servant we'll cover in greater detail in the next unit.)

WAS maintains the command string used to start the server, and on z/OS it does this as well. But the start string for z/OS is considerably different than for distributed. Here we see a very z/OS-specific value with the JCL start procedure name, the MVS JOBNAME specification and an environment string that's used to specify the server to be started.

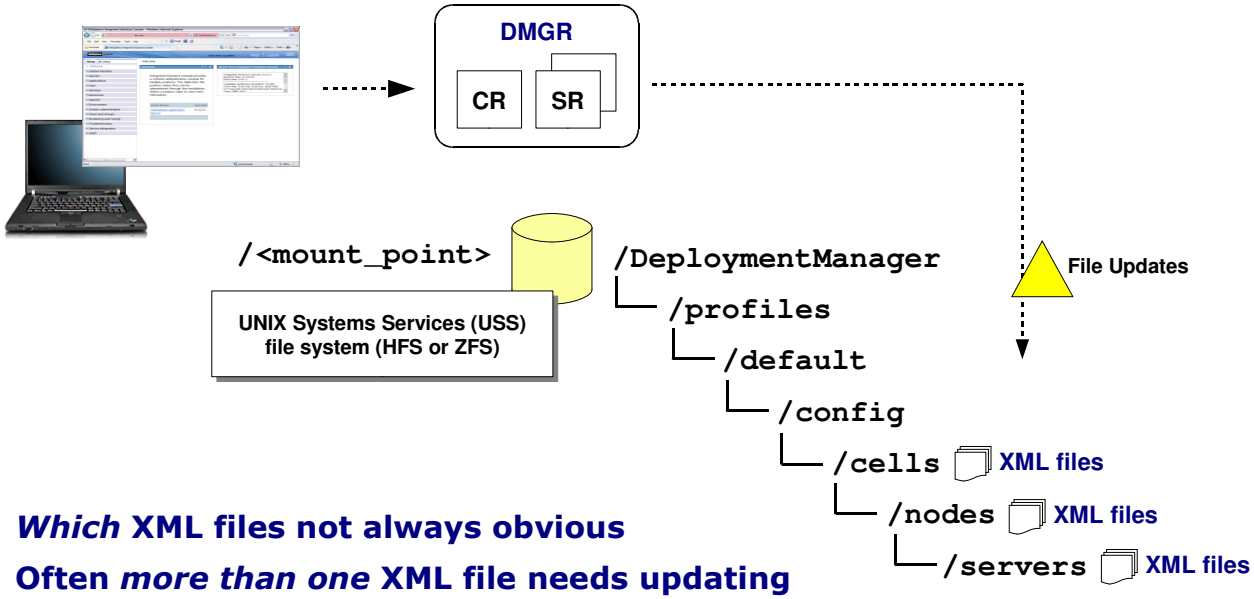
Under security we see there's a link for z/OS security options. This is used to map WAS security constructs to z/OS security in SAF.

Finally, WAS has incorporated into it Java batch function (it's under "System Administration") and there you'll see a checkbox for SMF usage. SMF is a z/OS-only function. So it is yet again an example of z/OS specific elements coming to the surface.

But overall the Admin Console for WAS z/OS is like the Admin Console on other platforms. Common things like deploying servers or setting environment variables is exactly the same.

Administrative Application - Smart XML Updater

It does other things, but a large portion of the Administrative Application's function is to know how to translate mouse clicks into XML updates:



***Which* XML files not always obvious**

Often *more than one* XML file needs updating

You could try to do this yourself ...

Better to allow administrative function to do it

Config file system ...

We like to say that the Admin Console (and more precisely, the administrative function behind the browser interface that is the Admin Console) is really a very smart XML updater function. We say that because the various mouse-clicks and keystrokes you use in the Admin Console ultimately get placed into XML files down in the configuration file system.

WAS's view of its world -- the entirety of its knowledge about its configuration -- is contained in the configuration file system. If you add to or update that world through the Admin Console, your updates are propagated down into XML so WAS can be aware of the changes.

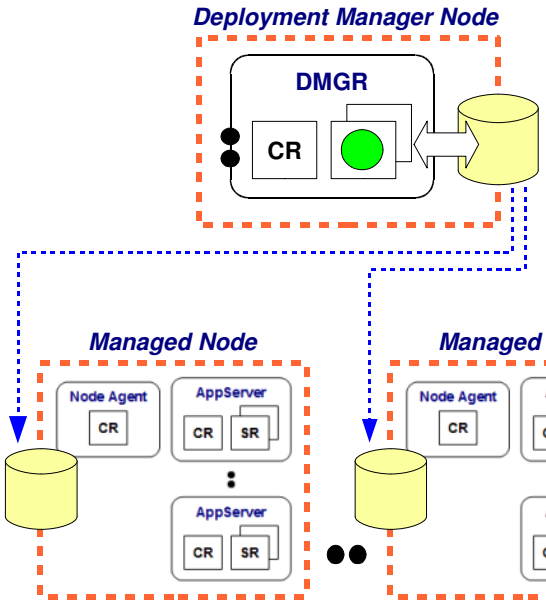
It might be tempting to think this means you can accomplish configuration updates by manually updating XML. The problem with that is several-fold -- first, it's not always obvious which XML files are to be updated. The administrative function knows ... it's smart, as we said. Second, many changes result in multiple updates across multiple XML files. Unless you know where all the updates are to be made you might well leave something out and then WAS will have trouble. It is far, far better to allow the administrative function to make these changes.

Note: there may be times when IBM Level 2 support indicates a manual change to an XML file is in order. If so, then follow their directions. But otherwise, do not make manual changes to configuration XML files.

Configuration File System

Each Node Has a Configuration Structure

In a Network Deployment (ND) configuration, each node has its own configuration file structure. DMGR owns the "master" and nodes are subordinate to that:



Master Configuration

- **Is maintained in a USS file system (HFS or ZFS)**
- **Is updated by the Administrative Application**
- **Has all the information about the whole cell**
- **Updates to master are propagated to each node via act of synchronization**

Node Configuration

- **Is maintained in a USS file system (HFS or ZFS)**
- **Can be in same file system as DMGR, but we recommend separate for each node**
- **Is updated by the Node Agent during synchronization**
- **Has all the information its node**
- **Has *some* information about other nodes**

Common file system layout …

In the first unit we spoke of "nodes" and how they're a collection of servers on a server box or LPAR. We were a bit vague in why WAS has this notion of "nodes." Here we'll start to clear up the concept a bit.

WebSphere Application Server on all platforms has a "building block" approach to constructing what's called a "Network Deployment" configuration -- that is, a configuration that has a Deployment Manager and servers on several servers or LPARs. That building block approach involves (a) creating the DMGR, (b) creating a node and joining it to the DMGR, and then (c) repeating step "b" as many times as you need to construct the desired topology.

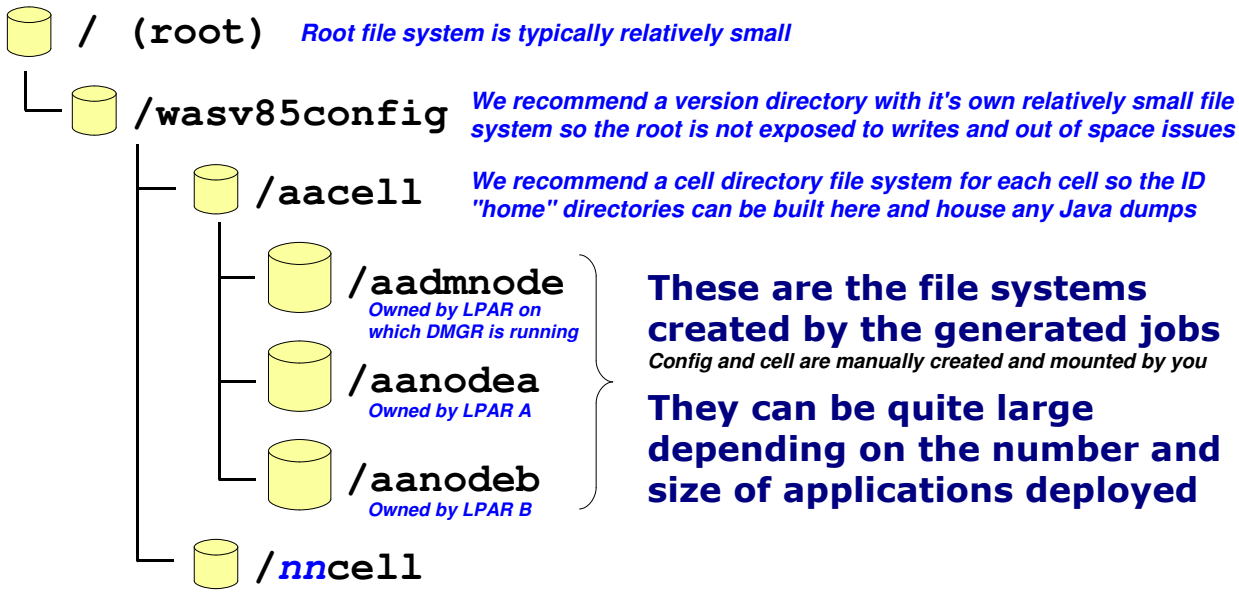
And therein lies the key point -- the construction of a node on z/OS involves the running of batch jobs that create a configuration file system and populate it with directories and XML files customized to your specifications. Therefore, each node has a configuration file system associated with it. That's true of application server nodes (called "managed nodes" because they're managed by a DMGR), as well as the DMGR itself. The picture shows an example of this -- a DMGR with a configuration file system (called the "master configuration") and two managed nodes, each with a configuration file system.

Note: on z/OS in a Sysplex it is possible to have one very large USS file system that holds all three nodes in this example. We do not recommend that for a number of reasons. We recommend each node have its own separate file system.

The master configuration holds information about *everything* ... the DMGR as well as every managed node. Each node's configuration file system holds complete information about itself and at *some* information about the other nodes. As noted earlier, updates made through the administrative function are written first to the master configuration, then copied out to the nodes through the act of synchronization.

How We Generally See the File Systems Deployed

Here's a picture that shows how the file system would be created and mounted based on the jobs generated using the PRS4686* planning spreadsheets:



V8.5 spreadsheets

TechDocs PRS4944

DMGR file system ...

What we're showing here is our recommendation for how to deploy the WAS z/OS files systems off the root. This is how the WAS V8 planning spreadsheets will attempt to do it by default.

Imagine you have a root file system. Typically they are not very large. Generally speaking you do not want the root file system to be exposed to getting filled up and running out of space. So we recommend you create a file system that will mount at `/wasv85config`. The mount point `/wasv85config` will be the starting point under which however many WAS z/OS V8 configurations you build. The file system mounted here protects the root file system from receiving any spurious write activity from the configurations. (That's rare, but we've seen it happen based on other problems with configurations.)

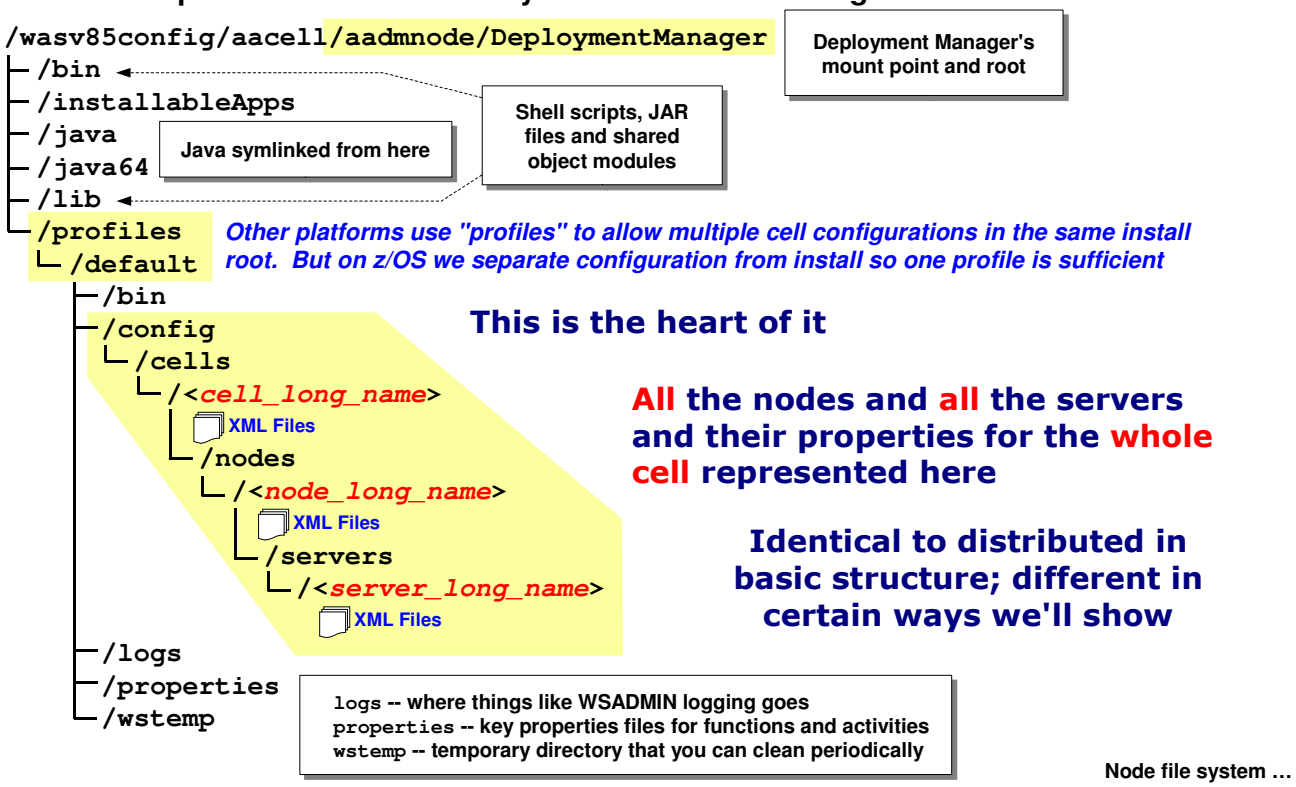
Under `/wasv85config` we recommend you create a mount point and a file system for each of the cells you create. The servers run with a small set of z/OS userids which require an OMVS segment (UNIX awareness) and a UNIX home directory. We recommend this home directory be placed under the cell directory ... that keeps the ID home directories organized with the cells. The home directories are where any Java dumps will go, so having them in a separate file system protects other file systems from out-of-space conditions.

Note: what we've outlined above are *recommendations* and not technical requirements. There's considerable flexibility in how you design the file system layout. We find this works. And it's in common use by many WAS z/OS customers.

Finally, under the cell directories go the node file systems. They are created by the jobs customized using the spreadsheet and the WCT. One of the jobs allocates and mounts the file system, another populates it with key directories and file systems. The final job makes all the updates needed to make that node fully customized to your settings.

The Essential Structure of DMGR Configuration Tree

Here's a snapshot of some of the key elements of the configuration structure:



We'll now take a look at the Deployment Manager's "master configuration" and the structure you'll find inside that. This is not an exhaustive picture. There are literally hundreds of directories and thousands of files. The picture illustrates a high level.

The `/bin` and `/lib` directories are where shell scripts, jar files and shared object files are located. But they're not physically there ... what's in those directories are *symlinks* over to the actual files housed in the product file system ... the *`hlq`*`.SBBOHFS` we spoke of earlier. The same holds true for the `/java` and `/java64` directories. They have symlinks because it allows common product files to be housed in one place (*`hlq`*`.SBBOHFS`) with many nodes simply linking over to them. We'll explore this symbolic link structure in a bit, and as part of that we'll discuss *intermediate symbolic links* which provide a way to isolate nodes for maintenance purposes.

Down under `/profiles/default` you'll being to see the cell's configuration tree. WAS z/OS has one profile because it separates the product install root from the configuration install root. WAS z/OS does not need to create different profiles for each cell. Each cell uses `/default` and simply links over to the product install root.

Here's where we start to see where the long names are used as part of the configuration structure. This is common across all platforms. The long names are used as directory names. XML files for the cell level are kept under the cell long name directory; XML files for each node are kept under that node's long name directory; and the same holds for servers.

Down at the bottom we see some directories for logs and properties files. The /wstemp directory is used to hold temporary administrative files. Over time this gets cluttered and you may go in and clean out directories

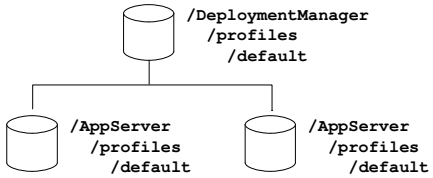
This picture is representing the DMGR's master configuration. As such it holds a picture of the entire cell, and includes every XML file for every node and server in the whole cell. On the next chart we'll see how each node has a *similar* file system, but without everything for everybody.

Each Managed Node Has Similar Configuration

The key difference is in that it maintains only partial awareness of *other* nodes but *full awareness of itself:*

```
/wasv85config/aacell/aanodea/AppServer
├─ /bin
├─ /installableApps
├─ /java
├─ /java64
├─ /lib
└─ /profiles
   └─ /default
      ├─ /bin
      ├─ /config
      │  └─ /cells
      │     └─ /<cell_long_name>
      │        ├─ XML Files
      │        └─ /nodes
      │           └─ /<node_long_name>
      │              ├─ XML Files
      │              └─ /servers
      │                 └─ /<server_long_name>
      │                    └─ XML Files
      ├─ /logs
      ├─ /properties
      └─ /wstemp
```

The mount point and the node root are different because this is a different node from the Deployment Manager node



It will have a directory for each node. Other nodes have some meta-data files that provides the partial information.

But the node directory for itself and the servers under it are fully populated with detailed XML

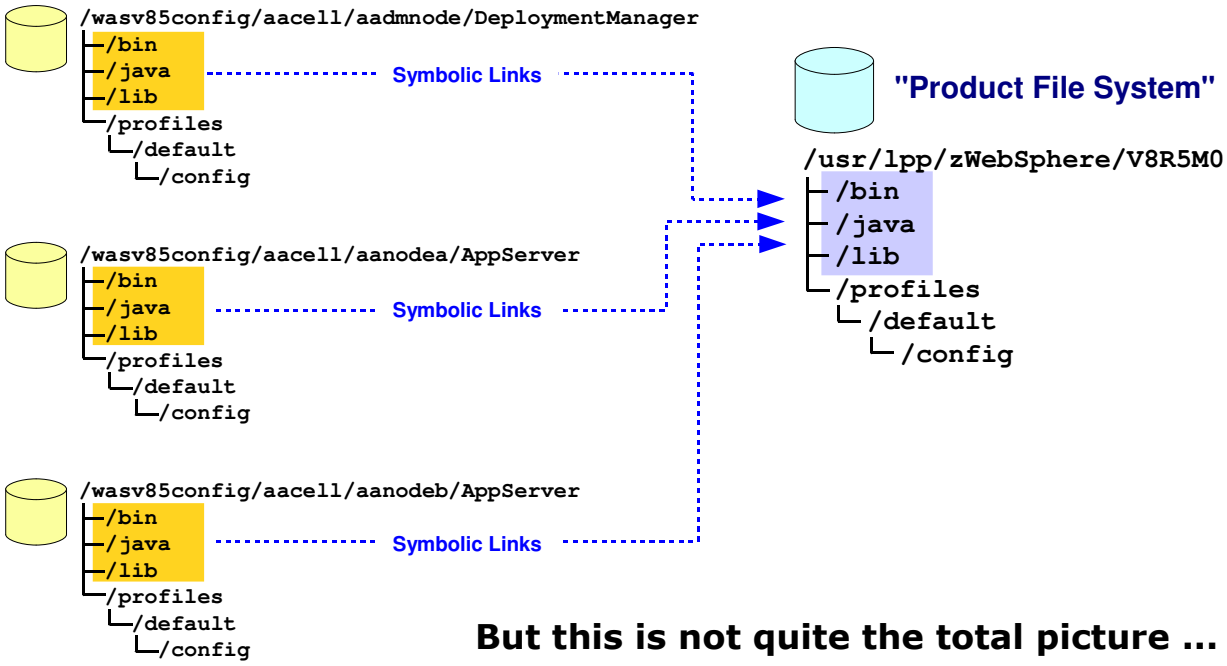
Relationship to install image …

A very similar picture to the previous chart but with some differences. The mount point is different ... rather than being the mount point for the Deployment Manager, this reflects the mount point for the specific node. And further down you'll find that the node and server directories map to the node and long names for this node, but information about other nodes is considerably less detailed. There's some information about other nodes, but in general each node knows full well about itself, but only something about the other nodes.

The little graphic shows the orientation of the file systems -- one for the DMGR, then one for each node. The DMGR file system is "master" in that it has information about everything. Each node has information about itself and at least some awareness of others nodes.

Relationship to "Install Image"

The "install image" is the file system that contains the product binaries. The configuration file systems link to that via symbolic links:



But this is not quite the total picture ...

Intermediate symlinks ...

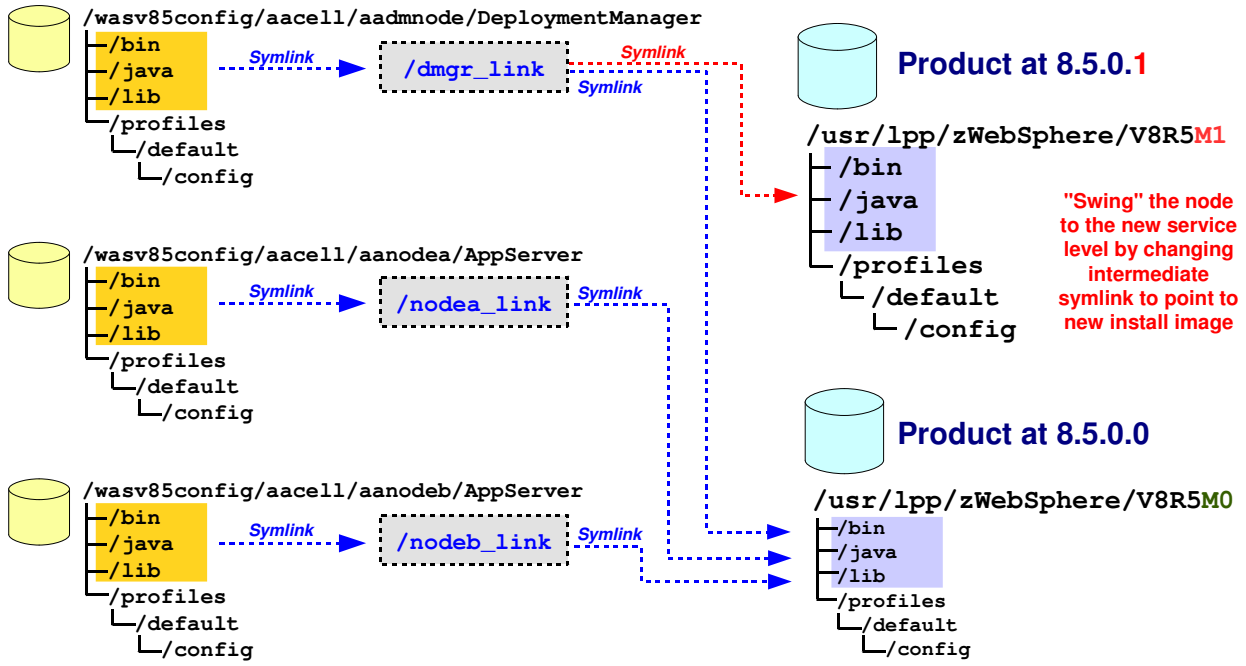
We mentioned earlier how many of the directories and files in the configuration file system are really just symbolic links to the product file system. That's done because duplicating every file in every node configuration file system would waste storage space and complicate maintenance considerably. So the symbolic links are there that point to the location of the comparable directories in the product file system.

Note: one difference between WAS z/OS V7 and V8 is that many of the symlinks are now moved up to the directory level whereas before they were file by file. That bit of information has very little impact on your life as a WAS administrator, but we offer it as a bit of trivia and so you understand why things might look a little different.

But this picture is not the total picture as we recommend it be drawn. If we ended this discussion with this picture you'd be left with an environment where maintenance applied to the product file system would result in updates to all the symlinked nodes. If you're looking to "roll" maintanence through a Sysplex environment this "big bang update" is definitely *not* what you want. You'd want more isolation. Which is why we have the concept of *intermediate symbolic links*, which is explained next.

Intermediate Symbolic Links

We've taken this symlink structure one step further by introducing an "intermediate symbolic link" for each node between the node and the install image:



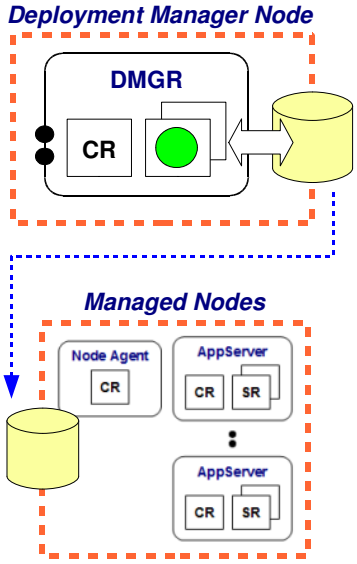
Intermediate symbolic links are just that ... symbolic links that sit at a spot between the configuration file system and the product file system. The benefit of this is it provides a way to change the product file system linked to on a node by node basis rather than having a "big bang" update across all symlinked nodes.

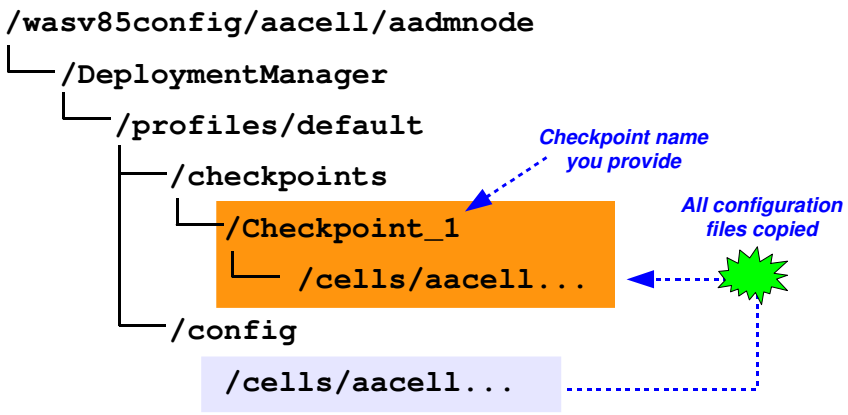
Note: The Planning Spreadsheet and the WCT tool support the concept of intermediate symbolic links. The customized jobs that are produced by the WCT will create the intermediate symlinks based on the input the WCT receives from the spreadsheets. Intermediate symbolic links are a fairly mainstream thing with WAS z/OS.

Imagine you use Installation Manager (IM) to create a new `hlq.SBBOHFS` at the 8.0.0.1 fixpack level. and you copy that out of your `/SERVICE` zone to a mount point that reflects the WAS level. To change one node to the new level of WAS you would simply stop the servers in that node, delete the intermediate symlink and recreate it so it points to the new WAS 8.5.0.1 product file system mount point. Then when the first control region in that node starts it'll spot the new level of was and run the `applyPTF.sh` update shell script and update the configuration file system so it matches the level of WAS it sees in the target.

Repository Checkpoints in Concept

It's fairly simple ... the Admin function now provides a way to "take a snapshot" of the master configuration and restore back to previous snapshots if you wish:





Notes:

- **Checkpoint is performed through Admin Console***
- **Location where checkpoints stored is configurable**
- **Multiple checkpoints possible**
- **Restore selected checkpoint through Admin Console***

*** or WSADMIN**

Taking a repository checkpoint ...

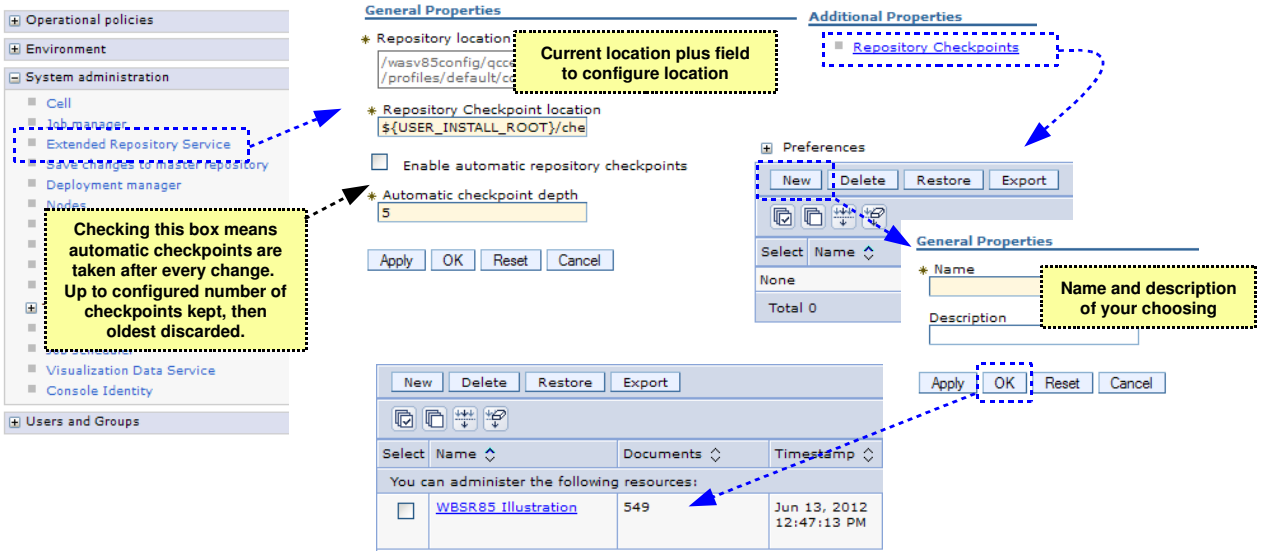
With WAS V8.5 (all platforms) comes a function that was originally part of the Virtual Enterprise product. It provides a way of taking a "repository checkpoint," which is a saved snapshot of the master configuration repository. You may then restore back to a known checkpoint if you determine the changes you've made since that point are not desired.

In concept it's relatively simple ... this operates in the Deployment Manager's "master configuration" and a checkpoint is a snapshot copy of all the configuration artifacts (directories and files) under the `/cells` directory for the cell. The checkpoint is kept under a configurable location and under a checkpoint name you may set.

You may maintain multiple checkpoints and restore back to any of the checkpoints that you have saved.

Admin Console for Repository Checkpoints

Some bitmap captures that illustrate the process of taking a checkpoint backup:



Checkpoints may then be "restored" to fall back to a previous configuration checkpoint. Configuration reverts to those settings.

Restoring a saved checkpoint …

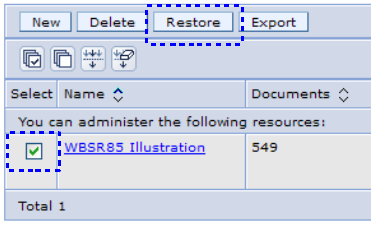
This chart shows the sequence of Admin Console clicks you would use to take a repository checkpoint.

There is an option to have the administrative function take automatic checkpoints. Every time you make a change it would take a checkpoint, and it would maintain up to the configured number of delta checkpoints. Once the configured depth is reached, then the older checkpoints are discarded.

Note: this "checkpoint depth" concept applies only to automatic checkpointing. Manual checkpointing, which is what the chart is illustrating, has no limit. You may maintain as many checkpoints as you have storage resources to hold the checkpoints. And the checkpoints may be configured to go to a location outside of the DMGR's configuration file system, so it's possible to maintain a very large file system at an external location and hold a great many configuration checkpoints.

Restoring Checkpoints

A couple of notes regarding this ...



Restoring means the checkpoint configuration files and directories are copied back to the master configuration's `/config/cells` path

Updates to node configuration file systems through normal synchronization process

Process will synchronize with the nodes if auto-synchronize is set for the Node Agents. If not, remember to manually synchronize to the nodes.

You may need to log off the Admin Console and back on to see the restored configuration artifacts in the Admin Console display

Restore puts configuration files back in place, but it does *not* restart servers or applications that were deleted and then restored

You should *still* have a solid backup/restore process in addition to this checkpoint function

MODIFY command ...

The taking of a checkpoint implies copying directories and files to a checkpoint location. *Restoring* the checkpoint implies copying those checkpoint files *back* to the normal `/config/cells` location in the master configuration.

If you made changes that were synchronized out to the node configuration file systems, them restoring a checkpoint implies the undone changes needing synchronizing out as well. If automatic synchronization is turned on then the system will take care of that when you restore a checkpoint. But if you have automatic synchronization turned off, then it'll be up to you to manually synchronize the restored configuration back to the nodes.

Restoring copies files back, but it does not restore operational state. The best way to understand this is to think of a scenario:

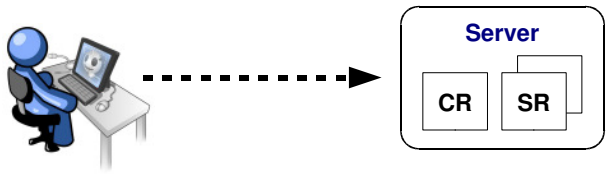
- You take a checkpoint of a configuration that has a server and an application
- You delete the server and the application ... moments later you say "ooops" ☺
- You then restore the checkpoint, which restores the server and the application
- *You will find the configuration is restored, but the server is not started nor is the application started*

So when you restore a checkpoint you may need to sweep through and start servers or applications to get the operational state of the environment back to where you need it.

z/OS MODIFY

MODIFY Facility of z/OS Operating System

MODIFY is a means of dynamically displaying information about started task, or dynamically updating the runtime settings for that started task



F <jobname>,*keyword*,*keyword*...

```
F Z9SR01A,HELP
F Z9SR01A,HELP
BBOO0178I THE COMMAND MODIFY MAY BE FOLLOWE
BBOO0179I CANCEL - CANCEL THIS CONTROL REGI
BBOO0179I TRACEALL - SET OVERALL TRACE LEVE
 :
BBOO0179I DISPLAY - DISPLAY STATUS
 :
BBOO0179I WLM_MIN_MAX - RESET WLM MIN/MAX SERVANT SETTINGS
BBOO0179I RECLASSIFY - RE-PROCESS WLM CLASSIFICATION FILE
 :
BBOO0179I FAILOVER - FAILS OVER CONNECTIONS FOR RESOURCE IDENTIFIED BY GIVEN JNDINAME
BBOO0179I FAILBACK - FAILS BACK CONNECTIONS TO RESOURCE IDENTIFIED BY GIVEN JNDINAME
```

Example of output generated by simply specifying HELP on the MODIFY

35 MODIFY commands for WAS z/OS

18 DISPLAY options

InfoCenter **rxml_mvsmodify**

First 18 commands ...

The z/OS operating system has for a very long time now provided a means of issuing operator commands against jobs and started tasks to either modify the runtime behavior or to display information about the task. WAS z/OS takes advantage of this and provides a rich set of MODIFY commands.

The z/OS command for MODIFY is "F" -- "M" having been reserved for a different function. The command is issued against a particular server JOBNAME, and takes as parameters a string of comma-separated keywords.

A good place to start with MODIFY is the HELP keyword. This will display a list of available MODIFY commands acceptable for use against the jobname supplied. For WAS z/OS V8 you will see 35 such MODIFY commands. One of those commands is DISPLAY, which itself has 18 options.

In short, there's a great deal of power and information you can derive from the MODIFY command of WAS z/OS.

The key here is not to make you an expert in all of those commands. Rather, we will take you on a brief survey of the commands and point you to the InfoCenter article that details each and every command.

MODIFY Commands, Part 1

Here's the first 18 of 35 `MODIFY` commands available with WAS z/OS V8:

```
CANCEL - CANCEL THIS CONTROL REGION
TRACEALL - SET OVERALL TRACE LEVEL
TRACEBASIC - SET BASIC TRACE COMPONENTS
TRACEDETAIL - SET DETAILED TRACE COMPONENTS
TRACESPECIFIC - SET SPECIFIC TRACE POINTS
TRACEINIT - RESET TO INITIAL TRACE SETTINGS
TRACENONE - TURN OFF ALL TRACING
TRACETOSYSPRINT - SEND TRACE OUTPUT TO SYSPRINT (YES/NO)
DISPLAY - DISPLAY STATUS
TRACE_EXCLUDE_SPECIFIC - EXCLUDE SPECIFIC TRACE POINTS
JAVACORE - GENERATE JVM CORE DUMP
HEAPDUMP - GENERATE JVM HEAP DUMP
JAVATDUMP - GENERATE JVM TDUMP
TRACEJAVA - SET JAVA TRACE OPTIONS
TRACETOTRCFILE - SEND TRACE OUTPUT TO TRCFILE (YES/NO)
MDBSTATS - MDB DETAILED STATISTICS
PAUSELISTENERS - PAUSE THE COMMUNICATION LISTENERS
RESUMELISTENERS - RESUME THE COMMUNICATION LISTENERS
```

Specifying `,HELP` on many these will then display the parameters acceptable for that particular command

We'll focus on the DISPLAY command in a moment

InfoCenter `rxml_mvsmodify`

Next 17 commands ...

The list of MODIFY commands produced with HELP is too long to show on one page without the font becoming too small to read. So we'll split this across two pages.

As you see, the MODIFY,HELP command produces a brief description of what the command achieves.

Many of these commands have required parameters. You may see a list of those by supplying `,HELP` after the command ... for example, F Z9SR01A,DISPLAY,HELP will list all the DISPLAY command parameters. Alternatively, you may go to the InfoCenter article shown and see the same information provided there.

Speaking of DISPLAY, we will provide some additional focus on that MODIFY command because it provides a very nice set of information about your server. But first, the final 17 MODIFY commands ...

MODIFY Commands, Part 2

Here's the final 17 of 35 MODIFY commands available with WAS z/OS V8:

```
STACKTRACE - LOG JAVA THREAD STACK TRACEBACKS
TIMEOUTDUMPACTION - SET TIMEOUT DUMP ACTION
TIMEOUTDUMPACTIONSESSION - SET TIMEOUT DUMP ACTION SESSION
TIMEOUT_DELAY - SET TIMEOUT DELAY VALUE
WLM_MIN_MAX - RESET WLM MIN/MAX SERVANT SETTINGS
SMF - SET SMF120 OPTIONS
DPM - DISPATCH PROGRESS MONITOR
RECLASSIFY - RE-PROCESS WLM CLASSIFICATION FILE
TRACERECORD - SET TRACE RECORD WRITE OPTIONS
MSGROUTE - SET ROUTING LOCATION OPTIONS
FORMFEED - ISSUE FORMFEED TO SYSOUT AND SYSPRINT
DISABLEFAILOVER - DISABLES FAILOVER SUPPORT FOR RESOURCE IDENTIFIED BY GIVEN JNDINAME
ENABLEFAILOVER - ENABLES FAILOVER SUPPORT FOR RESOURCE IDENTIFIED BY GIVEN JNDINAME
FAILOVER - FAILS OVER CONNECTIONS FOR RESOURCE IDENTIFIED BY GIVEN JNDINAME
FAILBACK - FAILS BACK CONNECTIONS TO RESOURCE IDENTIFIED BY GIVEN JNDINAME
SETOLATRACE - SET OLA TRACE LEVEL. SETOLATRACE=0..2, RGE | REGNAME | JOBNAME =x...x
SETOLATRACEPROPS - READ OLA TRACE PROPERTIES FILE
```

Specifying ,HELP on many these will then display the parameters acceptable for that particular command

InfoCenter **rxml_mvsmodify** **DISPLAY ...**

Here are the final 17 MODIFY commands. Many of these we will be using in the upcoming lectures and labs.

The DISPLAY Command

A particularly useful MODIFY command is DISPLAY, which has keywords that allow you to display specific information about the server:

```
F Z9SR01A,DISPLAY,HELP
BBOO0178I THE COMMAND DISPLAY, MAY BE FOLLOWED BY ONE OF THE FOLLOWING KEYWORDS:
BBOO0179I SERVERS - DISPLAY ACTIVE CONTROL PROCESSES
BBOO0179I SERVANTS - DISPLAY SERVANT PROCESSES OWNED BY THIS CONTROL PROCESS
BBOO0179I LISTENERS - DISPLAY LISTENERS
BBOO0179I CONNECTIONS - DISPLAY CONNECTION INFORMATION
BBOO0179I TRACE - DISPLAY INFORMATION ABOUT TRACE SETTINGS
BBOO0179I JVMHEAP - DISPLAY JVM HEAP STATISTICS
BBOO0179I WORK - DISPLAY WORK ELEMENTS
BBOO0179I ERRLOG - DISPLAY THE LAST 10 ENTRIES IN THE ERROR LOG
BBOO0179I MODE - DISPLAY THE EXECUTION BITMODE
BBOO0179I THREADS - DISPLAY THREAD STATUS
BBOO0179I ADAPTER - DISPLAY OLA ADAPTER STATUS
BBOO0179I OLATRACE - DISPLAY ADAPTER TRACE RECORDS. OLATRACE=* or jobname
BBOO0179I WLM - DISPLAY WLM SETTINGS
BBOO0179I SMF - DISPLAY SMF120-9 SETTINGS AND STATUS
BBOO0179I FRCA - DISPLAY FRCA INFORMATION
BBOO0179I DPM - DISPLAY DISPATCH PROGRESS MONITOR SETTINGS
BBOO0179I TRACERECORD - DISPLAY TRACERECORD SETTING
BBOO0179I MSGROUTE - DISPLAY MESSAGE ROUTING SETTINGS
```

Specifying ,HELP on many these will then display the parameters acceptable for that particular command

WSADMIN ...

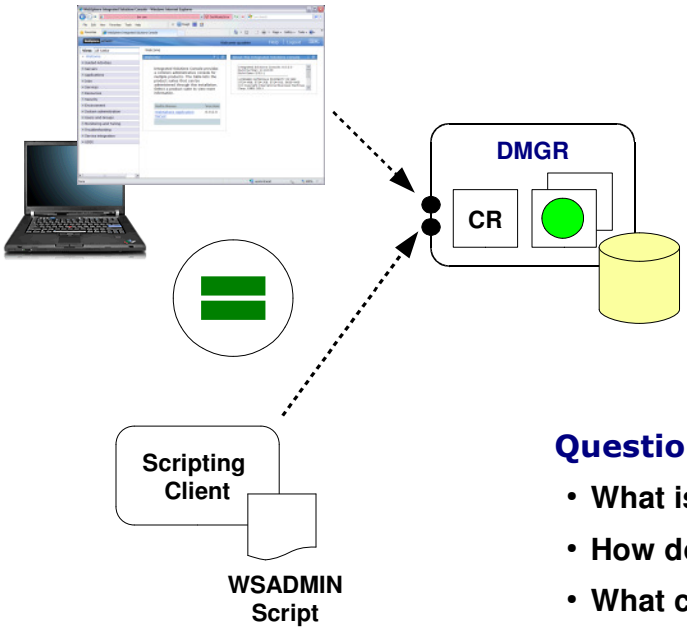
The DISPLAY command is of particular interest because it provides you with information about your server. There are 18 DISPLAY commands provided in WAS z/OS V8.

Again, many of these have required parameters. To see what those parameters are you may issue a ,HELP after the DISPLAY command, or go to the InfoCenter.

WSADMIN

In a Nutshell, WSADMIN is ...

... a set of interfaces to the administrative function you may use to automate tasks you might otherwise do with the Admin Console:



Updates to the master configuration, then syncronization out to the nodes

Questions that spill from this:

- What is it?
- How does it work?
- What can I do with it?

Command objects ...

You saw how the administrative function has a browser-based interface that allows you to make configuration updates.

WSADMIN is essentially a programmatic interface to the same administrative function. Rather than mouse clicks at a browser screen a script invokes the programmatic interface commands to achieve the same results.

For the next several charts we're going to explain what WSADMIN is, how it works and what you can do with it. It can be a very complex topic, depending on how detailed one wants to get. For the purposes of this workshop we're going to touch on some of the key concepts and point you to a wealth of other documentation on the subject.

The WSADMIN Command Objects

The interface is composed of four main "objects" (commands) that provide the administrative function:

Think of AdminTask as commands that contain other more "primitive" WSADMIN commands under the wrapper. It was created as a way to make scripting easier for common tasks ... hence the name "AdminTask"

Key Points:

- **WSADMIN is a command interface**
- **Four major commands, each with many sub-options**
- **Your script uses these commands to make the changes you wish**

A very simple example ...

The programmatic interface offered by WSADMIN comes in the form of four command objects, each with a long list of sub-commands (or "methods") that do different administrative functions.

Note: there is in reality a fifth command object -- `Help`. It's not on this chart so the focus can be on those command objects that perform the administrative functions. But do not forget about `Help` because it can be useful at times.

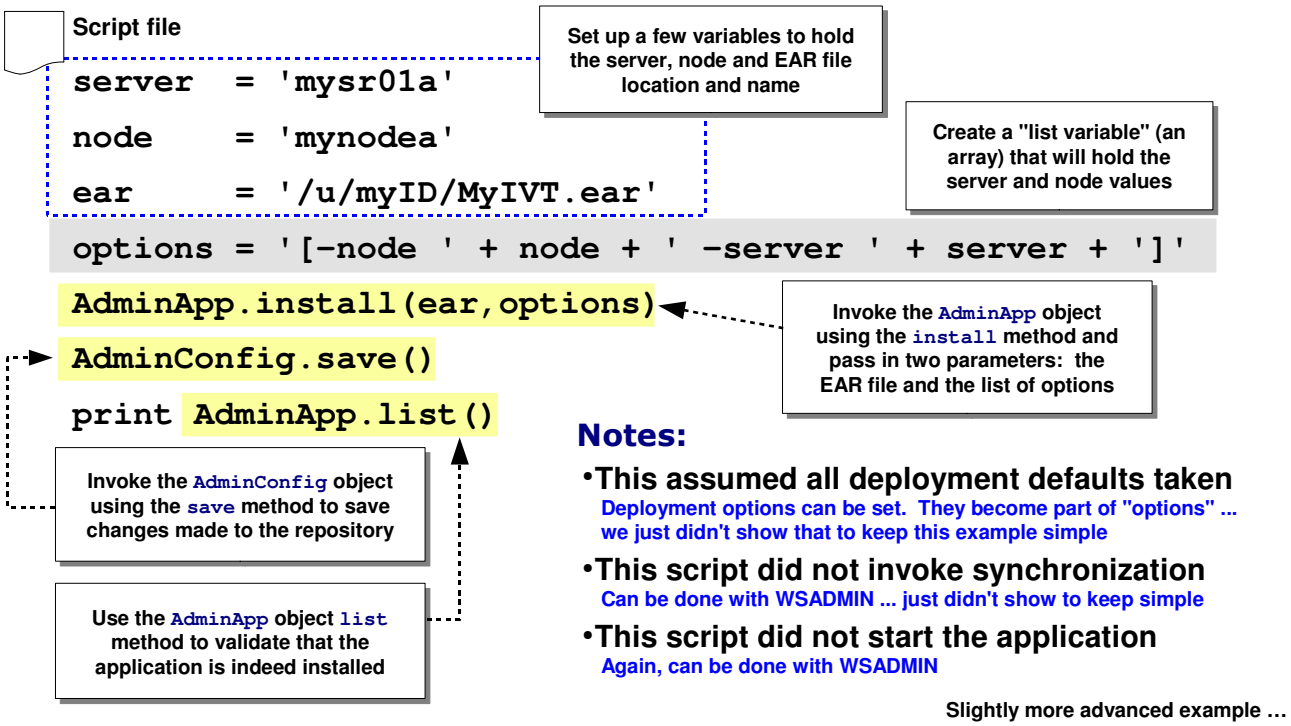
For example, if you wanted to list all the applications in a cell you'd use `AdminApp.list()`. If you wanted to uninstall an application you'd use `AdminApp.uninstall()`. The names of the command objects reflect what their focus is ... `AdminApp` relates to applications; `AdminConfig` to various configuration elements; and `AdminControl` to operational activities.

`AdminTask` is a bit unique in that what it provides is a set of pre-packaged ways to do complex things easier. For example the `changeHostName` method does with one command which used to take dozens of commands before `AdminTask` became available.

What's not show on this chart is an example of a Jython script that would use these commands to accomplish some administrative task. Let's look at a very simple example of a script.

A *Very Simple* Example of Installing an Application

Automating the deployment of applications is a very common use for WSADMIN. Here's an example of a Jython script that installs an application:



This is a simple seven-line script that installs an application. It uses two command objects -- `AdminApp` and `AdminConfig`. `AdminApp` is used with two different methods -- `install` and `list`.

- The first three lines establish Jython variables and place into those variables some hard-coded information. The three variables are the server name, the node name and the path to the EAR file.
- The next line builds a Jython "list variable," which is like an array. This list variable is used to hold the option list that `AdminApp.install()` needs to install an application. The list is built by inserting the option *name* followed the variable that represents the option *value*. For example, the option list resolves to:

  ```
  -node mynodea -server mysr01a -ear
  ```

- `AdminApp.install()` is used to install the application. The contents of the `()` resolve to `/u/myID/MyIVT.ear` and the option list.
- Just like in the Admin Console an application may be installed but not yet saved. That's the case here, which is why `AdminConfig.save()` is used to save the changes. That method requires no options inside of `()`, which is why the command is simply `AdminConfig.save()`.

  Note: this script does *not* include anything to synchronize the changes to the nodes. That *is* possible with WSADMIN. It's just not shown here to keep this example simple.
- Finally print `AdminApp.list()` prints back what WAS sees as the installed applications. This is a visual validation that the installation was successful.

There's a lot of caveats around this ... which the chart calls out. This is a very simple example that shows the basic usage of a few command objects.

Let's look at a slightly more sophisticated example.

The App Install Script from Upcoming Lab

Uninstalls app if already present, then installs the named application again:

```
import sys
# --------------------------
#  Uninstall the app if it i
# --------------------------------
application = ""
applicationlist = AdminApp.list().splitlines()
for application in applicationlist:
    if application == "SuperSnoop":
        print "Found and uninstalling " + application
        AdminApp.uninstall(application)
        AdminConfig.save()
    if application != "SuperSnoop":
        print "Application in list not SuperSnoop. Skipping. Name is: " + application
    continue
# --------------------------------
# Install SuperSnoop
# --------------------------------
application = "SuperSnoop"
print "Installing application " + application
earfile = "/wasetc/was8lab/applications/SuperSnoopProj.ear"
appopts = "[-appname "
appopts = appopts + application
appopts = appopts + " -MapModulesToServers [[ SuperSnoopWeb SuperSnoopWeb.war,WEB-INF/web.xml "
appopts = appopts + "WebSphere:cell=z9cell,node=z9nodea,server=z9sr01a ]]]"
# -- debug if needed: comment out if not needed ----------
print "appopts = " + appopts
# -- invoke the AdminApp.install() method ---------
AdminApp.install(earfile, appopts)
AdminConfig.save()
print "Application                                    started"
```

For Loop

Brings in a set of support functions useful to Jython

Retrieves all installed applications in the cell. `splitlines()` splits into a list that can be iterated through

Loops through list. If it finds SuperSnoop it uninstalls it, otherwise it notes application skipped over

Pointer to location of EAR

Building the application install options list array

Debug print ... just to see what the option list passed to `AdminApp.install()` actually looks like.

Invoke `AdminApp.install()` and then save the changes to the Master. This does *not* synchronize.

WSADMIN client ...

Here is a more complex example, but not terribly more complex. This too installs an application but it has some other Jython processing logic wrapped around it:

- The `import sys` at the very top is used to import into the Jython script a set of helpful scripting utilities. The `splitlines()` function is one such utility. `AdminApp.list()` returns a simple string of installed applications. `splitlines()` will do just that ... split the line into a list that can be more easily parsed and iterated through.
- The script then drops into a "for loop" looking to see if the application "SuperSnoop" is already installed. If it is, then it's uninstalled using `AdminApp.uninstall()`. Otherwise, a simple print is issued indicating an application was found but it's not SuperSnoop.
- Then the script drops through to processing very much like the example we just showed. But the options list is more complex. The previous simple example used the minimum options on `AdminApp.install()`. For more options exist and might need to be used, depending on how complex your install is.
- The line that prints out the `appopts` variable is there simply as a debug step. Getting the syntax right on option lists is required for things to work well. A printout of the variable helps you spot syntax problems.
- Finally we drop through and use `AdminApp.install()` and `AdminConfig.save()` to install and save the changes. Here again, no synchronization is done and the application is installed but not started.

This example shows that Jython is capable of doing more than simply issuing WSADMIN commands. In fact, Jython is capable of doing fairly sophisticated logic. And this example shows that a script might have more Jython processing than WSADMIN processing ... the yellow blocks in the picture above show relatively little WSADMIN command usage compared to other Jython usage.

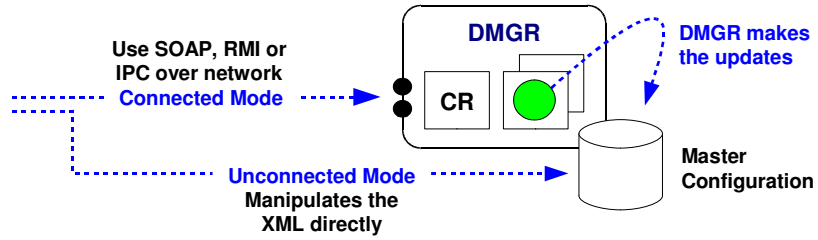
That's a little on scripting. Proficiency comes with practice and looking at samples. But how is the script invoked against the WAS administrative function? With the WSADMIN client ...

The WSADMIN Client Shell Script and Invocation

To use WSADMIN you must invoke the `wsadmin.sh` client. You pass in the script file you have written. It then works against the interface to do the work ...

```
/wasv85config/aacell/aadmnode/DeploymentManager
└/profiles
  └/default
    └/bin
```

wsadmin.sh



Connected Mode (Recommended whenever DMGR is available)

```
./wsadmin.sh -lang jython -conntype SOAP
   -host www.myhost.com -port 10002
   -user myadmin -password myadmin -f /u/myhome/myscript.jy args
```

Or RMI or IPC with corresponding port

Unconnected Mode

```
./wsadmin.sh -lang jython -conntype NONE -f /u/myhome/myscript.jy args
```

Passing in arguments ...

WAS comes with a shell script client for WSADMIN called `wsadmin.sh`. This is located down in the `/bin` directory. At the very highest level the shell script client takes as input parameters that indicate how it is to make the updates (connected mode vs. unconnected mode) and what script it is to run.

Note: Truth is, the options on `wsadmin.sh` are a bit more flexible than this. But what we just wrote about connected vs. unconnected and passing in a script file is the essence of it.

Connected mode implies the client establishing a network connection to a running Deployment Manager. The updates to the XML are not made by the client but rather by the administrative function that's running in the DMGR. The scripting client is talking to the administrative function which in turn makes the updates.

Unconnected mode is used when there is no DMGR running. The updates to the XML are made by the client by directly manipulating the files. To use unconnected mode you must invoke the copy of `wsadmin.sh` that's in the Deployment Manager's `/bin` directory. Connected mode may be invoked from anywhere provided there's a network connection and the security considerations we'll explain later are met.

Important: While it is possible to use unconnected mode with the DMGR up and running, it is strongly discouraged. The rule of thumb is this: if the DMGR is running, use connected mode. Allow the DMGR's administrative function to coordinate and make the updates. Use unconnected mode only if the DMGR is not running.

The chart shows the syntax for connected mode ... the shell script, followed by the scripting language, followed by the connection type, followed by the host and port, followed by the userid and password to authenticate into the DMGR, followed by the script file and any arguments passed into the script.

Unconnected mode is simpler ... scripting language, mode (NONE), file and arguments.

Passing in Arguments to a Script

Scripts may be made even more flexible by passing in arguments on the invocation command and using those passed-in arguments within the script

```
./wsadmin.sh -lang jython -conntype SOAP
   -host www.myhost.com -port 10002 -user myadmin -password myadmin
   -f /u/myhome/myscript.jy z9sr01a z9nodea /u/myID/MyIVT.ear
                             server   node    EAR location
```

```
import sys
server = ""
node   = ""
ear    = ""
if(len(sys.argv) > 2):
    server = sys.argv[0]
    node   = sys.argv[1]
    ear    = sys.argv[2]
else:
    print "Not enough arguments ... exiting"
    sys.exit()
 :
 :
```

Initialize variables

Check to see if there's enough arguments to satisfy the script's requirements. This script expects to see 3 variables. Check for greater than 2.

Parse the arguments and assign each to the respective Jython variable that will be used elsewhere in the script.

Note that sequence of arguments not enforced by Jython. You either assume it's correct or build in more error checking.

With arguments now held in Jython variables you may move on to the rest of your script processing

Script file character encoding ...

The previous chart suggested arguments can be passed into a script. Doing so means a script can be written to re-usable. Hardcoding variable values in the script limits the flexibility some.

Arguments are passed into a script after all the other `wsadmin.sh` options are given, and after the `-f` switch is used to specify the file to read in. They are passed in separated by spaces. The example shows three arguments passed in -- server, node and EAR. They are received in the script as a special variable called `sys.argv`.

The chart shows some sample code to process the arguments. First the variables that will hold the passed in values are initialized. Then some simple error checking is done -- check for the number of passed-in arguments and if less than the expected number is seen then take an error action. This script expects three arguments. By checking the length to make sure it's greater than 2 you capture the case of 3 and higher. If someone passes in six arguments the processing you see here will only process the first three. The rest are ignored.

If someone passes in 2 or less then the check fails and the script exits.

Next the arguments are parsed out of `sys.argv` and placed in their respective variables. Notice the 0 offset used by `sys.argv`. Notice also that this script assumes the sequence of the arguments is correctly passed in. If they were supplied in a different sequence then it's quite possible an error would occur when later using the variables with a WSADMIN command object. In this case the script is assuming things are correct. If that's not a safe assumption, then additional error checking would be needed.

Character Encoding of the Script file on z/OS

May be either ASCII or EBCDIC. WSADMIN by default expects ASCII. If you want to use EBCDIC you have to tell WSADMIN:

File stored in z/OS USS as ASCII

By default WSADMIN expects script file to have ASCII character encoding so the -javaoption is not needed if ASCII

File stored in z/OS USS as EBCDIC

```
./wsadmin.sh -lang jython -javaoption -Dscript.encoding=Cp1047
   -conntype SOAP -host www.myhost.com -port 10002
   -user myadmin -password myadmin -f /u/myhome/myscript.jy
```

WSADMIN and JCL batch …

In the examples leading up to this chart we showed how the `-f` switch is used to point to the location of the script file to be read in. If the `wsadmin.sh` script is invoked on z/OS, then that implies the script file is on z/OS as well. And that brings up the question of character encoding of the file.

It turns out you may store the file in either ASCII or EBCDIC encoding. WSADMIN expects ASCII by default. So if you store the file in EBCDIC you need to tell WSADMIN that. That's done with the `-javaoption` switch as shown in the chart. That switch comes after the `-lang` switch and before the `-conntype` switch.

WSADMIN and Batch

JCL invoking BPXBATCH works quite well ...

```
//WSADMIN JOB (ACCTNO,ROOM),REGION=0M,USER=MYADMIN,PASSWORD=MYADMIN
//STEP1    EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 BPXBATCH SH +
  /wasv85config/z9cell/z9dmnode/DeploymentManager/profiles/default+
  /bin/wsadmin.sh +
  -lang jython +
  -javaoption -Dscript.encoding=Cp1047 +
  -conntype SOAP +
  -host www.myhost.com +
  -port 10002 +
  -user MYADMIN +
  -password MYADMIN +
  -f /u/myID/myscript.jy args +
 1> /tmp/myID.out +
 2> /tmp/myID.err
/*
```

Complete pointer to the wsadmin.sh client

The invocation command is no different than before

This does bring up a few security issues we need to discuss ...

Security ...

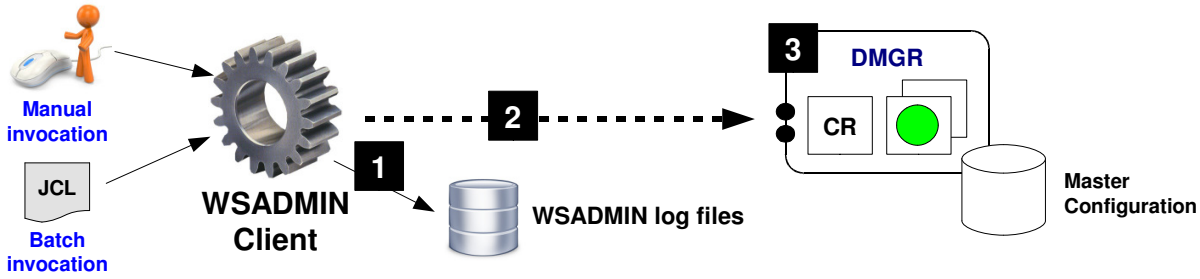
You might wonder whether WSADMIN can be invoked from JCL batch, and the answer is yes. BPXBATCH works perfectly well to launch a UNIX shell and invoke the shell script. The chart above shows an example of this.

One twist is the 72 character width limit of JCL. So line continuation symbols must be used. Those are the "plus" (+) signs at the ends of the lines.

Notice that you may pass in arguments in JCL as well. STDOUT and STDERR may be piped to a file as shown.

WSADMIN and Security

Within what we've discussed so far are three key security considerations that must be taken into account for WSADMIN to work:



1. **File permission access to WSADMIN log and trace files**
   Need write access, which requies at least GROUP access.
   This is the ID used to log into Telnet or USS, or the ID on batch JOB (or effective ID). WAS Admin ID
2. **Ability to establish SSL to DMGR when security enabled**
   Implies access to the CA certificate used to sign the DMGR's server certificate.
   This is the ID used to log into Telnet or USS, or the ID on batch JOB (or effective ID). WAS Admin ID
3. **Authentication and authorization to in the DMGR to perform the tasks**
   Valid RACF ID and proper access to EJBROLEs.
   This is the userid/password coded on the wsadmin.sh parameters. Again, WAS Admin ID.

Other IDs *can* be made to have these properties … WAS Admin ID has it by default

Resources …

There are three key security considerations to take into account when using WSADMIN in connected mode when global security for the cell is enabled. (If global security not enabled, or if you're using unconnected mode, then the security considerations drop to the first one list only.)

The main point being made on this chart is that if you wish things to work smoothly one good place to start is to use the WAS Admin ID and password both as the logged in ID when invoking `wsadmin.sh` (or supplied on the JOB card in batch JCL), and on the `-user` and `-password` switches. The logged in ID is what provides file access permissions as well as access to the keyring with the proper certificates. The `-user` and `-password` switches are what provide authentication into the DMGR as well as authority to invoke the functions in the DMGR.

You could make other IDs have these security authorities and keyrings and certificates. The point here is that the Admin ID will very likely have it as well and it's a good thing to use when first starting out.

Resources for Learning and Reference

The following resources are available to gaining more experience with WSADMIN:

IBM Techdocs -- ibm.com/support/techdocs

Techdocs Library > White papers >
WebSphere z/OS V6.1 - WSADMIN Primer (with Jython) WP101014

Techdocs Library > White papers >
Using Jython Scripting Language With WSADMIN WP100963

Techdocs Library > Hints, tips & Technotes >
Creating a New Server in WebSphere V7 for z/OS TD105447

Techdocs Library > White papers >
Staged Application Deployment in WebSphere on z/OS V7 WP101641

IBM InfoCenter -- publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp

Network Deployment (z/OS), Version 8.0 > Scripting the application serving environment (wsadmin)

Getting started with wsadmin scripting txml_script

Very good reference source for searches on specific WSADMIN commands or methods

WSADMIN client "Help" object and "help" methods

The WSADMIN client has extensive online help in its command syntax. It provides a way to drill down on syntax and usage for specific objects, method and attributes

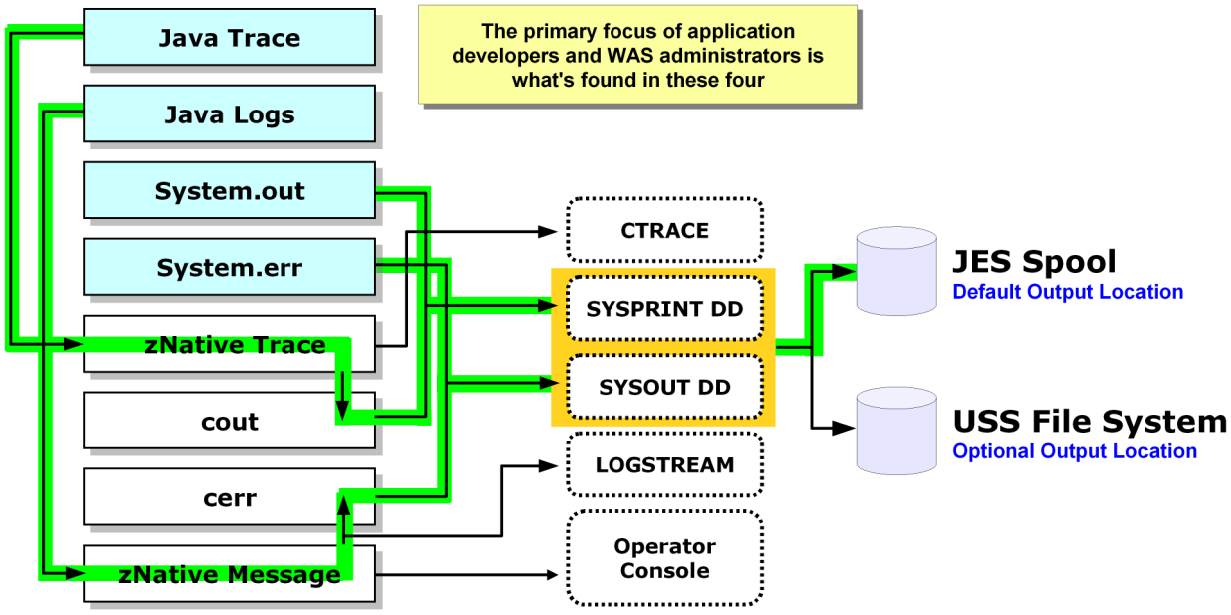
Logging ...

Some resources for learning WSADMIN.

Logging

The Default WAS z/OS Log and Trace Model

There are many sources of logging and tracing in WAS and WAS z/OS. This picture shows where output goes by default in WAS z/OS V8:



Key Point: **Multiple source of logging and tracing. On z/OS default routes most of it to SYSPRINT and SYSOUT, which then goes to JES spool.**

HPEL ...

This is a seemingly complex chart that's telling a somewhat simple message.

WAS z/OS writes output while it is operating. Anyone who's been near WAS z/OS has seen the messages written to the MVS operating console, and seen the output in the JES spool for the controller and the servant. What might not have been obvious is the various *sources* of that output. Those source can be broken down into two broad categories -- Java-based sources and z/OS native-based sources. The Java-based are represented by the light blue shaded boxes above and the native sources are the white boxes.

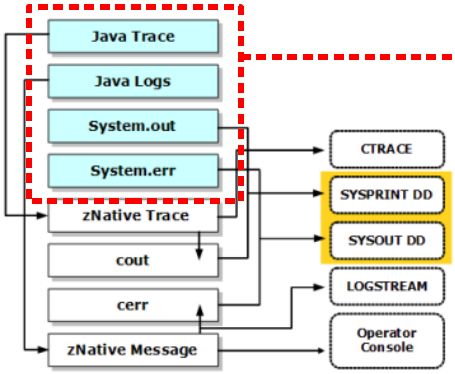
The green highlighting behind the arrows shows the typical routing for output from these various sources. What you see is that most by default will go to JES spool. There are things one could do to alter this ... setting up a real LOGSTREAM would be one; directing output to the CTRACE writer is another. But those are actions one has to take beyond the normal "default" actions most take when setting up WAS z/OS.

It's possible to direct a good portion of the JES spool output to a USS file system. Some do this because they want to give developers Telnet access rather than 3270 access to the output. But routing to a USS file has its downsides ... the facilities to automatically close the file and start a new one are lacking, for instance.

The point here is that for the most part, JES spool is where a great deal of WAS output goes. That's the starting point for the discussion of HPEL.

Introducing High Performance Extensible Logging

Or "HPEL" for short ... it is a new binary logging mechanism in V8 for all platforms. It provides a more efficient logging mechanism than clear text logging



WAS-specific binary format log file

Write to memory buffer, then file

Controls to dictate size limits, what to do when limit reached, how to trim files, start new files, etc.

Viewing the Log:

Admin Console View Facility

Usable tool to view binary file contents. Has ability to filter on criteria to limit what is seen

Log Viewer Shell Script Utility

File is logViewer.sh and it has parameters to limit what is seen in the produced text-readable output file

Download to PC

HPEL file is converted to readable text and downloaded as ZIP to your PC where standard text editors may be employed to view

Optional ... Traditional Mode still available and is default

Configurable on a server by server basis

High level of log viewer in Admin Console ...

The new High Performance Extensible Logging (HPEL) feature of WAS V8 (all platforms) provides an alternative place to log the Java-based components. Native component output still goes where it went before, which for z/OS is JES spool or the operator console.

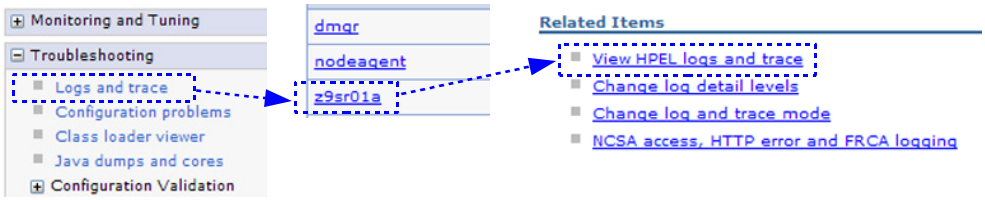
HPEL is a binary format logging facility, which means it writes less white space and it can be indexed and parsed more efficiently for filtering and display purposes. In addition, HPEL has built-in log management facilities such as size monitoring and daily closing of the logs and opening new ones.

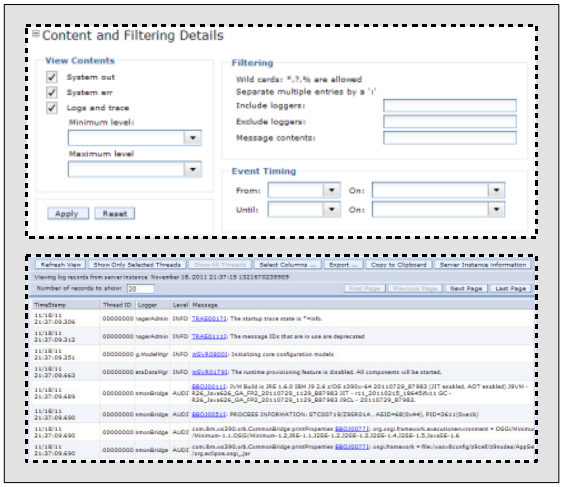
HPEL is configurable on a server-by-server basis. It is not required you use it, but if you choose to then you may do so for only those servers you wish.

The log can be viewed in a few different ways -- first, the Admin Console has a built in filtering and log viewer facility. We'll take a look at that in a moment. Second, there's a shell script utility that will parse the binary log and produce human-readable output. And third, there is a facility for downloading a text version of the HPEL log to yoru workstation where normal text editors can be used to browse, filter or find.

High-Level of the Admin Console Log Viewer

This is a graphical log viewer supplied as part of the Admin Console:





Log selection and content filtering section. This influences what appears below ...

... record by record display of HPEL content based on filtering down above.

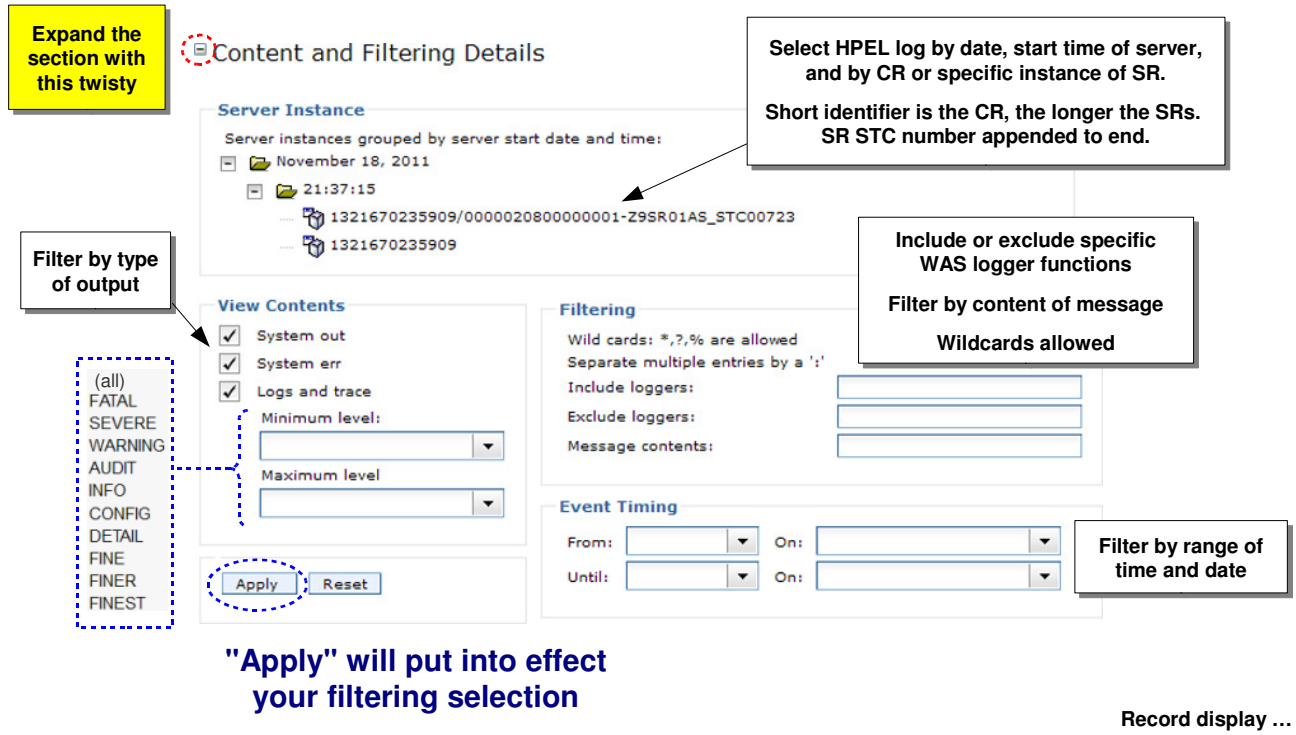
Content and filtering details ...

The log viewer that's provided with the Admin Console is accessible through the "Troubleshooting" section of the navigation tree. From there you select the server, then "View HPEL logs and trace."

The resulting log viewer has two main sections -- the top section provides a means of filtering the content shown by the viewer. You may elect to show everything, or you may elect to narrow the records shown by one a number of filtering rules found in this top section. We'll see details of this next. At the bottom of the screen will be the display of the records that match the filtering criteria. We'll see details of this as well.

Content and Filtering Details

This allows you to determine, with a fair degree of granularity, what HPEL records will be displayed in the output result set:



Here is what the content and filtering panel looks like. The first thing to note is you must expand the twisty to see this section. It's easy to overlook. Once this section is opened it will display a list of all the logs it sees in the log directory. You may have many logs here depending on (a) how often you stop and restart this server, and (b) how long it's been since you last cleaned out the logs. Server logs are closed out and new logs opened every 24 hours.

The first task is to select the log you wish to display. You'll see both CR and SR HPEL logs here. SR logs have the longer name. If you have multiple servant regions defined for a server and you're not sure which is which, note the Started Task (STC) number at the end. That will provide you a way to correlate the HPEL log to a specific servant instance.

You may filter by type (System out, System err, Logs and traces) as well as the level of the trace. For instance, if you just want to see if any SEVERE or FATAL messages are present, then filter on minimum=SEVERE and maximum=FATAL. You'll see just those records.

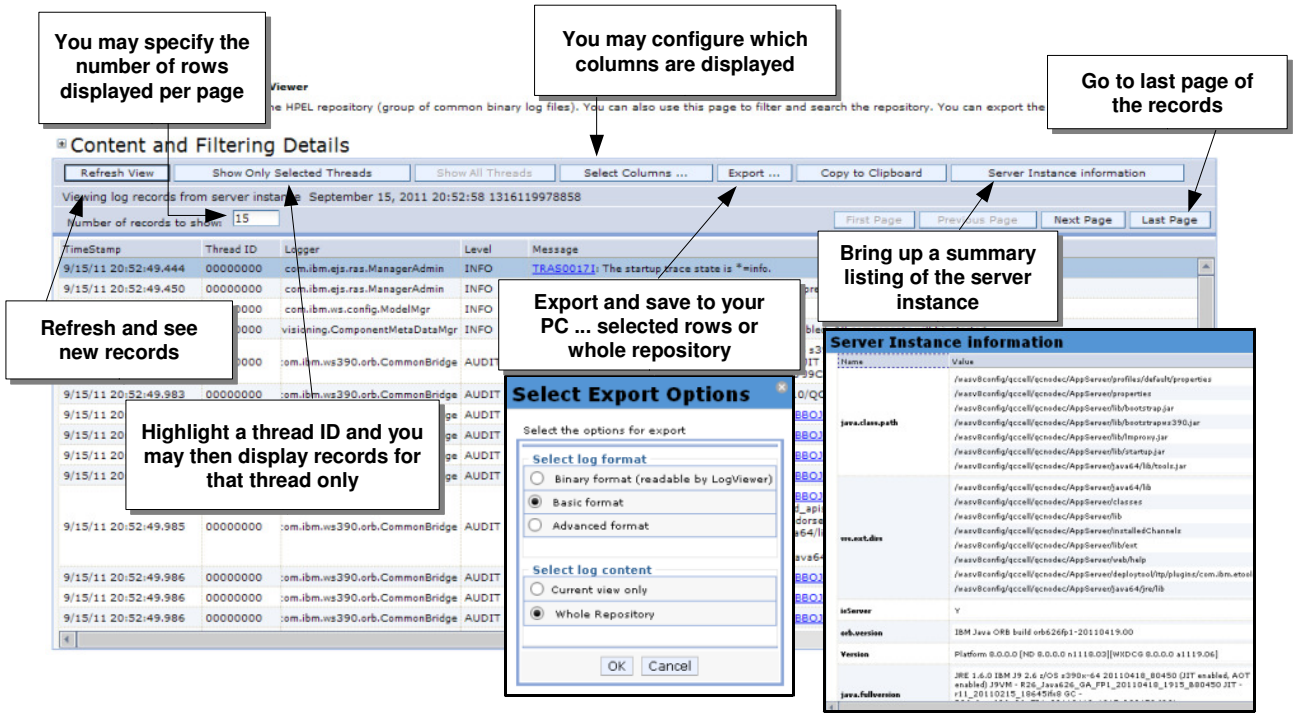
To the right you see where you may filter on the loggers. WAS consists of many internal components that log information. Say you know what logger you wish to filter on ... or suppose IBM Level 2 support requests to see only those records logged by a specific logger ... then you can include that logger name and see only those logs. Or you may search on message content using wildcards before or after your filter string.

Finally, you may filter on a window of time within the time represented in the 24 hour log.

Or you may use a combination of all these filtering methods to reduce the total number of records shown to just those you want to see. Then click "Apply" and your filtering is applied and those matching records are shown in the bottom half of the screen.

Record Display of Content Based on Filtering

This displays in your Admin Console for the selected server's HPEL log:



Example ...

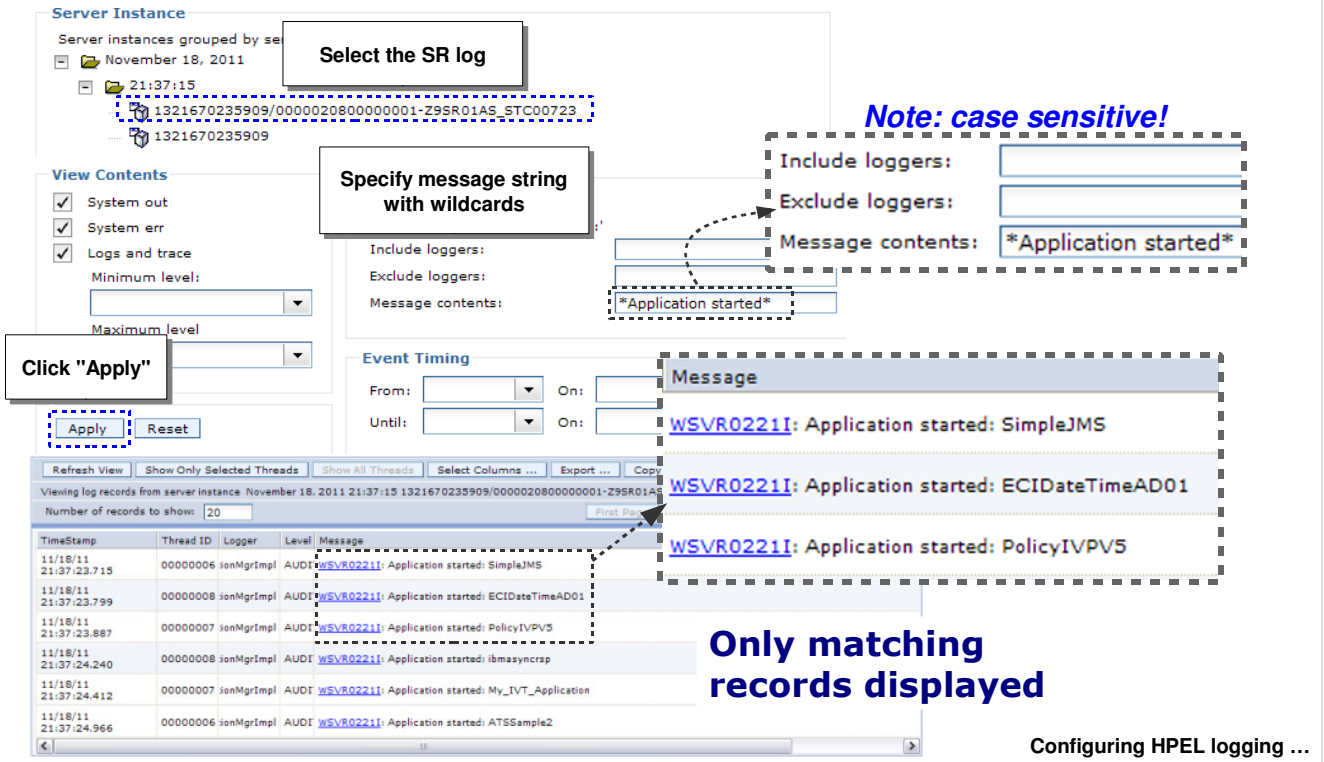
The records are diplayed in the lower portion of the log viewer, below the expandable "Content and Filtering Details" section we just looked at. By default all records in the log will be displayed. You may change many elements of this page layout, including the number of records displayed, the columns displayed, or to show only those records for a selected thread ID. The "Refresh" button reads the log and appends whatever new records may have come in since the last refresh.

It's possible to "export" the repository to a downloadable file and review it on your workstation ... either the whole repository or what portion is displayed on the log viewer.

Finally there's a button to bring up a nice summary of the server instance information. When reviewing logs and traces there often arises the need to understand what certain settings are. This button provides a place to retrieve all that information in a handy format.

Simple Example of Filtering

Suppose you wish to see all the "Application started" messages:



Configuring HPEL logging ...

Here's an example of using this facility. Suppose you just want to see all the "Application started" messages. Here's what you'd do:

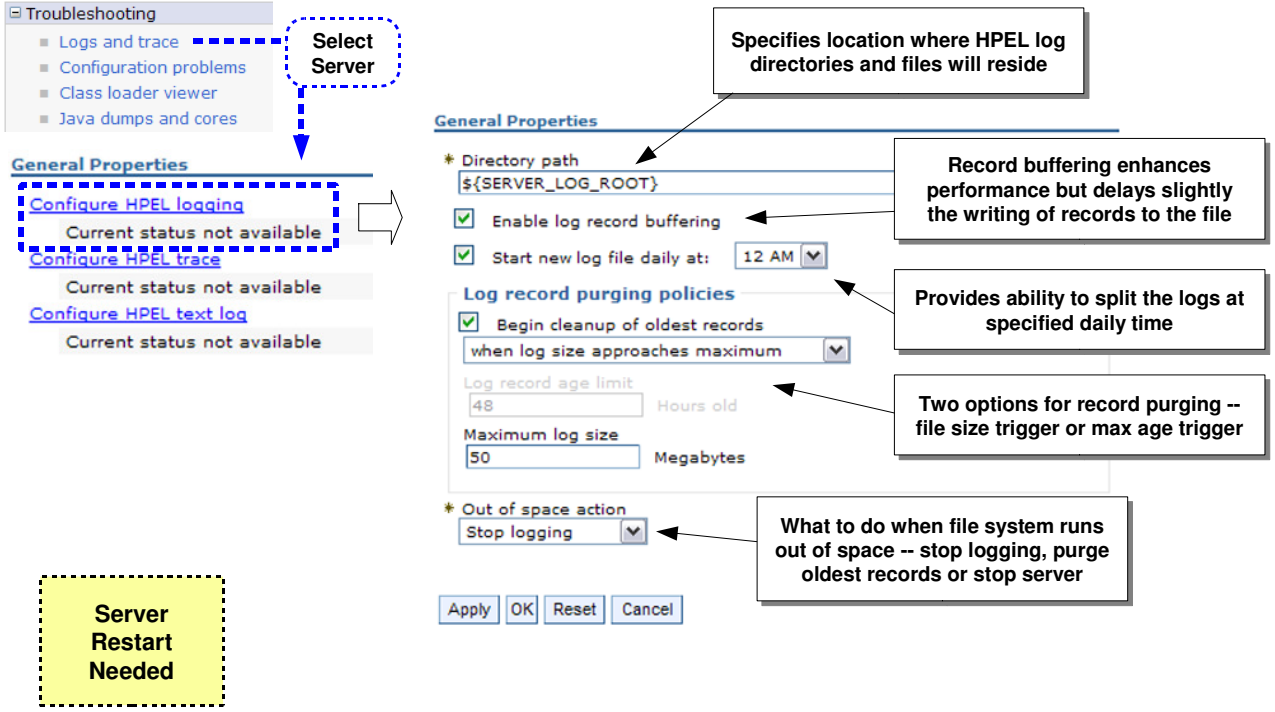
- First, select the log from which you wish to display records.
- Next, specify a "Message contents" filter with `*Application started*` as the string. The preceding asterisk is needed because the "Application started" records have a message ID to start the record. The trailing asterisk provides a wildcard for all that may follow. Note that the matching is case sensitive.
- The click on "Apply."
- Down in the record display section you'll see just those records that match your filtering criteria.

In this example a very easy check of all started applications in the server is made possible.

This is just one example. You can imagine many other filtering possbilities.

Configuring HPEL Logging for a Server

Process is relatively easy with a great deal of configurable options ...



Configuring HPEL tracing ...

Let's now look at how HPEL is configured. This is done by navigating to the "Troubleshooting" section of the navigation tree, selecting on "Logs and trace" and then select the server you wish to configure for HPEL.

You'll then be presented with three options for configuration of HPEL -- logging, trace and text log. Recall the earlier picture that showed the sources of output. Java logs and Java traces were two separate sources. You may configure them separately here as well. The text log is a human-readable log that's written concurrent with the binary logs. It's provided as a convenience for development and test. But as it involves additional overhead to write the less efficient text log concurrently it's not generally recommended for production. For WAS z/OS it's of limited value anyway ... it only writes the CR's text log, but not the SRs.

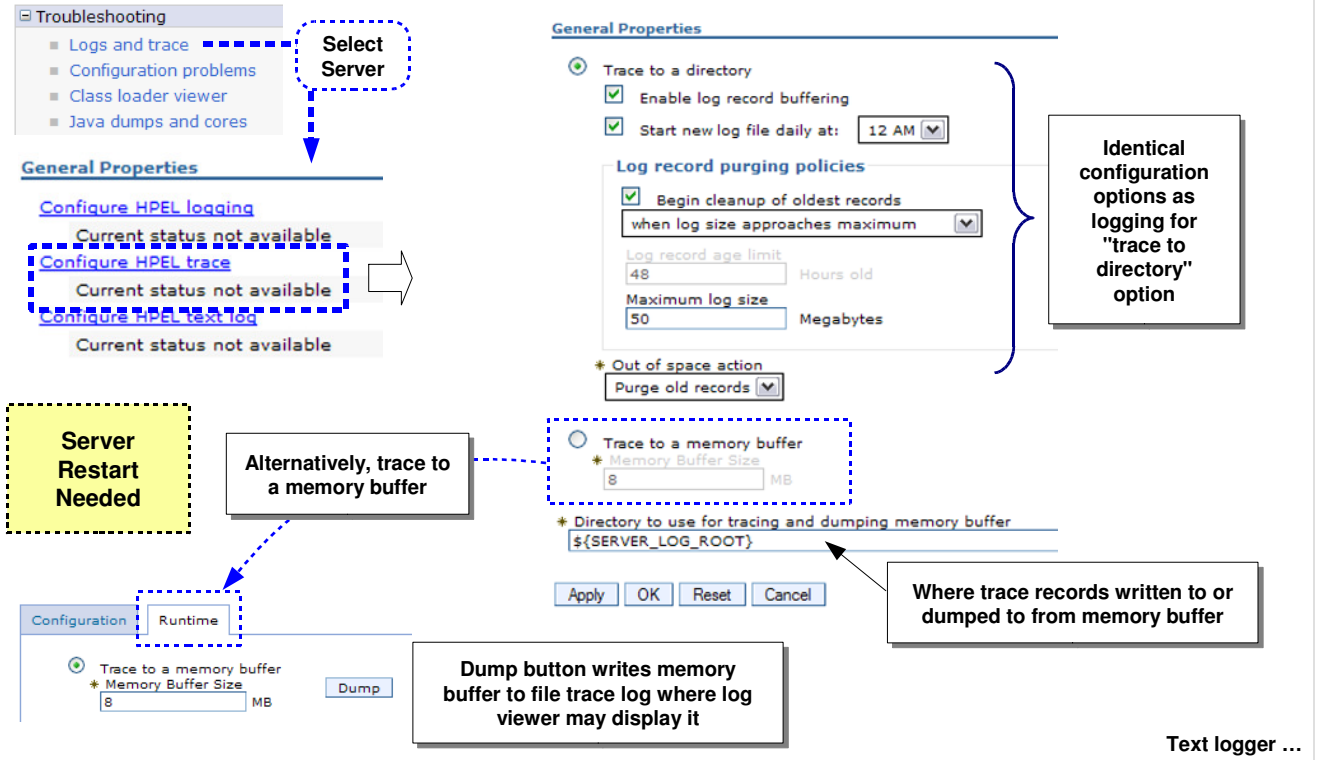
On this chart we're showing the configuration of logging sources. You're then offered several configuration options for this logging.

- Where you want the log to go. The default is shown on the chart but you can point this off to any location you wish. If you anticipate very large logs you may wish to point this to a location outside the configuration file system where you have a separate large file system mounted.
- Log buffering is a performance feature ... it allows WAS to write log entries to a memory buffer first, then write them out to the binary file.
- The log file is closed once a day and a new one opened. You get to specify when that close takes place.
- The final configuration elements relate to space management. These log files will not simply grow forever and ever. There's a maximum size setting and specifications for what to do when the limit is approached and reached. You have the option to clean old records using two triggers -- when the file size maximum is approached or when the age of the record exceeds a trigger (and remember, it's no more than 24 hours since the file is closed and a new one opened every 24 hours). And you get to set what happens if the log size maximum is reached ... stop logging, keep logging but make room by purging oldest, or stop the server.

Click "Apply" and then stop and restart the server to pick up the change.

Configuring HPEL Tracing for a Server

Process is relatively easy with a great deal of configurable options ...



HPEL *tracing* is in many ways similar to *logging*, but each comes from a different source. The configuration of tracing is in many ways similar to logging, but with one key difference -- with tracing you have the option of writing the trace records to a memory buffer rather than to a file.

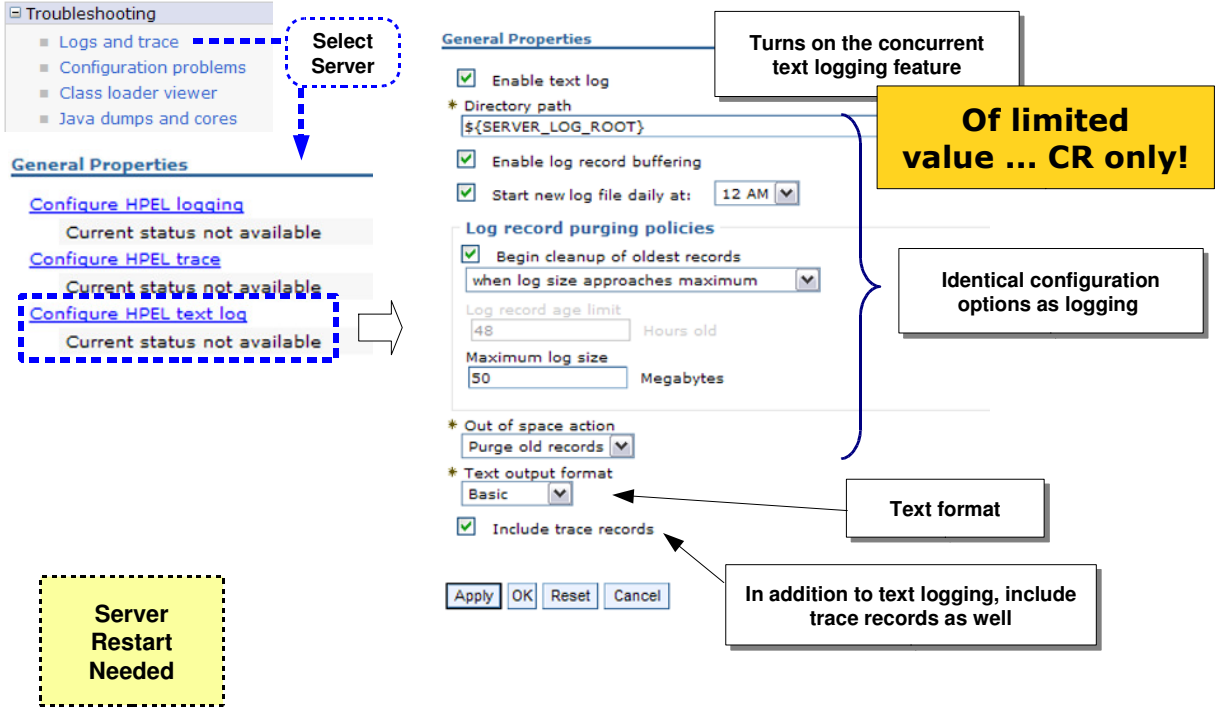
- *Trace to a directory* -- with this radio button selected you may then configure write-to-file in pretty much the same way you just saw for logging. You may specify whether temporary buffering is permitted, when to close to the file and open a new one, and how and when to purge records if space limits are reached.
- *Trace to a memory buffer* -- with this radio button selected the trace records are written to a memory buffer. This is different from the "Enable log record buffering" under "Trace to a directory" ... that is a temporary buffer before writing to file. The "Trace to a memory buffer" implies trace records written to a memory buffer and dumped to a file only by human action. The memory buffer is of a size you specify. If that buffer fills up, the oldest records are purged as new records are written.

  The "Dump" button is under the "Runtime" tab which is visible only when the server is running. What that button does is write the memory buffer contents at that moment out to the normal HPEL binary trace log file. Once the trace records are written to the directory file, then the Admin Console log viewer or the logViewer.sh utility may be used to filter and view the records.

  The memory buffer tracing is a performance feature ... tracing can be a very intensive activity and writing to a memory buffer is a faster operation than writing to a file. If you make the memory buffer large enough you should be able to capture the traces for your desired event. Then you dump those records to the directory file and review.

Configuring Optional HPEL Text Logging for a Server

Process is relatively easy with a great deal of configurable options ...



Command line logViewer.sh ...

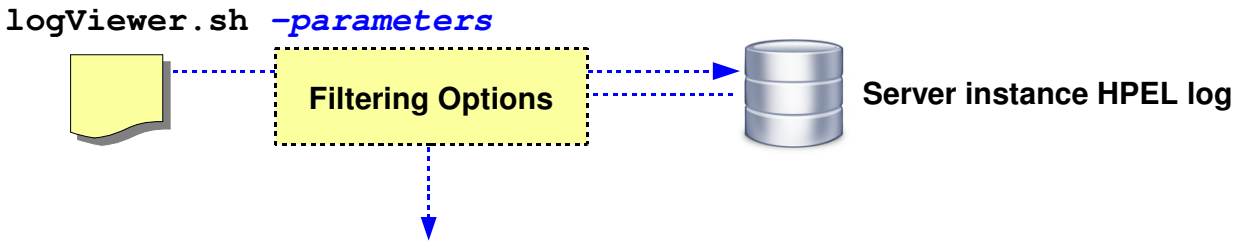
The text logger is a feature designed to provide testers and developers a human-readable format of the HPEL log concurrent with the binary log writing. When this function is enabled HPEL will format the binary records into text records and write them a file. There any text viewing tool will work.

For WAS z/OS this is of limited value due to the way this text logging feature is implemented. It writes the text log for the controller region but not the servant regions.

There is overhead associated with this function and since it's of limited value on WAS z/OS the recommendation is to *not* enable this. On distributed servers where the server is a single JVM process it works as expected, and there the value to testers and developers as a readily accessible text log might prove to be of a little more value.

The `logViewer.sh` Utility

A command-line utility to extract and view information from the binary HPEL logs. It has the same capabilities as the Admin Console log viewer:



- **Extract to a text-readable file**
- **Extract to a separate HPEL log**
- **Continuously monitor new output**

Let's take a tour of the parameters and options of this shell script utility. In lab you'll get a chance to use it.

Simple example ...

In addition to the Admin Console log viewer facility, WAS V8 provides a command line tool to read the binary HPEL log and do one of three things: extract to a text-readable file; extract to a separate HPEL log for more refined HPEL analysis, or monitor the HPEL log and provide text-readable output as updates are seen in the HPEL log.

This command line tool is provided as a shell script which takes various parameters which control the filtering options and the output options. Let's take a look at a simple example, then tour the parameter list in more detail.

The logViewer.sh Utility - Simple Starting Example

Here's a starting example of using logViewer.sh ... first list the server instances, then use the server instance ID to extract a human-readable output file:

Reminder of default location of HPEL logs *for a given server:*

```
/wasv85config/z9cell/z9nodea/AppServer/profiles/default/logs/z9sr01a
```

... and the location of the logViewer.sh shell script:

```
/wasv85config/z9cell/z9nodea/AppServer/profiles/default/bin/logViewer.sh
```

Determines the HPEL server instance logs that are present ***(all on one line)***

```
./logViewer.sh
  -repositoryDir /wasv85config/z9cell/z9nodea/AppServer/profiles/default/logs/z9sr01a
  -listInstances
```

```
Instance ID                                              Start Date
1321728440474                                            11/19/11 13:47:20.474 EST
1321728440474/0000012000000040-Z9SR01AS_STC00730         11/19/11 13:47:27.081 EST
1321728440474/0000010800000046-Z9SR01AS_STC00731         11/19/11 13:47:40.617 EST
```

Extract the HPEL log to a text-readable file ... view or download ***(all on one line)***

```
./logViewer.sh
  -repositoryDir /wasv85config/z9cell/z9nodea/AppServer/profiles/default/logs/z9sr01a
  -instance 1321728440474/0000012000000040-Z9SR01AS_STC00730
  -outLog /tmp/hpelout.txt
```

This file has the same format on z/OS as on other platforms. In EBCDIC, so download using "ascii"

Syntax ...

The logViewer.sh shell script utility is provided in the same location as all the other shell scripts -- in the /profiles/default/bin directory. A key piece of information it will need is the location of the directory in which the HPEL log you wish to view is located. Remember that HPEL logs are a server-by-server thing. By default they will be in the /profiles/default/logs directory, under the server name configured for HPEL.

With WAS z/OS there are multiple server "instances" for any given application server ... a controller region and at least one servant region and possible multpile servant regions. The logViewer.sh utility will need to know which instance you're interested in as well. The -listInstances option will tell you what instances it sees in the server log directory you specify. The chart shows an example output for a server with one controller and two servant regions. The date and time the server was started is also shown.

Suppose you wish to view the output in the servant instances highlighted in blue in the chart. Now that you have the log directory and instance information, you may issue a command such as illustrated above, which will read all records from the HPEL binary log and write a text-readable log file out to the named directory.

This is just one example ... and a very simple example. Do you recall all the filtering options the Admin Console log viewer had? Well, logViewer.sh has them as well. We see that level of detail surface as we look at the syntax of the options for logViewer.sh next.

Details of `logViewer.sh` Parameters

Here is the complete list of options and a brief description of each. Notice how the parameters mirror the Admin Console log viewer options:

`logViewer.sh`

`-repositoryDir <directory name>` *Location of HPEL repository to read from*

`-outLog <file_name>` *Path and file name of the output file*

`-startDate <date_time>` *Extract only records after this date and time (help provides syntax options)*

`-stopDate <date_time>` *Extract only records before this date and time*

`-level` `FINEST | FINER | FINE | DETAIL | CONFIG | INFO | AUDIT | WARNING | SEVERE | FATAL` *Level to extract*

`-minLevel` `FINEST | FINER | FINE | DETAIL | CONFIG | INFO | AUDIT | WARNING | SEVERE | FATAL` *Start range to extract*

`-maxLevel` `FINEST | FINER | FINE | DETAIL | CONFIG | INFO | AUDIT | WARNING | SEVERE | FATAL` *End range to extract*

`-format <basic | advanced | CBE-1.0.1>` *Format of output file*

`-monitor [interval]` *Continuously monitor and update output file*

`-includeLoggers <logger_names>` *Include loggers by logger class name*

`-excludeLoggers <logger_names>` *Exclude loggers by logger class name*

`-thread <thread_id>` *Extract only for specified thread*

`-extractToNewRepository <directory_name>` *Option to create a sub-repository based on extract rules*

`-listInstances` *List the server process instances found in the repository*

`-instance <instanceid>` *Extract only named process instance*

`-latestInstance` *Extract only the most recent server process instance*

`-message <message>` *Extract records that match message mask ... asterisk and question mark wildcards allowed*

APAR PM74923 ...

This chart lists all the options that apply to `logViewer.sh` and a very brief description of what they do. This information was extracted by issuing the `./logViewer.sh -help` command.

If you scan the options above you see they map to the same filtering options we saw earlier with the Admin Console log viewer.

Newest Method of Managing Output

New function introduced via an APAR to V7, V8 or V8.5:

- **Introduced by APAR PM74923**

Version	Maintenance
V7	7.0.0.29
V8	8.0.0.6
V8.5	8.5.0.2

- **Fully describe in techdoc:**

Techdocs Library > White papers >

Implementing the Output APAR (PM74923) enhancements in WebSphere Application Server on z/OS

`http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102267`

Overview ...

In addition to HPEL, another method of producing non-JES output has been introduced via an APAR. The chart above shows the maintenance levels for V7, V8 and V8.5 in which the APAR is delivered. In addition, the Techdoc shown on the chart has a complete description of function as well as a description of how to use an Apache webserver to provide controlled access to the log files produced.

The Method Introduced By the APAR

Produces log output to a file system location you specify:

- **File based.**
- **WebSphere ensures that there will be no naming conflicts**
- **Works for all components (Daemon, Dmgr, Nodeagents, Servers (both controllers, adjuncts, and servants)**
- **File switching is simple using a z/OS MODIFY command**
- **No need for users to access the filesystem**
- **Access to the output can be uncontrolled or controlled...**
  - **At the cell level**
  - **At the node**
  - **Server by server**
  - **Via normal security on the filesystem ... or ...**
  - **Via any security system that the HTTP server supports (SAF, LDAP, etc.)**

Variables ...

The main message of this chart is that the output produced by this new function is a set of files per server in the Unix Systems Services (USS) file system. This new function works for all server types, and you may use this function for one, several, or all the servers in your cell.

Note: that is, if you'd like some servers to go to JES, with other servers going to this new file-based output, that is quite possible. It's also possible to designate some servers as JES, some as HPEL and still others as this new function.

The file names created are based on the server name, STC number and data and time. That means WAS is responsible for unique names.

How does it work? What do I do to set it up?

It's really fairly simple ... a few WebSphere variables:

In WebSphere:

Add variables at appropriate scopes.

- **They will be inherited by lower levels...**

  Simplest setup is to just add them at the cell level and let all components write to the same path

  The same variable at a lower level will take precedence

- **Variable names:**

  `DAEMON_redirect_server_output_dir` (for the Daemon)

  `redirect_server_output_dir` (for everything else)

  Value is simply the path name where you wish the output to be written

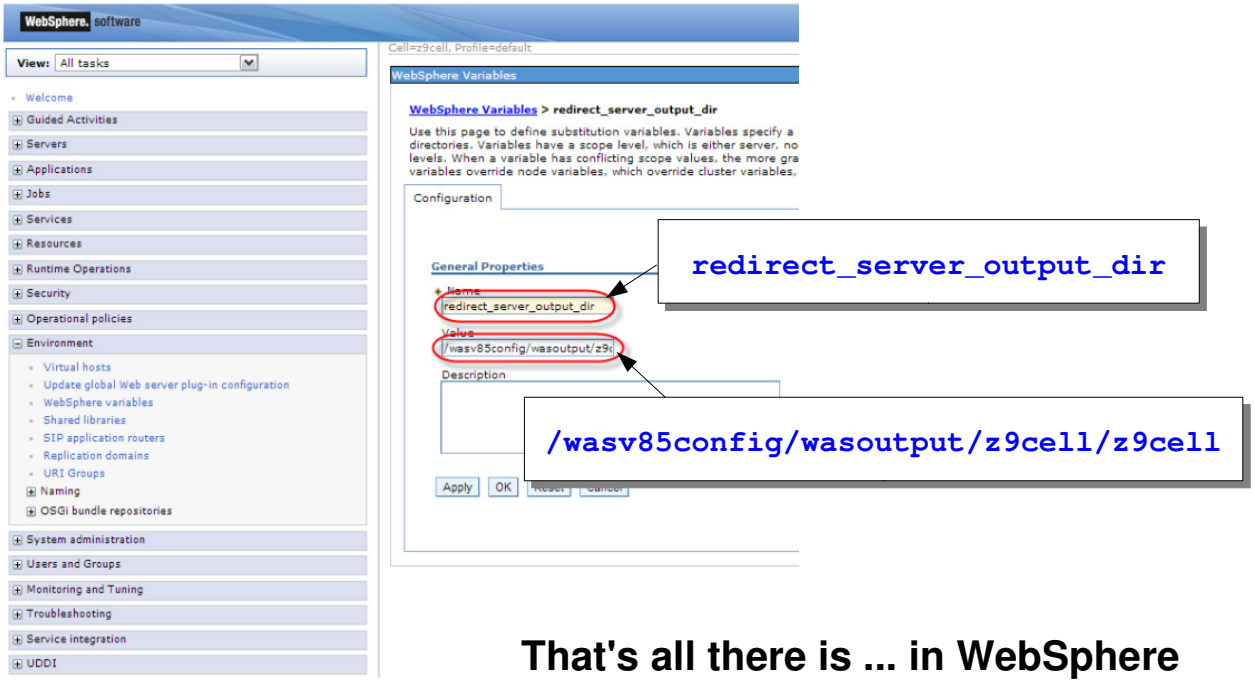
  Example: `/wasv85config/wasoutput/z9cell/z9cell`

Admin Console ...

Two variables control the function. The key is what scope you set the variables to.

Setup in WebSphere

This is standard WebSphere environment variable setup:



That's all there is ... in WebSphere

Server restart ...

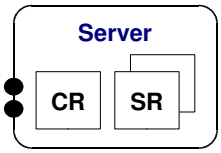
This chart is simply showing the variable being specified in the Admin Console.

Server Restart and the Result ...

In WebSphere all that is left is to restart the components you believe that you have modified, and the output will be redirected to the path you've specified.

`DAEMON_redirect_server_output_dir` **(for the Daemon)**

`redirect_server_output_dir` **(for everything else)**




```
:/shared/wasoutput/z9cell/z9cell
-> ls
Z9CELL.Z9DMNODE.WG31.Z9DEMN.STC12090.DAEMON.130701.174549.SYSOUT.txt
Z9CELL.Z9DMNODE.WG31.Z9DEMN.STC12090.DAEMON.130701.174549.SYSPRINT.txt
Z9CELL.Z9DMNODE.Z9DMGR.Z9DMGR.STC12089.CTL.130701.134548.SYSOUT.txt
Z9CELL.Z9DMNODE.Z9DMGR.Z9DMGR.STC12089.CTL.130701.134548.SYSPRINT.txt
Z9CELL.Z9DMNODE.Z9DMGR.Z9DMGRS.STC12091.SR.130701.134606.SYSOUT.txt
Z9CELL.Z9DMNODE.Z9DMGR.Z9DMGRS.STC12091.SR.130701.134606.SYSPRINT.txt
Z9CELL.Z9NODEA.Z9AGNTA.Z9AGNTA.STC12092.CTL.130701.134710.SYSOUT.txt
Z9CELL.Z9NODEA.Z9AGNTA.Z9AGNTA.STC12092.CTL.130701.134710.SYSPRINT.txt
```

Output for the Daemon

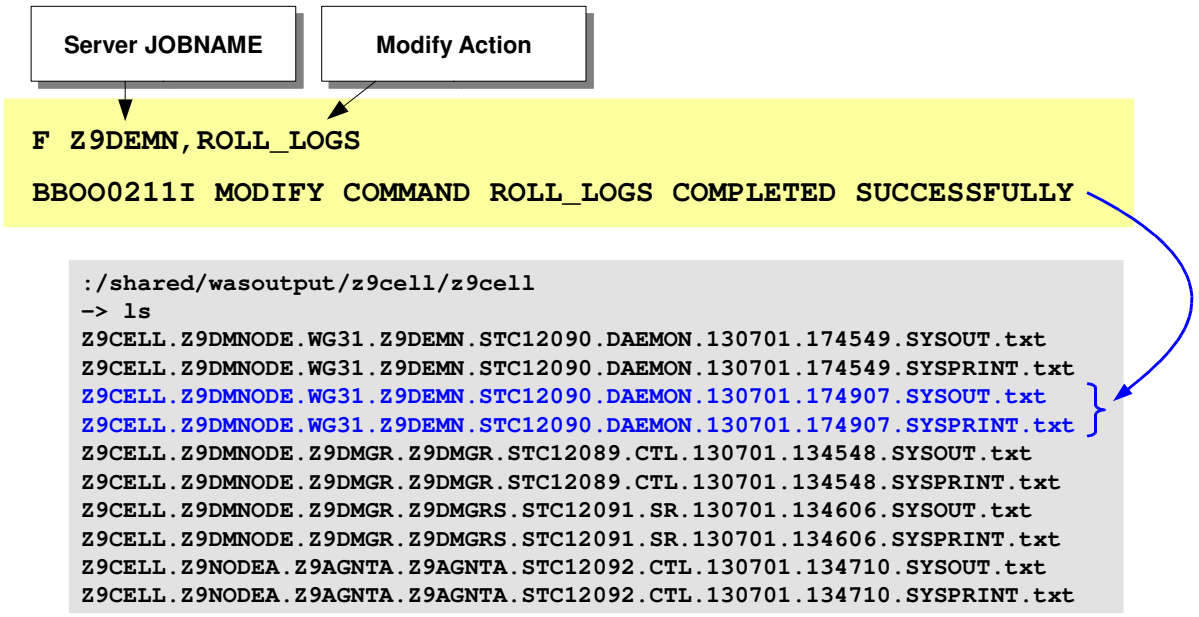
Output for other servers in the cell

Rolling log files ...

With a restart of the server to pick up the change you will begin seeing the output piped to the location specified.

Rolling Log Files

If you wish to switch to a new file, from any z/OS Console:



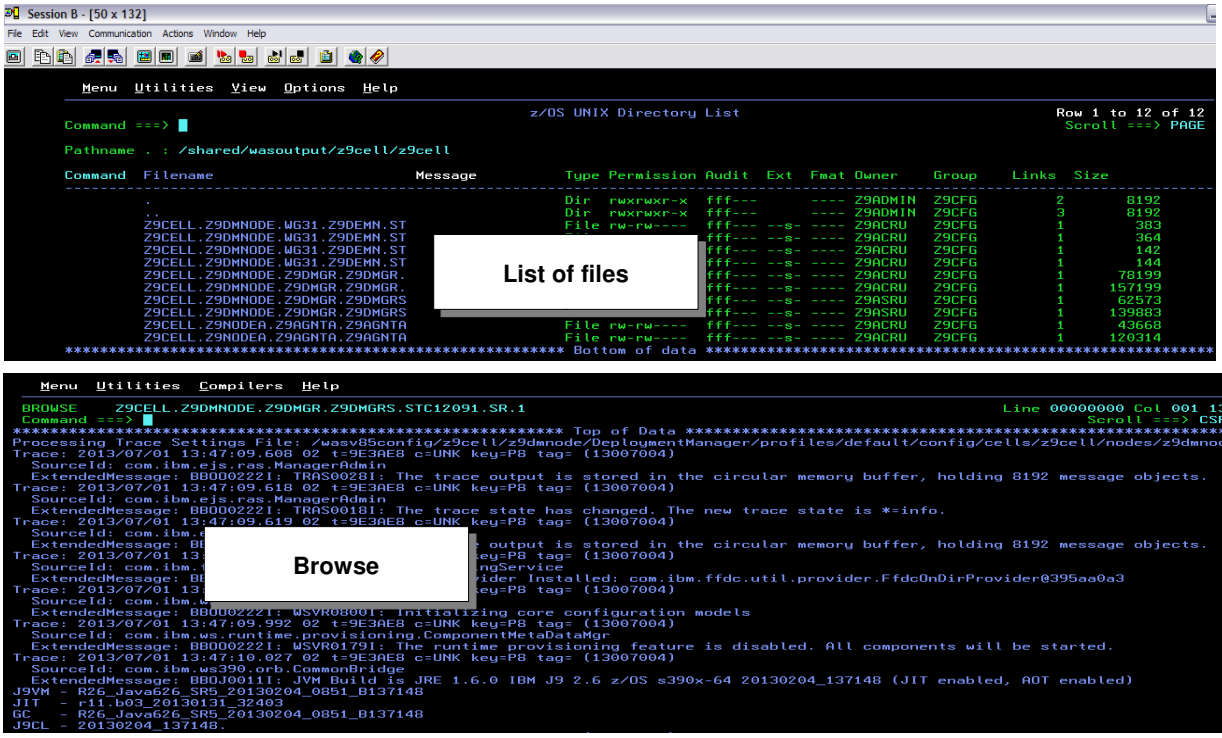
At this point, if all you want is ISPF browse or telnet access, you are done...

Result …

One of the shortcomings of the old method of routing log output to the file system was no good way to stop writing to one file and begin writing another. That shortcoming has been addressed with this new function by way of a new MODIFY command that will roll the log files to a files for a specified server.

The Result

Then browse any file name you choose:



The result is a set of text files in the file system that you may then browse.

End of Unit