WebSphere Application Server

WebSphere for z/OS Version 5.1

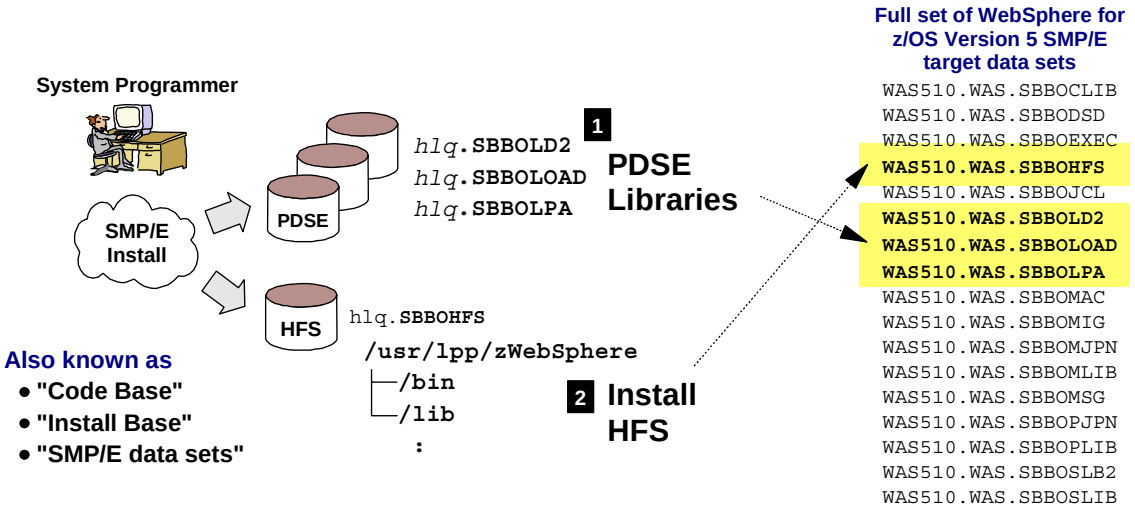
Test, Production, Maintenance

IBM Americas Advanced Technical Support -- Washington Systems Center
Gaithersburg, MD, USA

(This page intentionally left blank)

Two "Install Environments"

When WebSphere Application Server for z/OS V5.1 is installed, SMP/E creates two "install environments":



Key Points:

- **WebSphere comes in two pieces -- typical MVS libraries and an HFS file system**
- **The "Install HFS" is different from the "Config HFS" you created for your cell**

The fundamental issue we'll address is the relationship between your configuration and these "install environments"

When WebSphere Application Server for z/OS Version 5.1 is installed, you get a bunch of data sets. The list is shown n the chart above. For the purposes of this discussion, there are two "environments" we're very much interested in:

- The PDSE libraries -- these are where the typical MVS program modules reside. There are three data sets of interest here: `SBBOLD2`, `SBBOLOAD` and `SBBOLPA`.
- The "Install HFS" -- WebSphere has a significant portion of its code delivered in the form of an HFS file system. That file system is packaged in an MVS data set called `SBBOHFS`. That data set is mounted at a mount point to form the "SMP/E Home Directory" for the WebSphere installation.

Together, these form what some call the "Code Base," or perhaps the "Install Base" or maybe even the "SMP/E data sets." Whatever term you apply to the target data sets for WebSphere, please understand two key points:

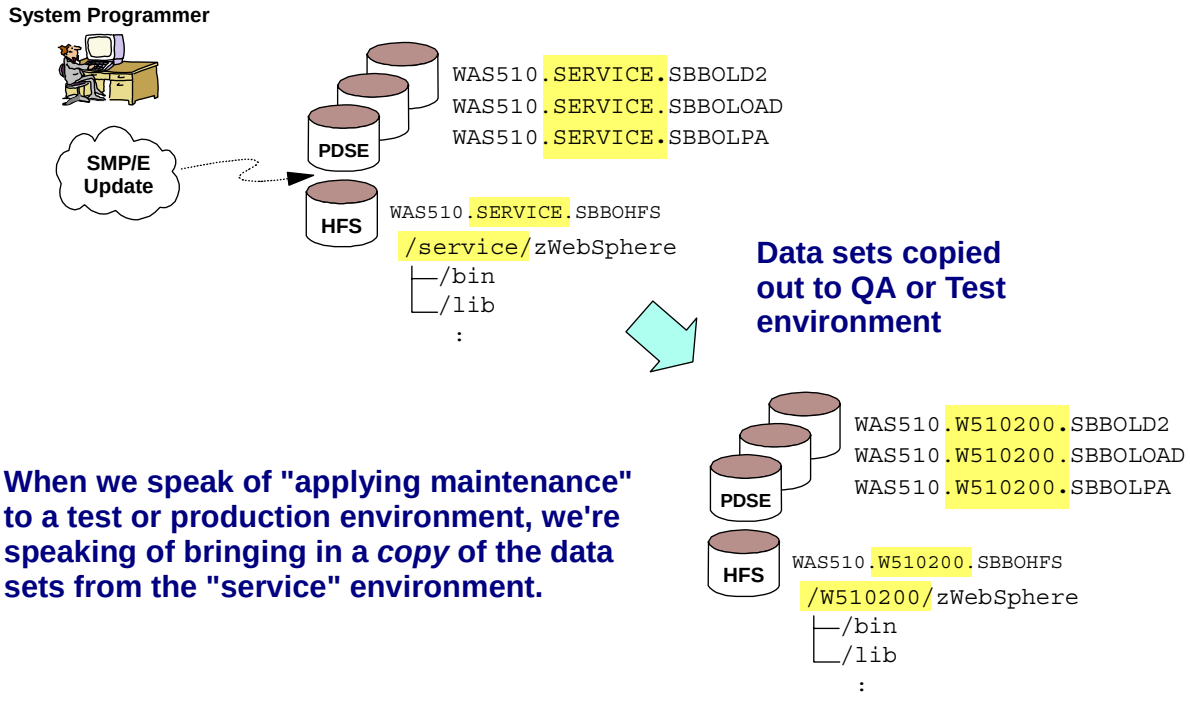
- WebSphere is not just a set of MVS module libraries. A significant amount of the WebSphere code is actually held in the "Install HFS" piece of the product.
- Your "Configuration HFS" -- the one you created when you ran your customized jobs -- is different from this "Install HFS." Understanding that the two are different is key.

This topic of "Test, Production and Maintenance" is all about the relationship between your configuration (or plural, configurations) and the install base made up of MVS module libraries and an install HFS. That relationship determines the degree of isolation you can achieve and hence the ability to form separate test and production environments. That relationship also determines the degree of isolation within a configuration, hence the ability to "roll" maintenance non-disruptively.

Version Date: September 22, 2004

Applying Maintenance to Service Env.

Maintenance is typically applied against "service" environment, then copied out to the test or production environment:



When we speak of "applying maintenance" to a test or production environment, we're speaking of bringing in a *copy* of the data sets from the "service" environment.

This chart is meant to simply reinforce something you probably already know about -- that is, in most shops new maintenance is SMP/E applied against a separate "service" environment first, then copies of the updated target data sets are then moved to the Test or QA environments.

Throughout this presentation we'll speak of "applying maintenance" to a configuration. What we mean by that is bringing in a copy of the data sets from the "service" environment.

Multiple Versions Possible

There's nothing about WebSphere for z/OS Version 5.1 that prevents you from having multiple version levels on the same MVS image:

Production Version	New Maintenance Level Being Tested
WAS510.WAS.SBBOCLIB	WAS510.W510200.SBBOCLIB
WAS510.WAS.SBBODSD	WAS510.W510200.SBBODSD
WAS510.WAS.SBBOEXEC	WAS510.W510200.SBBOEXEC
WAS510.WAS.SBBOHFS	WAS510.W510200.SBBOHFS
WAS510.WAS.SBBOJCL	WAS510.W510200.SBBOJCL
WAS510.WAS.SBBOLD2	WAS510.W510200.SBBOLD2
WAS510.WAS.SBBOLOAD	WAS510.W510200.SBBOLOAD
WAS510.WAS.SBBOLPA	WAS510.W510200.SBBOLPA
WAS510.WAS.SBBOMAC	WAS510.W510200.SBBOMAC
WAS510.WAS.SBBOMIG	WAS510.W510200.SBBOMIG
WAS510.WAS.SBBOMJPN	WAS510.W510200.SBBOMJPN
WAS510.WAS.SBBOMLIB	WAS510.W510200.SBBOMLIB
WAS510.WAS.SBBOMSG	WAS510.W510200.SBBOMSG
WAS510.WAS.SBBOPJPN	WAS510.W510200.SBBOPJPN
WAS510.WAS.SBBOPLIB	WAS510.W510200.SBBOPLIB
WAS510.WAS.SBBOSLB2	WAS510.W510200.SBBOSLB2
WAS510.WAS.SBBOSLIB	WAS510.W510200.SBBOSLIB

Data sets have different qualifiers

Not quite as simple as that ... do have considerations related to:

- **Which copy -- if any -- resides in LPA/LNKLST**
- **How to "toggle" a configuration from one code base to another**
- **How to non-disruptively introduce new level into a production environment**

That's what this presentation is all about ...

■

It's quite possible to have multiple copies of WebSphere for z/OS on a given MVS image ... it entails having the data sets possess different high-level qualifiers. So when new maintenance is available and that maintenance is applied against the "service" environment (see the previous chart), the copy (or "clone") of that environment can be brought into the picture with a different HLQ, perhaps indicating the maintenance level contained within the copy.

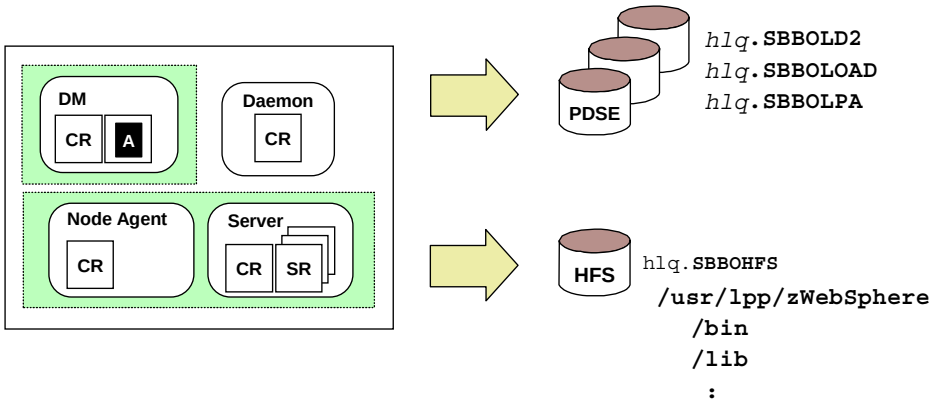
What we now have to do is focus on the realities of this ... how, for instance, you define which copy resides in LPA/LNKLST, how to change a configuration from one code base to another, and how to move maintenance into a production environment so that at least part of the configuration is always available. That's what this presentation is all about.

Basics: How Config Accesses Code

Load libraries accessed in one of two ways:

- **LPA/LNKLST**
- **STEPLIB**

Fairly familiar stuff, but there are a few twists. We'll spend some time understanding this first.



HFS files are accessed via symbolic links in the configuration HFS

Some new things here ... we'll focus quite a bit of time on this area of the puzzle.

A WebSphere for z/OS cell -- a "configuration" -- accesses the two SMP/E environments in different ways:

- The PDSE libraries are accessed in the traditional way -- either via LPA/LNKLST or STEPLIB statements. This is a standard operating procedure ... but there's something you may not know about. That something is the location of STEPLIB statements other than in JCL. We'll show you where those are located in a bit.
- The SMP/E HFS is accessed with symbolic links in the configuration HFS. This is something you may not be familiar with. These symbolic links are the key to understanding the strategy behind configuration isolation.

We'll look at the STEPLIB issue first, then turn our attention to symbolic links.

STEPLIB or LPA/LNKLST?

Do you remember when you first built your node with the ISPF panels? Whether STEPLIB is present or not is determined in "System Locations 1 of 2" panel:

```
Full Names of Data Sets

  PROCLIB:  SYS1.PROCLIB
  PARMLIB:  SYS1.PARMLIB
  SYSEXEC:  SYSS.WSC.SYSEXEC              In LNKLST or
                                          LPA (Y or N)?
  SCEERUN.:  CEE.SCEERUN                      Y
  SCEERUN2:  CEE.SCEERUN2                     Y
  SBBOLOAD:  WAS510.WAS.SBBOLOAD              Y
  SBBOLD2.:  WAS510.WAS.SBBOLD2               Y
  SBBOMIG.:  WAS510.WAS.SBBOMIG               Y
  SBBOLPA.:  WAS510.WAS.SBBOLPA               Y
  SGSKLOAD:  SYS1.GSK.SGSKLOAD                Y
```

Question asked is whether libraries in LPA/LNKLST ...

Y -- *no* STEPLIB built

N -- STEPLIB created

Wish to change a configuration to a new level of code?

Libraries in LPA/LNKLST?	***Refresh* contents of LPA/LNKLST**
STEPLIBs already present?	***Change* STEPLIB statements**
STEPLIBs not present but you want them?	***Add* STEPLIB statements**

Where can STEPLIB statements be found?

■

Your configuration accesses the PDSE libraries in one of two ways -- by relying on the modules being in LPA/LNKLST or by using STEPLIB statements. Which of the two your configuration makes use of depends on how you answered the questions on the "System Locations 1 of 2" panel when creating your configuration. The "In LNKLST or LPA?" question for the data sets determines whether STEPLIB statements are built. Answer "Y" and no STEPLIBs are built -- you're telling the configuration that the modules are in LPA/LNKLST so no STEPLIBs are needed. But answer "N" and STEPLIBs *are* built -- you're indicating that the modules are not in LPA/LNKLST so STEPLIBs are needed.

So what this means is that if you're looking to change the SMP/E libraries a configuration uses, it will depend on how your configuration accesses those libraries:

- If your configuration relies on LPA/LNKLST, then what you need to do is refresh the modules in LPA/LNKLST.
- If your configuration uses STEPLIB statements, then it's just a matter of changing the STEPLIB pointers so they point to a different set of data sets.
- If your configuration relies on LPA/LNKLST, but you would prefer it to use STEPLIB, then it means *adding* STEPLIB statements. And this is the where we need to tell you about the two places where STEPLIB statements are found.

STEPLIB -- *Two* Places

JCL procedures are in two parts ... "Z member" is where STEPLIB would be:

G5ACR -- the main body of the JCL

"Include"

G5ACRZ -- the "Z member" associated with the proc.

```
//*
//* Output DDs
//*
//CEEDUMP   DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSOUT    DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSPRINT  DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//*
//*Steplib Setup
//*
//STEPLIB  DD DISP=SHR,DSN=WAS510.WAS.SBBOLD2
//         DD DISP=SHR,DSN=WAS510.WAS.SBBOLOAD
//         DD DISP=SHR,DSN=WAS510.WAS.SBBOLPA
```

***All* procs have two-part JCL:**

- **Application server controllers**
- **Application server servants**
- **Deployment Manager controller**
- **Deployment Manager servant**
- **Node Agent controllers**
- **Daemon**

STEPLIB may also be in shell script `setupCmdLine.sh` found in `/bin` directory

/bin

setupCmdLine.sh

This establishes STEPLIB for all the shell scripts that are part of WAS.

```
 :
#
STEPLIB='WAS510.WAS.SBBOLD2':$STEPLIB
STEPLIB='WAS510.WAS.SBBOLOAD':$STEPLIB
STEPLIB='WAS510.WAS.SBBOLPA':$STEPLIB
export STEPLIB
 :
```

Careful! Multiple copies of this file ...

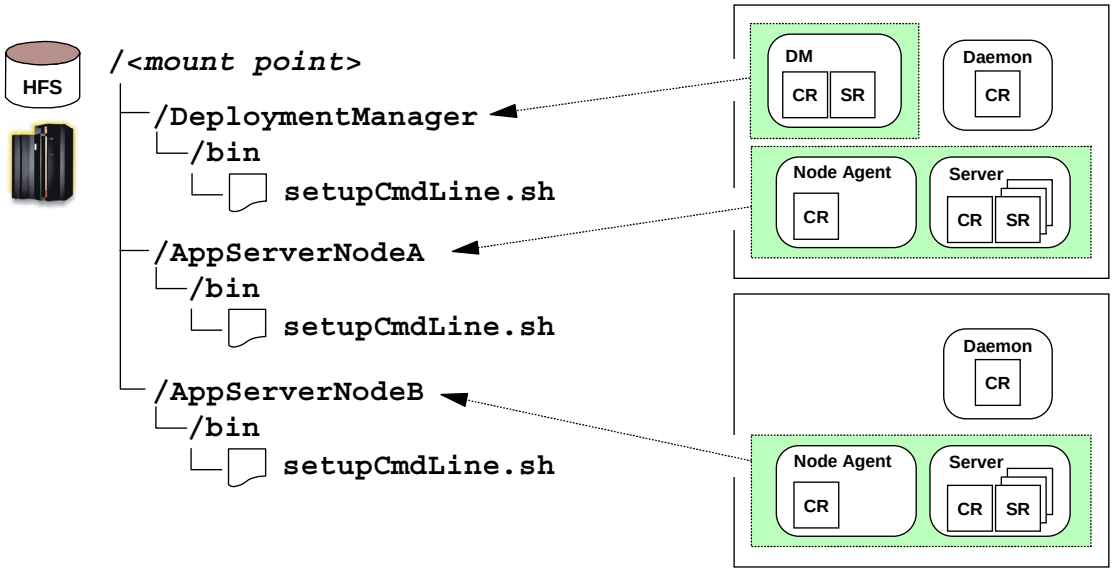
STEPLIB statements -- if they exist in your configuration -- are found in two places:

- The JCL procedures for the servers. The start procedures are delivered in two parts -- a "main body" that has the main program reference, and a "Z member" that is "included" into the main body and contains the SYSOUT and SYSPRINT statements as well as STEPLIB. But the STEPLIB will only be there if you answered "N" on the "System locations 1 of 2" panel.
- A shell script called `setupCmdLine.sh`. This shell script is used to establish the command line environment for the other shell scripts that are run as part of WebSphere's operations. If your configuration is pulling program modules from LPA/LNKLST, then you won't find STEPLIB in the `setupCmdLine.sh` file. But they'll be there if you answered "N" to the "In LPA/LNKLST?" question on "System Locations 1 of 2".

The normal conclusion to this is that if you wanted to add STEPLIB to your configuration, you'd add it to the JCL and to the `setupCmdLine.sh` file. That's true ... but be careful. There are multiple copies of that file.

Multiple Copies of `setupCmdLine.sh`

A multi-node configuration will have multiple copies of the `setupCmdLine.sh` shell script ... each node structure's `/bin` directory will contain a copy:

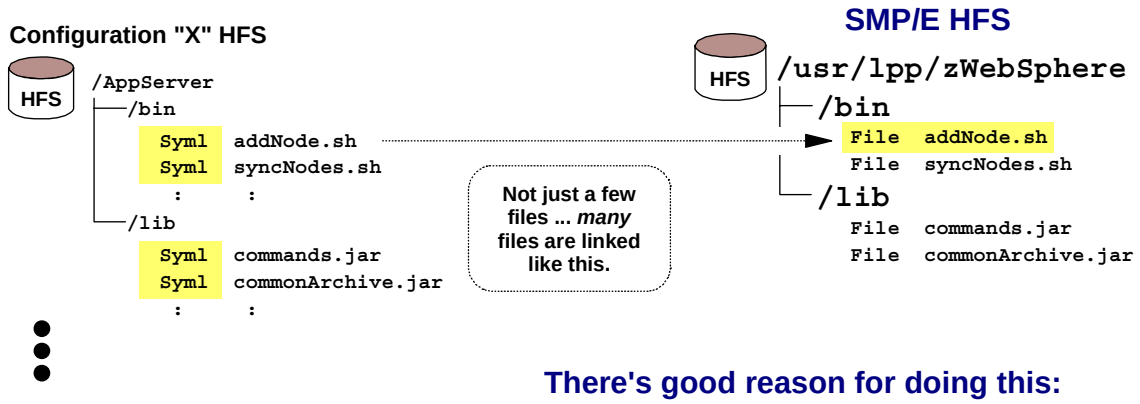


Key Message: any changes to STEPLIB -- either to add STEPLIB or to change STEPLIB -- involves changes to all "Z members" *and* all `setupCmdLine.sh` files.

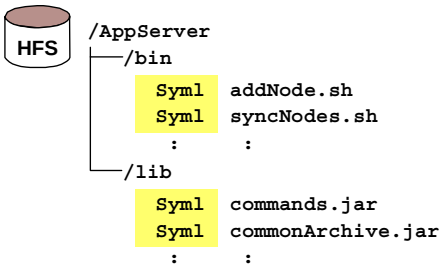
The `setupCmdLine.sh` file exists in the `/bin` directory of every node structure in your cell, including your Deployment Manager node. That means if you ever wanted to make a change to your STEPLIB arrangement -- either to add them or to modify them -- you need to change all the `setupCmdLine.sh` instances in your configuration, not just one of them. This would be in addition to any changes to your JCL.

Symlinks in the Configuration HFS

Many of the files in the Configuration HFS are really just symbolic links that point off to the actual file in the SMP/E (or "Install") HFS:



Configuration "Y" HFS



There's good reason for doing this:

- **Eliminates unnecessary duplication of the same file in lots of different configuration HFS**
- **Gives SMP/E one place to update files rather than lots of different places**

Where's the pointer to the SMP/E HFS mount point specified? In the ISPF Dialogs ...

Now we shift gears and talk about how the configuration HFS relates to the SMP/E HFS. This is done with symbolic links in the configuration HFS that related directly to actual files in the SMP/E HFS. For example, in the configuration HFS's `/bin` directory you'll find a symbolic link called `addNode.sh`. Over in the SMP/E HFS there's a *real* file called `addNode.sh`. The symbolic link in the configuration HFS points to the real file.

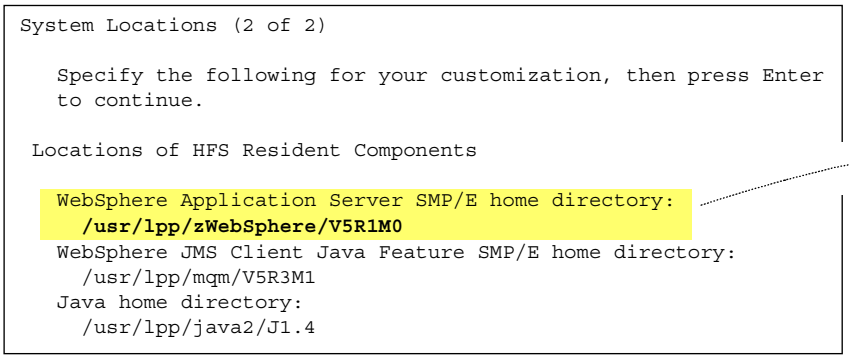
This is not just being clever for the sake of being clever. There's good reason for this design. It allows multiple configurations -- and remember, you may have any number of those -- to have symbolic links rather than the same file in all places. That avoids unnecessary duplication of files. The other advantage of this is allows a single copy of the real file in the SMP/E HFS to be updated by maintenance, and have that change apply to all the configuration's that link to it.

Note: There are lots of symbolic links in the `/bin` and `/lib` directories, not just a handful. So you can't think about changing the symbolic links there .... there's just too many of them.

The question now is where the pointer to the SMP/E HFS that's in the symbolic links came from? From a field in the ISPF customization dialogs. That's next.

Contents of the Symbolic Links

Contents of the symbolic links in the Configuration HFS are determined by value you supply for "WebSphere SMP/E home directory" in panels:



Whatever value you supply on panel is used to created all the symlinks in the node

ISHELL listing of a sample configuration HFS

```
   Type  Filename
_ Dir   .
_ Dir   ..
_ Syml  addNode.sh
_ Syml  admincons.pax
_ Syml  admincons.unpax.sh
_ Syml  applyPTF.sh
  :
```

```
Symbolic link contents:
/usr/lpp/zWebSphere/V5R1M0/bin/applyPTF.sh
```

Once set ... they're set. Can't change them.

This has a profound impact on how different configurations are isolated from one another, and impacts how "non-disruptive" maintenance can be achieved.

The symbolic links that are in your configuration HFS are populated with a pointer to the actual file in the SMP/E directory. The contents of the symbolic link are set when you specify the "WebSphere Application Server SMP/E home directory" field in the "System Locations 2 of 2" panel in the ISPF customization panels.

You can validate the contents of a symbolic link by going into ISHELL and issuing the command "a" (for attributes) next to the file. What you'll see is information on the "Symbolic link contents," which will be equal to the value you supplied in the panel.

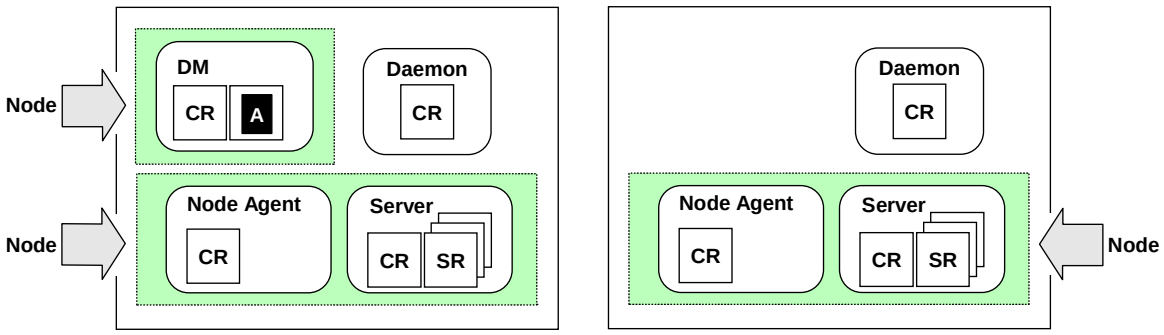
Note: Once the values are set in all the symbolic links for a node, they're set. Don't think about trying to go in and alter the symlinks to point to a different directory. There are too many of them and you'll make a mistake. Besides, there's an easier way to provide flexibility. That's coming up.

This all seems simple enough, but in fact this plays a major role in the way you can isolate one configuration from another and how you can configure a cell so maintenance can be "non-disruptively" rolled through it.

Granularity of Symlinks: Node Level

WebSphere configurations are built by constructing nodes and then federating them into larger and larger cells:

- **The symbolic links are created when the node is built**
- **Symbolic links are in the `/bin` and `/lib` directories ... *above* server level**
- **Therefore, symbolic links are no more granular than the node level**



Why is this important? Because it determines the granularity of isolation that can be achieve. And that leads directly into discussion of isolating test from production, and isolating node from node so maintenance can be "rolled" through a configuration.

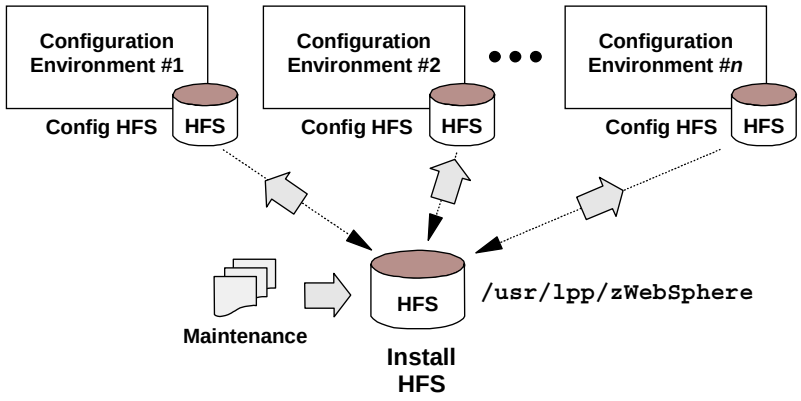
Recall that WebSphere for z/OS "network deployment" configurations are built by first constructing nodes, then federating the nodes into the Deployment Manager. It is during the running of the ISPF customization panels for each node that you specify the "WebSphere SMP/E home directory" value which sets the contents of the symbolic links. That means that the symbolic links are set node by node, and you can have each node have a different value if you wish.

But you can't make it any more granular than that. The `/bin` and `/lib` directories (where the symbolic links are located) are above the server level in the configuration HFS tree. Those directories relate to the node as a whole. So the symbolic links are granular down to the node level, but no lower.

This is a key point ... it means that the granularity of isolation for the purposes of rolling maintenance is node by node.

Issue with Common SMP/E Mount Point

If you have a number of different configuration environments all pointing to the same "WebSphere SMP/E Home Directory," maintenance will apply to all:

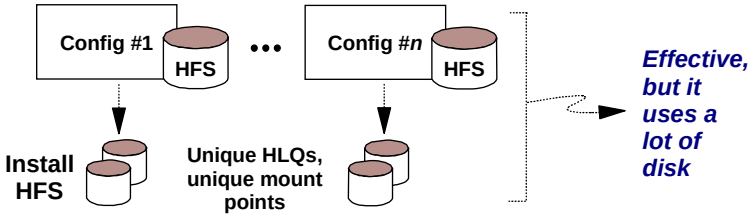


This may or may not be okay, depending on what you're trying to accomplish:

- **Multiple environments where you want the same code level? Okay.**
- **Test and Production environments? Not okay.**

Two ways to provide isolation between environments:

Provide separate install data sets for each:



Other method is to employ "intermediate symbolic links" ...

Consider a situation where several different configurations all point to the same mount point where the WebSphere "SMP/E HFS" is mounted. Then consider what happens when maintenance is applied to that code base. The maintenance will then apply to *all* the configurations that point to that HFS.

This may be exactly what you desire. Or it may not. If you have several different development or functional test configurations where you want them to be at the same level of code, this is ideal. Apply maintenance once and it applies to all of the configurations.

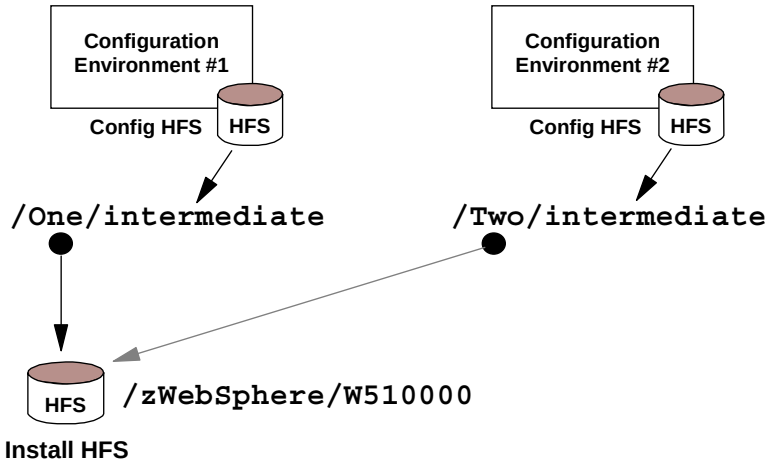
But in the case of test and production, this is exactly what you *don't* want. This would mean you don't have isolation between the test and production configurations, and when new maintenance is brought in, it applies to both test and production. There's no chance to validate the maintenance before moving it to production.

There are two ways to provide isolation:

1. One is to have completely separate copies of the WebSphere code base installed, each with different and unique high level qualifiers and mount points. That way one configuration could point to one copy of the code; the other configuration point to the second copy. That's an effective way to provide isolation, but it uses a lot of disk, particularly when there are many configurations you wish to have isolation.
2. The second method involves using what's known as "intermediate symbolic links." That's where you place another symbolic link between the SMP/E HFS mount point and the symbolic links in your configuration HFS. By doing so, you provide the *ability* to isolate the environments.

Intermediate Symbolic Links

Symbolic links may be "chained" ... this provides considerable flexibility:



Create the intermediate symbolic link ahead of time and then go through the panels as usual. But rather than pointing directly to the Install HFS, point to the intermediate symbolic link for that configuration.

What's so special about that? Watch ...

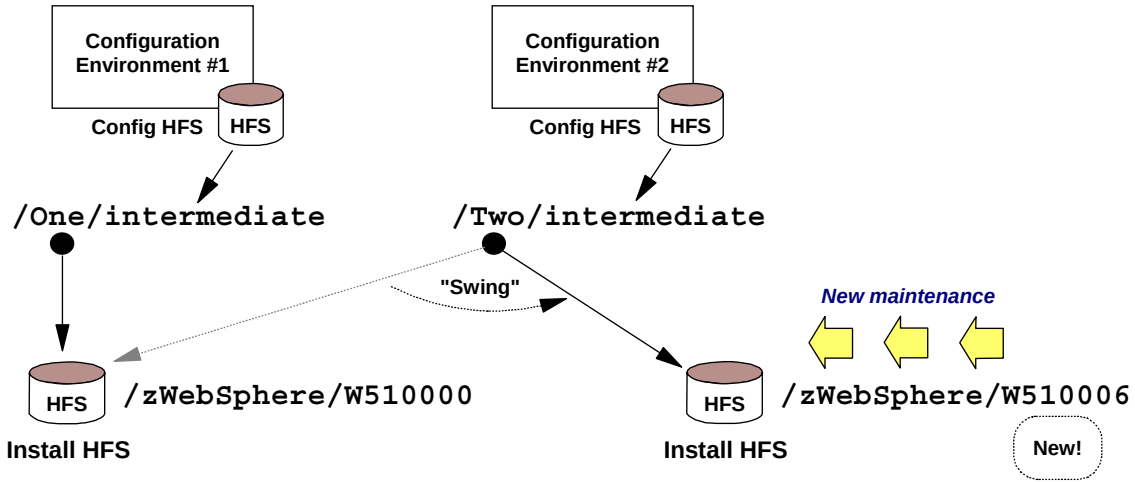
Now imagine two configurations, both pointing to the same "SMP/E Home Directory," but rather than pointing directly to the home directory there's a symbolic link between the two. The symbolic links have different names -- `/One/intermediate` and `/Two/intermediate`, for example -- but the *contents* of each is the same: the true SMP/E Home Directory.

Provided these intermediate symbolic links are in place before the ISPF panels are run, the process of configuring the node is same, except that rather than point to the actual SMP/E Home Directory you point to the intermediate symbolic for that configuration.

What's so special about this? It permits you to switch a configuration to another code base and leave the other intact ...

Intermediate Symbolic Links

Now when maintenance brought in, link can be changed to new code base



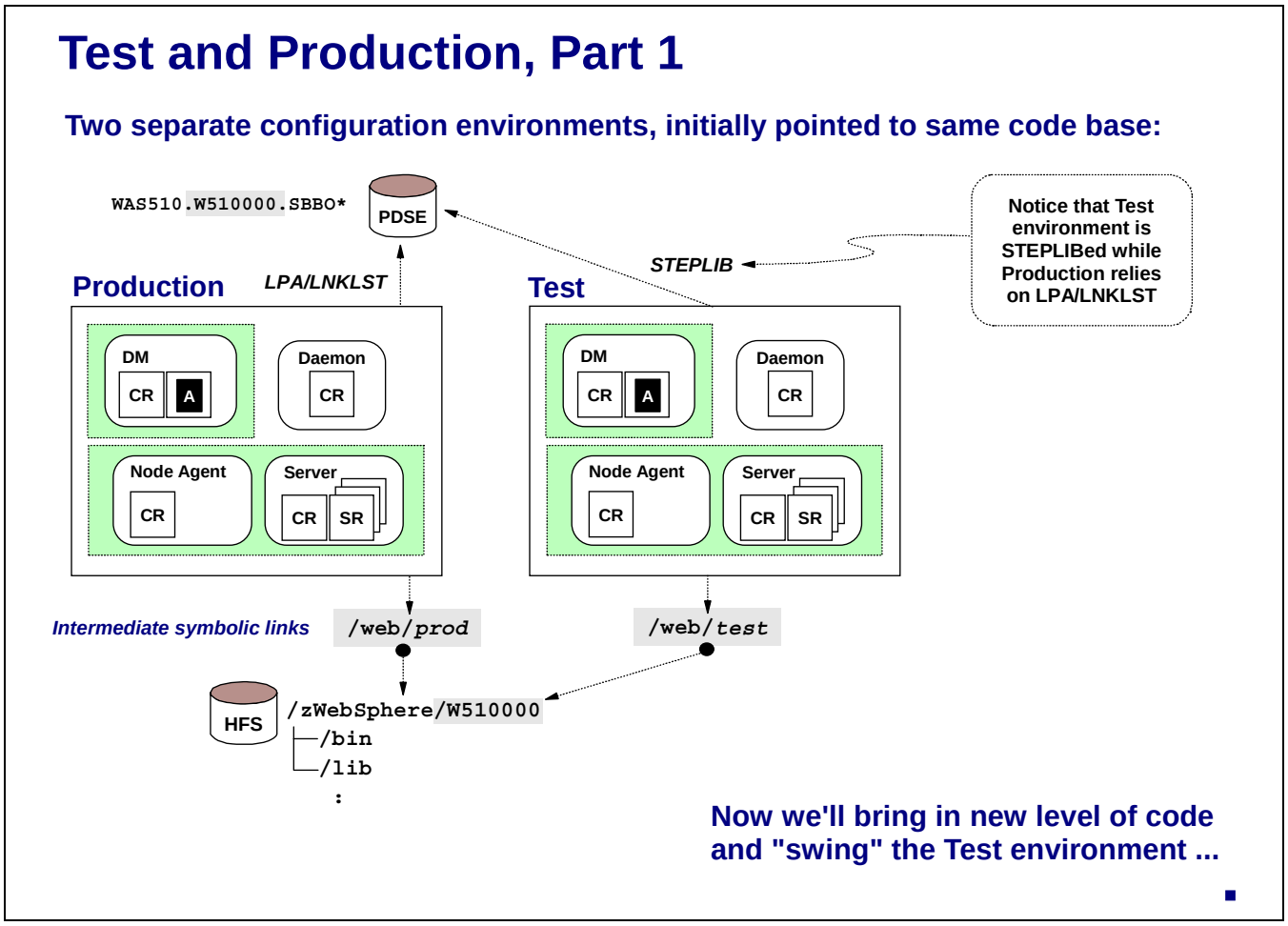
Intermediate symbolic link contents changed to point to new Install HFS

- **Introduction of maintenance is under your control**
- **This allows you to "swing" a config to new maintenance**
- **This is the key to "rolling" maintenance through a cell (more in a bit)**

Here's the same scenario as the previous chart, but now a new level of maintenance is brought into the picture. The new copy of the WebSphere SMP/E HFS is mounted at a different mount point than before. In fact, note that the mount points in the picture have an indicator of the maintenance release of the code mounted there. (That's something you wouldn't want to do if the mount point was the permanent value baked into the symbolic links of the configuration.)

To change configuration #2's code base, you would simply change the contents of the intermediate symbolic link so it points to the new SMP/E HFS. In effect, you're "swinging" the link to the new code.

Now the introduction of maintenance is under your control.



To illustrate the notion of isolated test and production environments, consider the picture above. Two different cells, initially using the same code base.

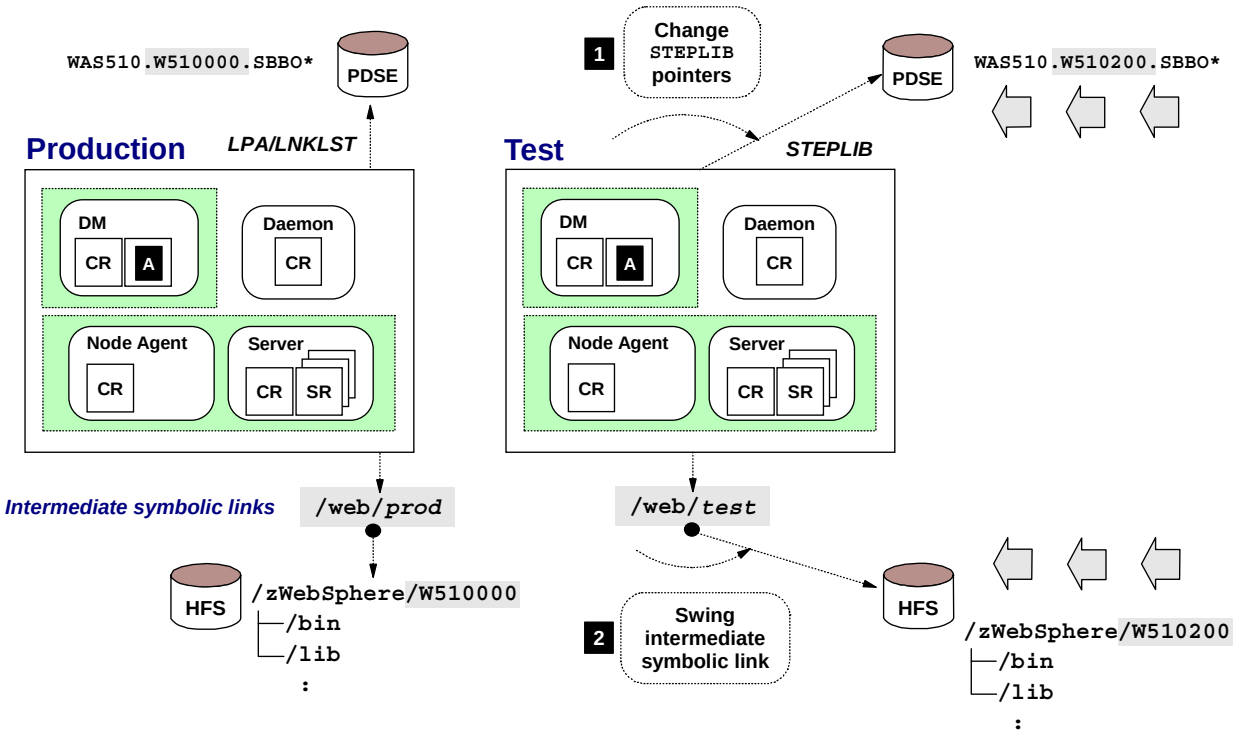
Key Point: Isolated configurations don't necessarily have to have different code bases. They simply must have the *ability* to have different code bases. But at times they might very well be using the same.

Note two things:

1. One configuration relies on LPA/LNKLST for its modules, while the other uses STEPLIB. Both could use STEPLIB, but both can't use LPA/LNKLST (unless the two configurations are on different MVS systems).
2. Each configuration employs its own unique intermediate symbolic link. Initially both intermediate symbolic links point to the same "SMP/E Home Directory," but that'll change as we roll new maintenance in.

Test and Production, Part 2

Slide in second SMP/E data sets representing new maintenance:

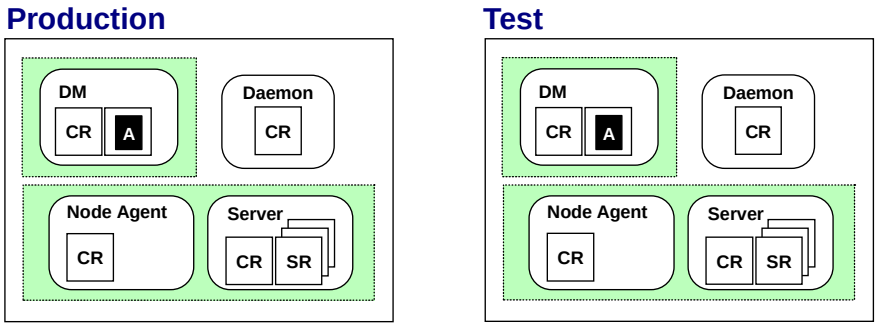


Use of intermediate symbolic links permitted this flexibility

New maintenance is brought into the picture in the form of a new set of PDSE libraries and a new HFS, mounted at a new mount point. The act of switching the Test configuration over to the new code involves switching the STEPLIB statements to the new PDSE libraries and swinging the intermediate link over to the new HFS.

It was the intermediate symbolic link that provided this flexibility. Without the intermediate symbolic link this would not have been possible.

Keys to Test/Production Isolation



Pointers to PDSE Data Sets:

- One environment LPA/LNKLST, other STEPLIB, or
- Both environments STEPLIBed
- Can't both be LPA/LNKLST and achieve isolation
  - **Unless on separate MVS images where separate refresh of LPA/LNKLST possible**
- Implies different JCL procs for test and production
  - **Likely to be the case anyway since different config mount points require different JCL start procedures (because of `SET ROOT=` value in JCL)**

Pointers to SMP/E HFS:

- Have physically separate SMP/E HFS mounts points, or
- Use "intermediate symbolic links" to provide ability to isolate

Key is the ability to isolate the configuration to a different code base.

Here's a summary chart listing the keys to isolating two configurations from one another for the purposes of test and production.

- With respect to the PDSE libraries, the key here is that both configurations can't rely on LPA/LNKLST if the configurations are on the same MVS image. If that were the case, there would be no way to refresh that code in LPA/LNKLST without affecting both configurations. So at least one configuration must use STEPLIB to have the isolation. Both could use STEPLIB if you wished.
- With respect to the SMP/E Home Directory, you could go the route of physically separate install HFS mount points, but that implies two copies of what may be the same code. The use of intermediate symbolic links provides the flexibility to "swing" a configuration away from a code base to another.

The critical thing is not that the two configurations use different copies of the code base, but rather that they have the *ability* to use different code bases.

Other Symlinks that are Created

Be aware that JMS Client Java Feature and Java Home Directory on "System Locations 2 of 2" is used to create symlinks in the configuration HFS:

```
System Locations (2 of 2)

   Specify the following for your customization, then press Enter
   to continue.

 Locations of HFS Resident Components

   WebSphere Application Server SMP/E home directory:
     /usr/lpp/zWebSphere/V5R1M0
   WebSphere JMS Client Java Feature SMP/E home directory:
     /usr/lpp/mqm/V5R3M1
   Java home directory:
     /usr/lpp/java2/J1.4
```

Issue is exactly the same as for "WebSphere SMP/E home directory"

Use intermediate symlinks to provide ability to isolate

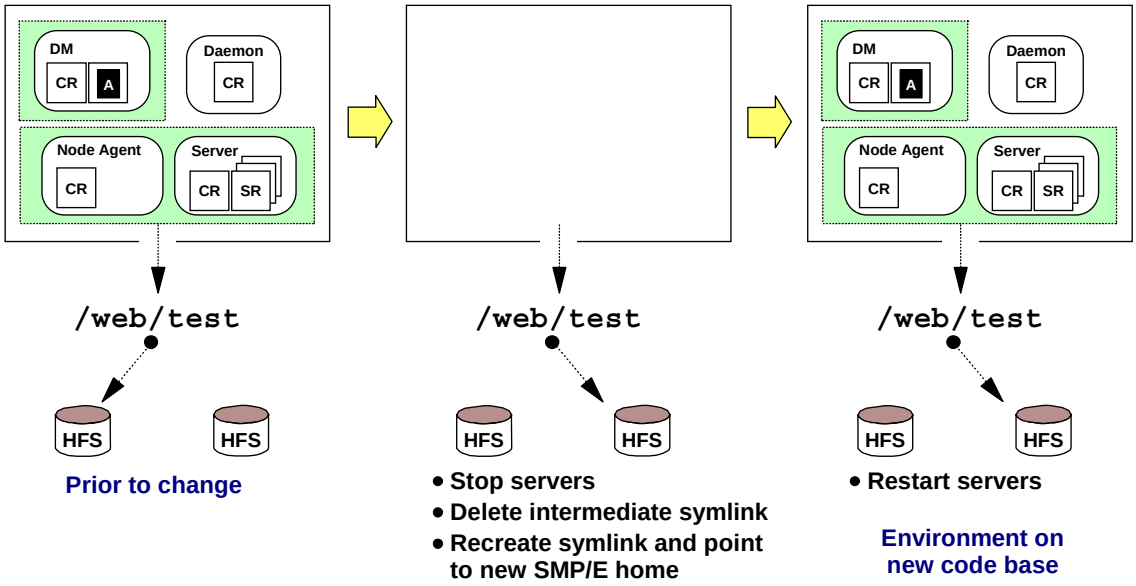
If you recall the "System Locations 2 of 2" panel, the "WebSphere SMP/E home directory" wasn't the only HFS mount point requested there. The "JMS Client" and "Java home" directories were also requested. These are used for the same thing as the "SMP/E home directory" -- to create symbolic links in the configuration HFS.

We've spent some time discussing the use of intermediate symbolic links for the WebSphere SMP/E linkage. You need to understand that these other two pointers should also be configured with intermediate links so they can be isolated if need be.

Changing Intermediate Symlink Content

Two key points here:

- **Must delete intermediate symlink and recreate; can't simply change contents**
- **If server running that's using intermediate symlink, won't be able to delete**



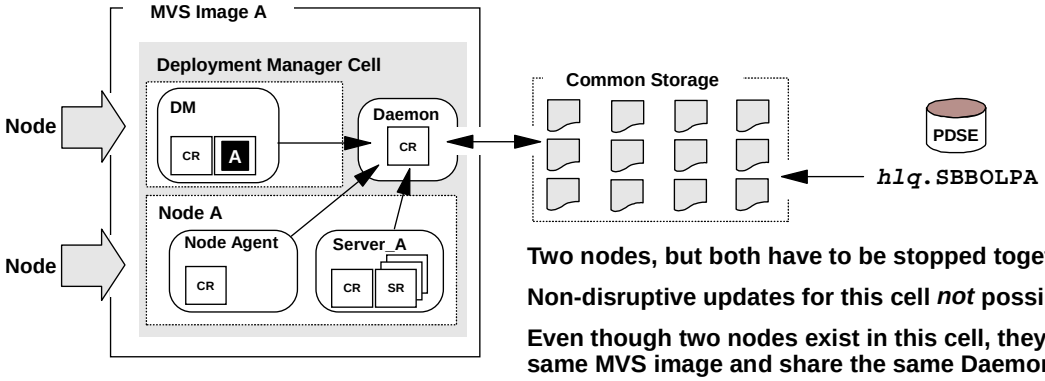
Recall that granularity of symlinks is at the node level. Means code can be introduced node by node. Be careful ... the Daemon comes into play now ...

The act of changing an intermediate symbolic link to point to a new target mount point involves stopping all the servers that make use of the intermediate link, deleting that intermediate link, and then re-creating it with the new contents. After the new intermediate symbolic link is in place the servers can be restarted.

We discussed earlier that the granularity of the symbolic links in the configuration HFS is no lower than the node level. That implies that maintenance can be introduced on a node by node basis. That's true, but you have to be very careful. The Daemon server -- not really part of any node structure -- comes into play.

Two Nodes, Same Cell, Same MVS

The Daemon holds `SBBOLPA` modules in common storage. Other servers in the same cell on that MVS image access those modules through Daemon.



Two nodes, but both have to be stopped together.

Non-disruptive updates for this cell *not* possible

Even though two nodes exist in this cell, they're on the same MVS image and share the same Daemon

Can't wait to update Daemon until after servers -- results in mixed environment: with servers at higher, LPA modules in Daemon's common storage at lower.

To update the code used by a Daemon means stopping/restarting that Daemon.

- **Stop a Daemon and all servers in that cell on the MVS image also stop**
- **Servers in cell on *other MVS images* have their own Daemons; they don't stop.**
- **Therefore, non-disruptive maintenance requires cell to span MVS images**

When new maintenance is brought into the picture, the Daemon server also needs to be updated. The Daemon server makes no use of the any of the files in the SMP/E HFS, but it does use modules from the SMP/E PDSE libraries. The Daemon server can get those modules either through LPA/LNKLST or by STEPLIB. In any event, the only way to refresh a Daemon server is to stop it and restart it with the new code level.

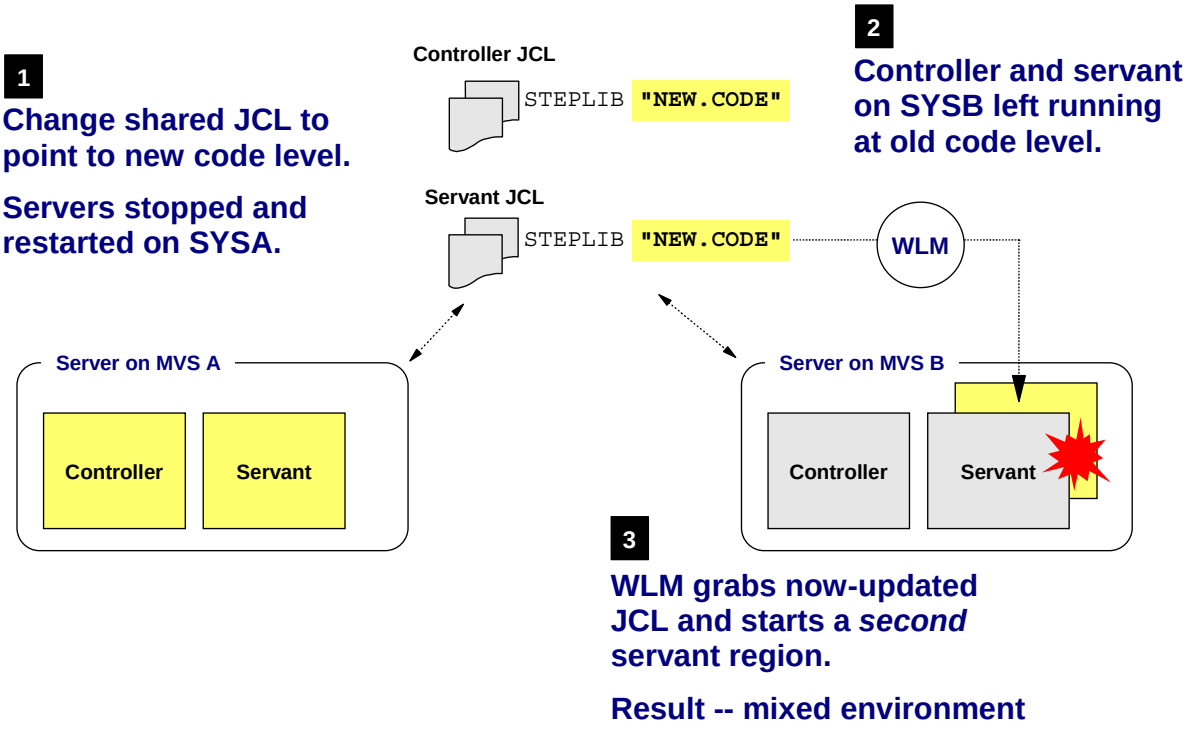
Here's the issue. The Daemon server maintains modules from the `SBBOLPA` library in its common storage, and *all the other servers in that cell on that MVS image rely on the Daemon's common storage copies*. Stop a Daemon server and all the other servers in that cell on that MVS image also stop.

The picture above shows a configuration with two nodes -- the Deployment Manager node and an application server node -- on the same MVS image. They therefore share the same Daemon server. If you go to update that Daemon's server's code, you'll have to stop/restart it. Doing so will mean the DMGR and the application servers will come down. Therefore, two nodes in a cell on an MVS image can't be migrated to new code independently; they must be migrated at the same time.

Recall that there's one Daemon server per MVS image per cell. A cell that spans MVS images will have multiple Daemons. Stopping a Daemon on SYSA will not stop servers on SYSB. Therefore, a non-disruptive configuration implies at least two MVS images.

Separate JCL By Node May Be Needed

If your environment uses STEPLIB, need to have separate servant procs for each MVS image. This is because of potential for WLM to start servant at any time:



WebSphere for z/OS Version 5 is designed to permit multiple servers to use the same JCL start procedures. This design was intended to reduce the number of update to the PROCLIB that were needed with V4. However, there's an aspect to this design that you need to be careful with when rolling maintenance into a configuration. This issue relates only to configurations that use STEPLIB.

Imagine you have a two node configuration that spans MVS images. You have it set up so servers on SYSA and SYSB both use the same JCL. You wish to roll new maintenance into SYSA. So you stop the servers on SYSA and change the JCL STEPLIB pointers to the new level of the PDSE libraries. You start the servers up and SYSA is now on the new code level.

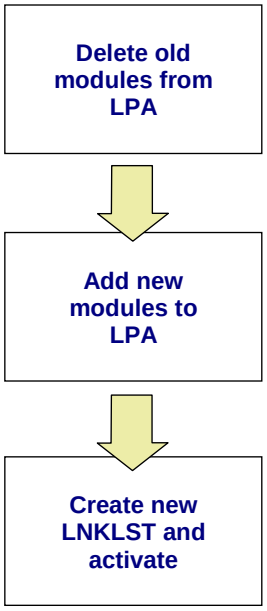
But over on SYSB workload is continuing to be processed, and let's assume WLM decides it must start a new servant region to handle the workload. But the JCL for the servant region has been changed to point to new PDSE libraries! The second servant region on SYSB comes up, but with the new level of code. The controller and the other servants (as well as the Daemon) are on the old level of code. You now have a mixed environment, which is not a good thing.

You can get around this in several ways:

- Don't use STEPLIB. If your nodes use LPA/LNKLST you can refresh the contents of LPA/LNKLST on a system-by-system basis.
- Don't use common JCL. Have separate JCL for each MVS image represented in your configuration.

Refresh LPA/LNKLST System by System

If your environment uses LPA/LNKLST, then you can control when a system is toggled to new code by refreshing LPA/LNKLST:



- Must be done module by module
- Two WebSphere libraries: **SBBOLPA** and **SBBOLOAD**
- Create member in **PARMLIB** with each module named in **LPA DELETE** statement
- Then **SET PROG=** that member to delete modules from LPA

- Easier -- may load whole libraries at once
- Two WebSphere libraries: **SBBOLPA** and **SBBOLOAD**
- Create member in **PARMLIB** pointing to new code level libraries
- Then **SET PROG=** that member to add modules to LPA

- One WebSphere libary: **SBBOLD2**
- Delete old **LNKLST** and add new
- Activate **LNKLST**

Other datasets that may be in LPA/LNKLST:

- **SCEERUN**
- **SCEERUN2**
- **SGSKLOAD**

See WP100396 for sample PARMLIB members that do these things.

If your configuration pulls the code from LPA/LNKLST, and you wish to change to a new level of the code, it involves refreshing the LPA/LNKLST for the MVS image on which the servers reside. To update the code used by the servers you will need to stop and restart the servers.

This process is not unique to WebSphere ... it's the same process you would use for any product. As a reminder, here's what's involved:

- The old modules need to be deleted from LPA. This can't be done on a library-wide basis; it has to be done on a module by module basis. Two libraries have code up in LPA -- `SBBOLPA` and `SBBOLOAD`. The way we've done it is to create a member in `PARMLIB` with all the modules in both libraries listed, and then we `SET PROG=` that member to delete the modules from LPA.

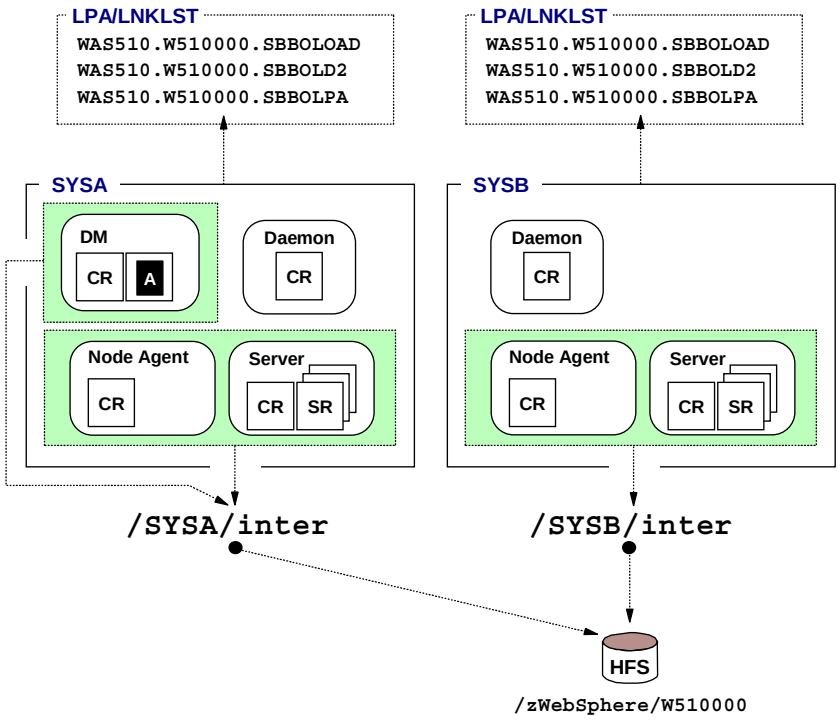
  Note: See WP100396 found in `www.ibm.com/support/techdocs`. That white paper has an example of the members used at the Washington Systems Center to delete modules from LPA.

- Adding the modules to LPA is easier ... it can be done on a library basis. Add back `SBBOLPA` and `SBBOLOAD` for the new level of code you wish to use.
- LNKLST can be refreshed by deleting the old copy and adding the new based on the new code base. Then activate the new LNKLST.

The "Systems Locations 1 of 2" panel gave you the opportunity to specify whether the libraries would be in LPA/LNKLST or not. Three other data sets were also listed on that panel -- `SCEERUN`, `SCEERUN2` and `SGSKLOAD`. They're not part of the WebSphere code distribution, but they are used by WebSphere. Changing to a new level of WebSphere code does not necessarily mean moving to a new level of these modules. We mention them here just to close the loop on the what other things may be in LPA/LNKLST and be somewhat related to the WebSphere runtime.

Example Non-Disruptive Rolling Maint.

Start: all nodes at W510000 level



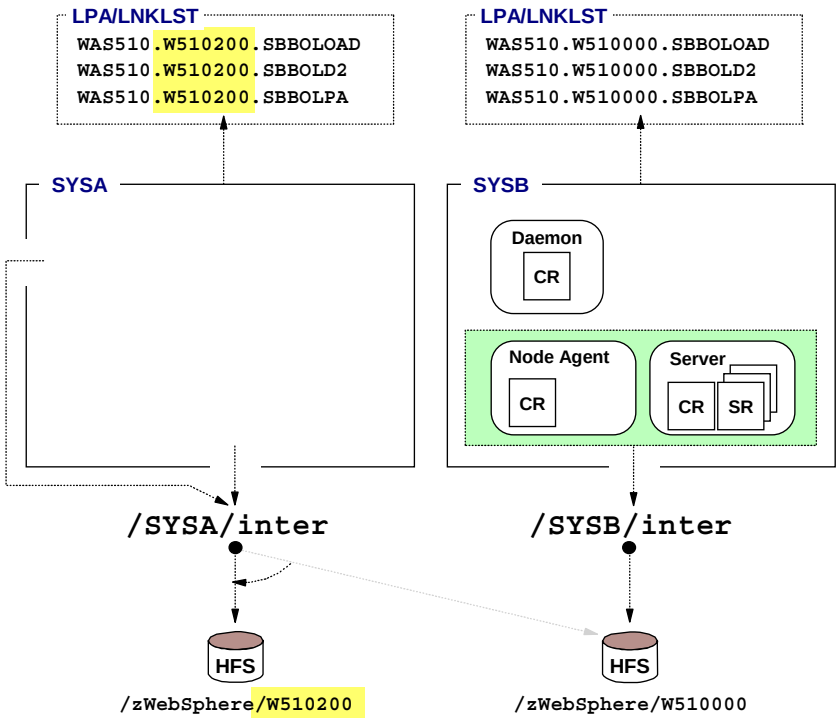
Let's now look at an example of introducing maintenance into a cell in a non-disruptive manner. Here's our starting point:

- Code is at the W510000 level. (That's just a value used at the time this presentation was written ... when you read this we may be well above that number.)
- The cell spans MVS images
- The nodes on both MVS images rely on LPA/LNKLST (though they could use STEPLIB ... the concepts aren't different, just the manner in which the PDSEs are accessed changes)
- Each MVS image has its own intermediate symbolic link. Both links initially point to the same SMP/E mount point, which has an indicator of the code level in that HFS.

We're going to introduce maintenance to SYSA first, then roll it to SYSB.

Switch SYSA Systems Nodes

Step 1 -- Stop servers on SYSA. Swing link to new code; refresh LPA/LNKLST



Assume that new maintenance is applied to the "service" environment and a copy of that environment is made and brought out to the general environment. For the sake of illustration, assume that new maintenance is the W510**200** level of code. Here's how we introduce that new maintenance to the configuration, starting with SYSA:

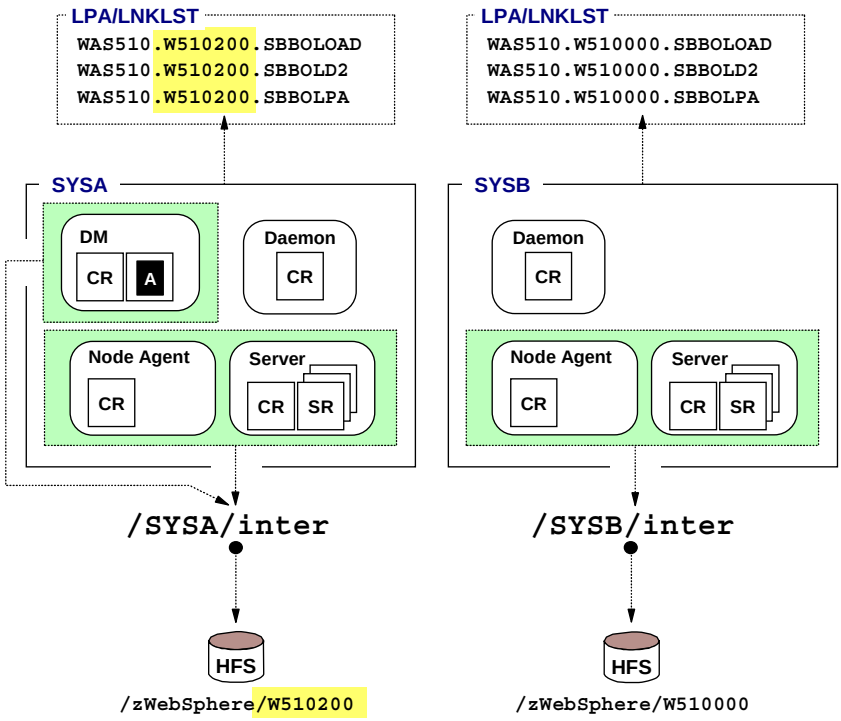
- First, stop the servers in SYSA.
- Refresh the contents of LPA/LNKLST (or toggle the STEPLIBs if the configuration used them)
- Delete the intermediate symbolic link and recreate it, pointing to the new SMP/E HFS.

All the while, your servers on SYSB continue to run. Some points about this:

- The Deployment Manager is down, but that does not mean servers on SYSB can't continue in their steady-state operations. The DMGR is not a critical piece of steady-state server operations.
- There is, however, the issue of quiescing the traffic to the SYSA servers prior to shutting them down. That's a topic beyond the scope of this document. It can be done ... there are mechanisms in routers and switches to starve traffic off from a destination prior to stopping those servers.

Restart Servers on SYSA

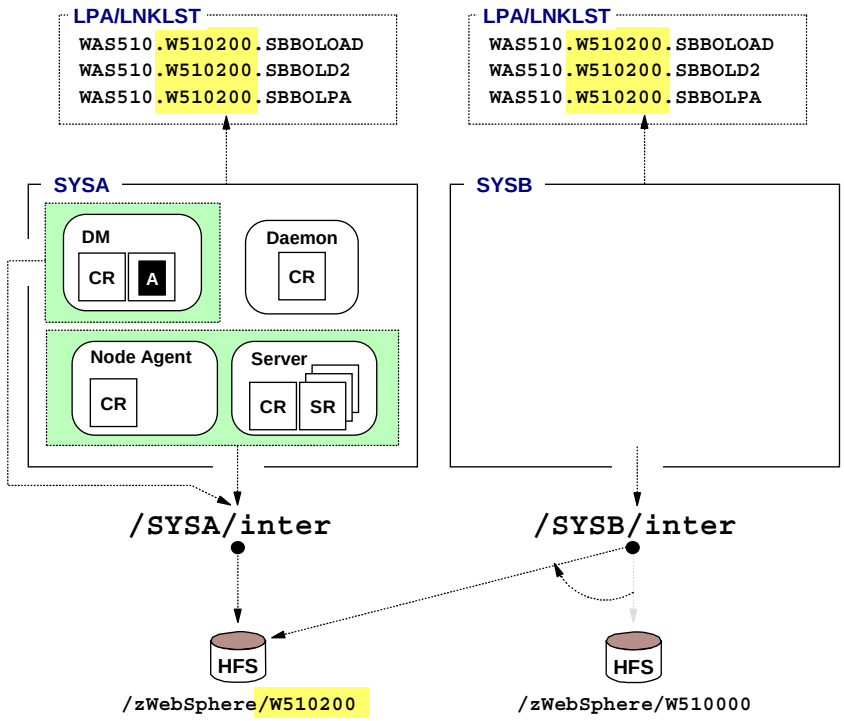
Step 2 -- Restart DMGR and servers on SYSA



Start the servers back up on SYSA. They are now running at the W510200. The PDSE libraries and the SMP/E HFS have been updated. SYSB continues to run on the older copy of the code.

Switch SYSB System Node

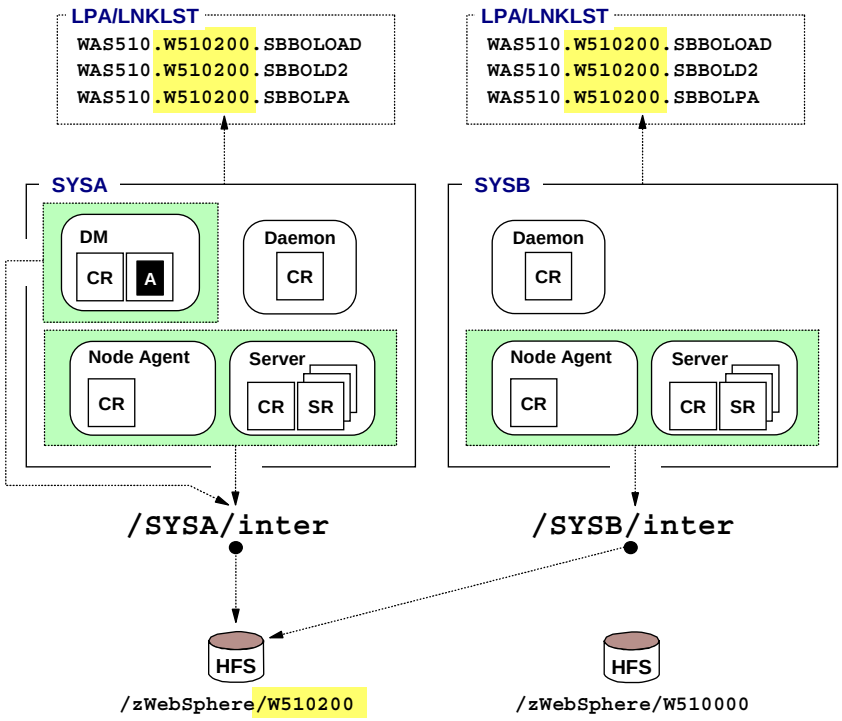
Step 3 -- Stop servers on SYSB. Swing link to new code; refresh LPA/LNKLST



Quiesce the traffic going to SYSB and route it all to SYSA. Now stop the servers on SYSB, refresh LPA/LNKLST there, delete/recreate the intermediate symbolic link so it points to the new level of code.

Restart Servers on SYSB

Step 4 -- Restart the servers on SYSB



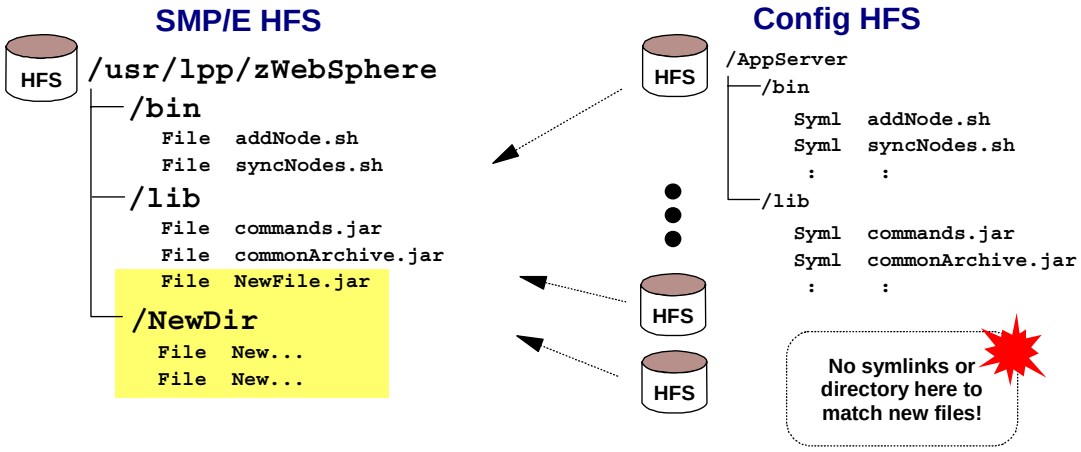
Restart the servers on SYSB. Open SYSB to traffic. The entire cell is now running on the new level of code, and at no point in time was the entire cell down. Non-disruptive maintenance achieved.

At this point you could delete the W510000 data sets if needed more packs. Or you could hold onto them just in case you needed to fall back to that level of code. The issue of backing out maintenance is taken up next.

Maintenance Dilemma

Imagine this scenario:

- **Lots of different configurations, each with own Config HFS**
- **Each Config HFS has symlinks pointing to SMP/E HFS**
- **IBM maintenance introduces several new files and a directory into SMP/E HFS**



SMP/E has no idea about all the different configuration environments. When new files or directories introduced into code HFS, updates to config HFS required complicated, error-prone `++HOLD` processing. Then came `applyPTF.sh` ...

What we'll do on this chart is introduce a dilemma that the designers of WebSphere faced. It's done here as an introduction to the `applyPTF.sh` processing we'll speak about next.

Imagine a scenario where lots of different configurations are present and each has symbolic links that point to a given copy of the SMP/E HFS.

Note: This dilemma would be the same even if all those configurations employed "intermediate symbolic links."

Now imagine that IBM comes out with new maintenance that includes a new file in the `/lib` directory and a whole new directory with files. Those new things will be in the SMP/E HFS, but *not in any of the configuration HFS*. The symbolic links in the configuration HFS point back to files in the SMP/E HFS, but include only those files present at the time the configuration was built.

Two things are going on here:

- Because most people apply SMP/E maintenance in a "service" environment and simply copy out the data sets, there's no way for SMP/E to act upon all the different configuration HFS structures.
- Even if SMP/E was used to apply maintenance directly against the SMP/E HFS pointed to by the configuration HFS, SMP/E would have no idea how many or where all those configuration HFS structures were.

The solution to this dilemma used to be an extensive list of `++HOLD` instructions. The problem with this approach, of course, is that's it's time consuming, prone to error, and could introduce problems that would be hard to debug, wasting everyone's time. So an automated update process was introduced ... `applyPTF.sh`.

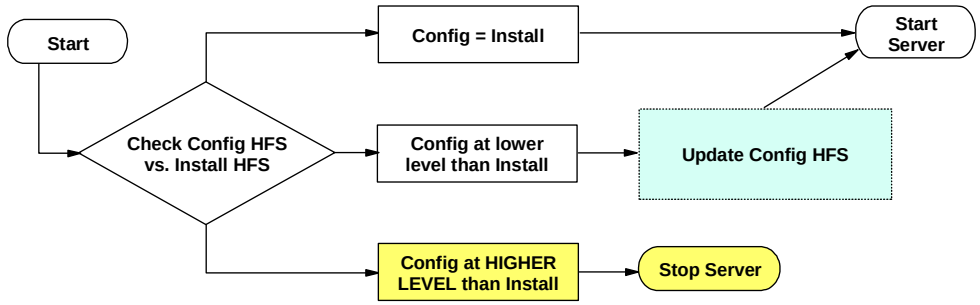
`applyPTF.sh` Processing

Shell script invoked in controller JCL checks to see if Configuration HFS needs updating:

All controller JCL procedures (DMGR, appserver, Node Agent)

```
//G51ACR  PROC ENV=,PARMS=' ',Z=G51ACRZ
// SET ROOT='/team##/g5cell'
// SET FOUT='properties/service/logs/applyPTF.out'
//APPLY   EXEC PGM=BPXBATCH,REGION=0M,
// PARM='SH &ROOT./&ENV..HOME/bin/applyPTF.sh inline'
```

`applyPTF.sh` compares Config HFS with Install HFS, then takes this action:



This would be the case if you simply dropped back to lower maintenance after config HFS had been updated

Means *how* you back off maintenance is critical ...

Somewhere back in the W501 version of the code IBM introduced `applyPTF.sh`. That shell script's role in life is to update the configuration HFS based on update instructions found in the SMP/E HFS linked to by the node. The way `applyPTF.sh` is invoked is that statements are included in the JCL for all the *controller* procs (not servant procs). The logical flow of events is illustrated at the bottom of the chart:

When a controller starts, `applyPTF.sh` is run. If checks the maintenance level of the configuration HFS for that controller against the SMP/E HFS linked to by the configuration. There are three "states" that might exist:

1. The two HFS structures are the same. That would be the case for most of the controller startups after new maintenance has been around a while. In this case, `applyPTF.sh` simply stops and the controller is then brought up.
2. The configuration HFS is at a lower level than the SMP/E HFS seen. This would be the case when a new maintenance level is introduced. In this case `applyPTF.sh` does what it was designed to do -- update the configuration HFS.

Notes:

- Remember that a configuration is for a node. A node may have lots of servers with controllers in it. The *first controller* that starts up after new maintenance is introduced is the one that updates the configuration HFS for that node. All subsequent controllers -- each of which have `applyPTF.sh` in their JCL as well -- will see the two HFS structures are equal and simply go straight to starting the servers.
- The process of updating a configuration HFS may take some time. For example, if a new copy of the administrative console EAR file is being installed, it may take 10 or more minutes.

3. The third state is that the configuration HFS is at a *higher* level than the SMP/E HFS. This would be the case when an SMP/E HFS was simply backed off after the configuration HFS had been updated. That is not good -- WebSphere won't allow a server to start if it sees that the configuration HFS is higher than the SMP/E. So it becomes critical how maintenance is backed off a configuration. You can't simply swap out the install base -- you have to bring the configuration HFS back to the previous level as well.

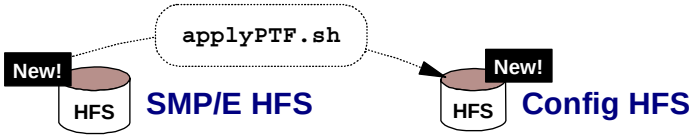
   Note: Please understand that if applyPTF.sh never runs, the configuration HFS is not updated. So if you roll new maintenance into an environment and discover a problem before any controller in the node is started, then the configuration HFS is where it was before the maintenance. In that case you can simply back out the SMP/E HFS. But only if applyPTF.sh hasn't yet run.

How you back out maintenance is covered next.

One Way to Back Out Maintenance

Snap a copy of the Config HFS *just before* new maintenance is brought in

Bring new maintenance in ... `applyPTF.sh` updates Config HFS



Bad maintenance! Restore previous SMP/E HFS and copy of Config HFS

Careful!

- **If backup copy isn't recent, falling back may mean you lose configuration changes made since backup taken.**
- **If after maintenance applied you find you have to fall back, any post-update changes to configuration will be lost.**

Another way: `backoutPTF.sh`

The easiest way to back out maintenance is to take backup copy of your configuration HFS data sets ***just prior*** to the point when `applyPTF.sh` runs. Then if the maintenance proves to be bad, you can simultaneously restore both the previous SMP/E HFS and the copy of the configuration HFS.

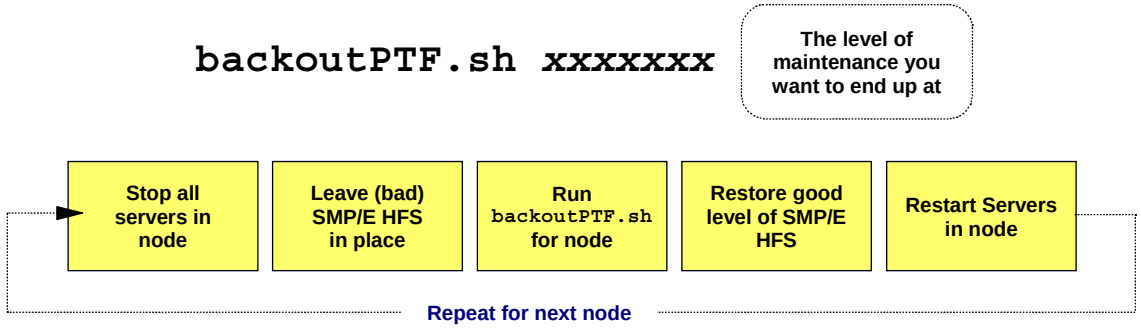
Note: Please note that any configuration changes you make *after* taking a backup copy will not be in the backup copy. So it is critical that the backup of the configuration HFS be made just before the new maintenance is brought into the picture. Two points:

- Don't rely on a backup of the HFS that's not really "fresh" and current
- If you apply maintenance and make some configuration changes and only later find that you need to drop back, restoring the backup copy of the configuration HFS will mean those configuration changes will be lost.

But if you don't have a copy of the configuration HFS, you can still back off maintenance by running a shell script called `backoutPTF.sh`.

Another Way

Another utility -- `backoutPTF.sh` -- is provided. This will restore Config HFS to the specified level. You'll find this in the `/bin` directory of each node.



Notes:

- **Process needs "bad" SMP/E HFS left in place so it can know what changes to back out of Config HFS (list of updates for each level of maintenance kept in SMP/E HFS)**
- **Process can be lengthy ... all the while servers are down**
- **Remember -- if `applyPTF.sh` not run against node, don't need to back out anything**

Message: copy/restore of Config HFS is probably better way to go.

WebSphere comes with a utility called `backoutPTF.sh` that reverses what `applyPTF.sh` does. The shell script takes as a parameter the maintenance level you want to end up at. The key to this is that you have to leave the most recent (and presumably "bad") SMP/E HFS in place because that HFS has all the information in it that tells `backoutPTF.sh` what to back out. (It was that same information `applyPTF.sh` used to update the configuration HFS, so it makes sense that `backoutPTF.sh` would use it to reverse the updates.)

`backoutPTF.sh` operates against the node. If you have multiple nodes you want to back out maintenance on, then `backoutPTF.sh` must be operated against each node.

Note: You can't fall back any further than the code level used when the configuration was initially built.

So the process to use `backoutPTF.sh` is this:

- Stop all the servers (it won't run if any of the servers in the node are up)
- Leave the latest (and presumably "bad") SMP/E HFS in place
- Run `backoutPTF.sh` and provide the maintenance level you want to end up at.
- Un-mount the "bad" SMP/E HFS and restore the previous level (the "good" level of maintenance)
- Start the servers in the node
- Repeat for each node you need to restore to the previous level of code.

This is not a trivial or quick process. It can take a long time to back out the changes. For that reason, you may find that restoring a copy of your configuration HFS is a more appropriate way to bucket changes to your configuration HFS.

End of Document