WebSphere Application Server for z/OS V6.1

Changing a Cell's Host Name and System Name

Using the new WSADMIN AdminTask object to quickly and easily change the host name and system name used by a WebSphere for z/OS cell so it'll start on a different Sysplex

This document can be found on the web at:
www.ibm.com/support/techdocs
Search for document number **WP100792** under the category of "White Papers"

Version Date: June 26, 2010
Please see "Document Change History" on page 22 for updates provided in this version of the document.

IBM Washington Systems Center

Don Bagwell
IBM Washington Systems Center
301-240-3016
dbagwell@us.ibm.com

Many thanks to **Rohith Ashok** from WebSphere for z/OS Development organization.

Table of Contents

Executive Overview

In the past, if you wanted to change the IP host name or system name values that were baked into a cell's configuration XML, it required extensive hand-modification of XML files. The process was quite prone to error.

The need to change the host name and system name is most often associated with disaster recovery *testing* -- trying to start the WebSphere configuration on the disaster recovery site's Sysplex.

Note: During an *actual* disaster recovery a *copy* of the original system is started at a disaster recovery site, using all the same system values as before -- host name, IP address, system name and Sysplex name. So modifying the values isn't necessary. But testing is a different matter -- the original site is still up, so the test site would need to use a different host name and possibly system name as well.

New WSADMIN functions -- AdminTask changeHostName, modifyNodeGroupProperty

A new set of WSADMIN scripting methods[1] provides the simple answer. For example, to change the host name and system name for a node involves one and perhaps both[2] of the following commands:

```
$AdminTask changeHostName {-nodeName wpdmnode
                -hostName wsc4.washington.ibm.com -systemName SYSD}

$AdminTask modifyNodeGroupProperty DefaultNodeGroup
                {-name was.WAS_DAEMON_protocol_iiop_daemon_listenIPAddress
                -value wsc4.washington.ibm.com}
```

WSADMIN does all the work -- *it* changes all the XML files.

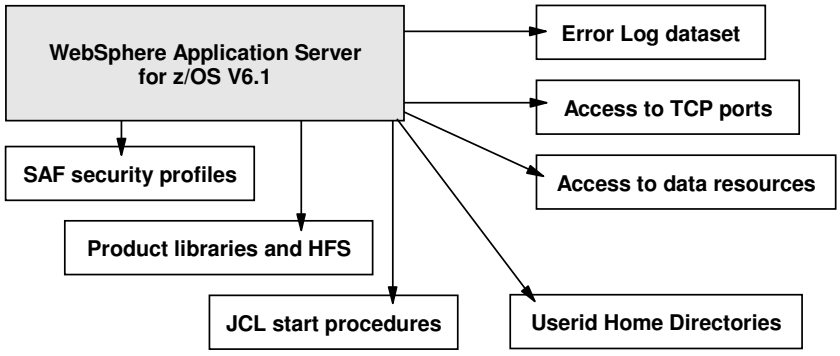
Note: We'll illustrate how that's used in a bit. For now, all we want is for you to understand that the commands above replace a dozen or more manual steps. And it removes a good deal of the human error potential.

Note about the length of the commands

The `AdminTask` commands you see above are long. If you are using WSADMIN in interactive mode with OMVS, the input field may not be long enough to contain the whole command. Using Telnet gets around that restriction, as would invoking WSADMIN using JCL batch with the `-c` parameter.

Important assumptions

Changing the host and system names is not the only thing that allows WebSphere to operate on another system. WebSphere relies on quite a few things *outside its configuration* HFS to operate:



A WebSphere cell relies on many things being present ... not just its configuration HFS

[1] WSADMIN is the programmatic scripting interface to WebSphere. With WSADMIN it's possible to do things like install applications, start and stop servers, and many other things -- all programmatically.

[2] See “A unique case -- moving a node but leaving the DMGR where it was configured” on page 19.

The main point here is that these things must be present for the WebSphere servers to start and operate. They need to be available for WebSphere to operate.

Let's quickly review each one:

- *SAF security profiles*

  This *must* be identical to the supporting profiles on the original site. Many companies copy their security database over, or use mirroring technology to ensure one site's SAF profiles are the same as another site's. Do not try to piece together the profiles manually -- you'll miss something and create a very difficult problem to debug. Make sure your SAF database is duplicated, or run the BRAK jobs to recreate the profiles exactly.

- *Product libraries and HFS*

  This involves two things:

  1. The data sets such as `SBBOLOAD`, `SBBOLD2` and `SBBOLPA` -- if you use `STEPLIB`, then make sure they properly point to the libraries. (And be aware that `STEPLIB` is not *just* in JCL but also in a shell script called `setupCmdLine.sh`)
  2. The product HFS, which has a data set name that ends with `SBBOHFS`. Depending on how your configuration references the product HFS, you may be forced to mount this HFS on a mount point with the exact same name as you used at your original site. If you used "intermediate symbolic links" then you have a degree more flexibility.[3]

- *JCL start procedures*

  They must be present, and have the same names as before. WebSphere maintains the JCL procedure names in its configuration. Preserving those names as the same between the original and DR site is important, particularly for Daemon JCL procs.

- *Userid home directories*

  Starting with V6 and continuing into V6.1, the "security domain" `BBOSBRAM` job is used to create a home directory for the Config Group, the Servant Group and the Unauthenticated Guest group. In the past these were assumed to be `/tmp`, now they are explicitly pointed elsewhere. By default the `BBOSBRAM` job creates a directories under `/var/WebSphere/home`, one directory per group ID with the ID's name as the directory under `/home`.

  If you fail to provide the home directory for each ID as known by RACF, the controller job will fail to start, and the error message you'll see is very cryptic.

  ```
  FZSR01C – STEP WAS EXECUTED – COND CODE 4095
   :
  FZSR01C FZSR01C – STEP WAS NOT RUN BECAUSE OF CONDITIONAL
       EXPRESSION ON STATEMENT 9
  ```

  If you happen to look in the `/properties/service/logs/applyPTF.out` file, you'll see this:

  ```
  FSUM1004 Cannot change to directory </var/WebSphere/home/FZCFG>
  ```

  That's the true story ... the directory isn't there. This is a very easy thing to overlook when bringing a configuration over from another site.

- *Error log, TCP ports and data resources*

  The main point here is that for WebSphere to work, it needs to have access to these things. The error log you could probably get by without it being present. WebSphere will definitely need access to its defined ports. Remapping the ports manually is possible, but time-consuming. And for your applications to do any work, they'll of course need access to things like CICS or DB2 or whatever data source those applications make use of.

Some names won't change

The Washington System Center's recommended naming convention has place holders for "system identifier." So, for instance, a node's short name might be FZNODEC, where C represents SYSC. The process we'll illustrate here will not change that name. So if on your DR test site it's really starting on SYSA, then you'll simply have to keep in mind things like that.

---

[3] See WP100396 at `www.ibm.com/support/techdocs` for a description of how a configuration references the product HFS, what "intermediate symbolic links" are and why they are valuable.

We do not recommend you try changing the short names. Some are tied to RACF profiles and changing them could break things.

Simply understand this -- the process we'll show you here makes the changes necessary so WebSphere will start on a system with different host and system names. But not all names will change. Only those necessary for the servers to start on a new system.

Virtual Host definitions not changed

One file where host names might be found is the `virtualhosts.xml` file. By default your host names won't be there, but would be if you modified or added a host alias to include the specific host name. The WSADMIN AdminTask function will not change those values. That means it's possible an application won't work on the new system if it's bound to a virtual host and makes use of an alias that has a specific host IP name on it.

An Illustration of the AdminTask Object

We'll break this section into four main headings:

Simple outline for those familiar with WSADMIN

Please see the more detailed sections for a more complete explanation of each of these steps.

Network Deployment cell

- ☐ Invoke WSADMIN with `-conntype NONE` in DMGR node
- ☐ Issue AdminTask command to change host name and system name for the DMGR node:

```
$AdminTask changeHostName {-nodeName <node_long>
          -hostName <host_name> -systemName <system_name>}
```

- ☐ While still working with the DMGR's master configuration, issue the same command for *each federated node* in the cell, changing the node long name, host name and system name as appropriate
- ☐ Issue the AdminTask command to change the host name for the cell's Daemon instances:

```
$AdminTask modifyNodeGroupProperty DefaultNodeGroup
  {-name was.WAS_DAEMON_protocol_iiop_daemon_listenIPAddress
   -value <host>}
```

- ☐ Save the configuration changes -- `$AdminConfig save`
- ☐ Start the Deployment Manager
- ☐ Use the `syncNode.sh` shell script found in *each federated node's* `/bin` directory to synchronize the change from the master configuration to the node:

```
syncNode.sh <DMGR_host> <SOAP_port> -username <admin_ID>
   -password <admin_pw>
```

- ☐ Start the servers in the federated nodes

Standalone Server cell

- ☐ Invoke WSADMIN with `-conntype NONE` in Standalone server node
- ☐ Issue AdminTask command to change host name and system name for the node:

```
$AdminTask changeHostName {-nodeName <node_long>
          -hostName <host_name> -systemName <system_name>}
```

- ☐ Issue the AdminTask command to change the host name for the Daemon instance:

```
$AdminTask modifyNodeGroupProperty DefaultNodeGroup
  {-name was.WAS_DAEMON_protocol_iiop_daemon_listenIPAddress
   -value <host>}
```

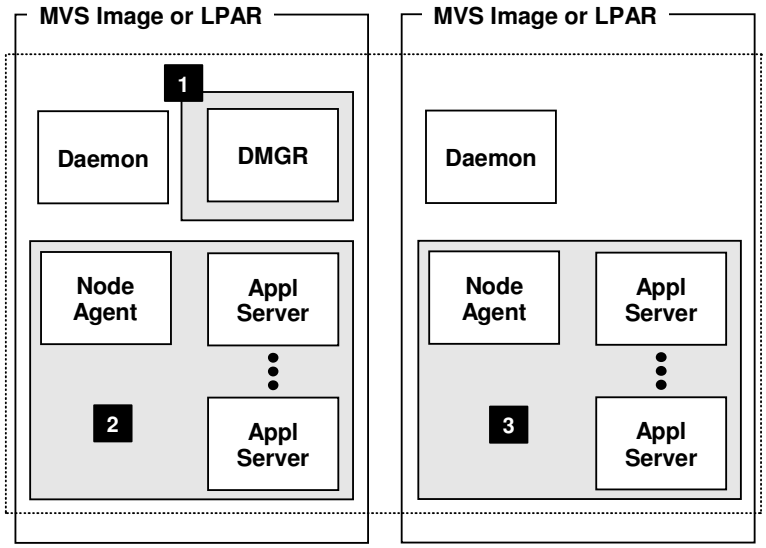
- ☐ Save the configuration changes -- `$AdminConfig save`
- ☐ Start the standalone server

Important concepts

Before launching into an illustration of how to use the AdminTask object, let's quickly cover some important things.

Nodes and cells

A "node" is a logical collection of servers, and by definition is limited to a single MVS image or LPAR. In addition, the Deployment Manager has its own node as well. The following picture illustrates a typical WebSphere cell consisting of three nodes:



Typical network deployment cell with three nodes

The reason we remind you of this is because the changing the host and system names is done on a node-by-node basis. For this picture that implies making three sets of updates -- one for the DMGR node, and one for each federated application server node.

Note: Actually, *four* updates -- `AdminTask changeHostName` for each node, and `AdminTask modifyNodeGroupProperty` once for the Daemons, which technically reside outside the nodes. You'll see this illustrated in this document.

Each node has its own configuration structure. The Deployment Manager, for example, has a long structure starting with `/DeploymentManager` and going many directories deep. Application server nodes start with `/AppServer` and go down many directories as well.

Note: Ignore for this discussion the question of shared or separate HFS -- it has no bearing in this issue of updating host and system names.

The XML files that hold the host name and system name information are deep in each node's configuration structure, scattered in a few different places. The beauty of the WSADMIN solution is that it -- WSADMIN, not you -- is responsible for finding those files and updating them.

The master configuration and synchronizing changes

The Deployment Manager is unique in that it holds what's known as the "master configuration" -- a copy of every node's configuration information, including its own. Changes made through the Admin Console are first written to the master configuration, then copied out to the nodes' configuration trees.

The act of copying the changes out to the nodes is known as *synchronization*. The changes are in the form of whole XML files that are simply file-transferred from the DMGR to the node and

put in place there. We mention this because what we'll illustrate is the changing of information in the master configuration using WSADMIN and AdminTask, then synchronizing those changes out to the nodes using the `syncNode.sh` shell script.

> **Note:** Changes to the DMGR's master configuration using WSADMIN can be made while the DMGR is *down*. Synchronization using `syncNode.sh` requires that the DMGR be up and running.

Saving changes

A small but important detail to know is that when you make a change to the configuration using the Admin Console or WSADMIN, the changes are first made to a temporary location. They are "committed" to the master configuration only when a "save" operation is performed.

So in addition to the WSADMIN AdminTask commands, we'll also need on other WSADMIN command, a very simple one:

```
$AdminConfig save
```

> **Note:** FYI ... this command will also regenerate non-XML files such as `was.env`. So the changes made to the XML files by `$AdminTask` will get refreshed out to non-XML files when the save operation is executed.

The `syncNode.sh` shell script

The `syncNode.sh` shell script is simply a script that makes contact with the DMGR and initiates the node synchronization process. Changes in the form of XML files updated by WSADMIN are then copied to the node from which `syncNode.sh` is invoked. When it's completed, the node is updated with the changes made to the master configuration.

The use of `syncNode.sh` only applies to Network Deployment cells -- that is, a cell with a Deployment Manager node and at least one application server node. Standalone servers don't need to worry about synchronization because by definition they have only one node.

What WSADMIN is

WSADMIN is WebSphere administrative scripting interface. Two words jump out there:

- *scripting* -- WSADMIN makes use of a script to tell the utility what to do and what changes to make. Two scripting options are available: JACL and Jython. Both are industry standard scripting languages. For this document we'll illustrate the use of JACL.
- *interface* -- WSADMIN is really an API to WebSphere. WSADMIN provides five "objects" -- AdminApp, AdminConfig, AdminControl and AdminTask. AdminTask is the new one with V6.1, and is the one we'll use to change the host name and system names. Each object has "methods" (sub commands) which do different administrative things.

You invoke WSADMIN with a shell script -- `wsadmin.sh`.

We'll make use of only a very small subset of the power of WSADMIN for this exercise. With WSADMIN you can do pretty much everything you could do with the Admin Console, only do it programmatically.

> **Note:** If you want to know more about WSADMIN, you can take a look at WP100421 at `ibm.com/support/techdocs`. That document is designed to help someone get from almost no knowledge of WSADMIN to a good working knowledge. It's written with V5 in mind, so it's starting to get a little old, but the main concepts still apply.

Modes of WSADMIN operation

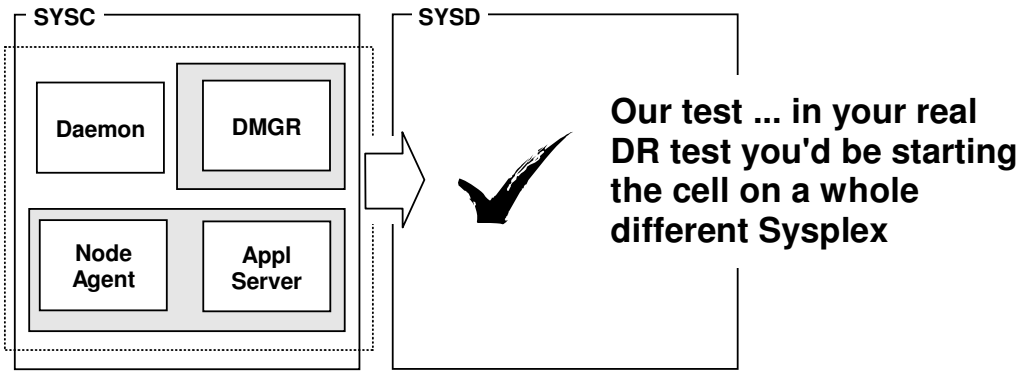
WSADMIN has two basic modes of operation:

1. "Local" -- WSADMIN operates against the configuration HFS directly. No servers need to be up for this to work. WSADMIN is smart enough to know where key configuration files reside in the HFS, and it will directly modify those files. Contrast this with the next mode ...
2. "Remote" -- WSADMIN makes contact with the running Deployment Manager and uses the management interface of it to make the changes. The DMGR makes the changes to the configuration files, WSADMIN is simply asking the DMGR to do it. It's called "remote" mode simply because a TCP connection is made to the running server.

In this document we'll illustrate the use of "local" mode -- direct modification of the configuration HFS. We do this because we need to make the changes *before* the DMGR is started. In a DR testing scenario, we need to make the changes so that the DMGR will start. WSADMIN in local mode provides this ability.

Illustration of Network Deployment Cell

We didn't have a whole different Sysplex environment to illustrate the use of the new AdminTask object, so we did the next best thing ... we moved a cell from one LPAR to another in a Sysplex. The two LPARs have different host names and different system names, so it comes very close to illustrating the bringing up of a cell a DR site:



The "Disaster Recovery test" scenario used for the Network Deployment cell illustration

??? "Not a fair test!" You say. "The Sysplex name is the same." Yes and no. See "The Sysplex name in the WebSphere configuration XML" on page 19 for an explanation of where the sysplex name is actually stored in the configuration XML, and how to change it if you wished.

Here was our "before" and "after" information:

	Before	*After*
Host Name	`wsc3.washington.ibm.com`	`wsc4.washington.ibm.com`
System Name	`SYSC`	`SYSD`

Stopped all servers in cell

The WSADMIN command is capable of making the XML changes with the servers up, but for the changes to take effect the servers would need to be stopped and restarted. So we'll illustrate making the changes while the servers were down.

Invoked WSADMIN client

The WSADMIN client is the `wsadmin.sh` shell script, which was located at the following location:

Location of WSADMIN client shell script

Notes:

1. The mount point we had for our configuration HFS
2. The default home directory for the DMGR, `/DeploymentManager`
3. WebSphere makes use of "profiles," which are different configurations. For z/OS, we only have one profile possible, and it'll always be `/default`. It's important to drive down into the `/profiles/default/bin` directory, and not the higher `/bin` directory.
4. The WSADMIN client shell script itself. This is what we invoked.

Here's what we did to invoke WSADMIN:

```
su fzadmin 1

     2          3
./wsadmin.sh -conntype NONE
```

Commands used to invoke WSADMIN client

Notes:

1. It's always best to run WSADMIN under the authority of the Admin ID. That insures access to all the necessary directories.
2. The shell script itself
3. The parameter `-conntype NONE` specifies that WSADMIN should operate in "local" mode; that is, no connection to a running server and all changes are made directly to the configuration HFS.

??? How did it know which configuration to operate against? It is based on which copy of `wsadmin.sh` is invoked. Each node has its own copy. With `-conntype NONE`, WSADMIN assumes it is to operate against the same node from which it was invoked.

The result we saw was:

```
WASX7357I: By request, this scripting client is not connected to any
server process. Certain configuration and application operations will 1
be available in local mode.

WASX7029I: For help, enter: "$Help help" 2

wsadmin> 3
```

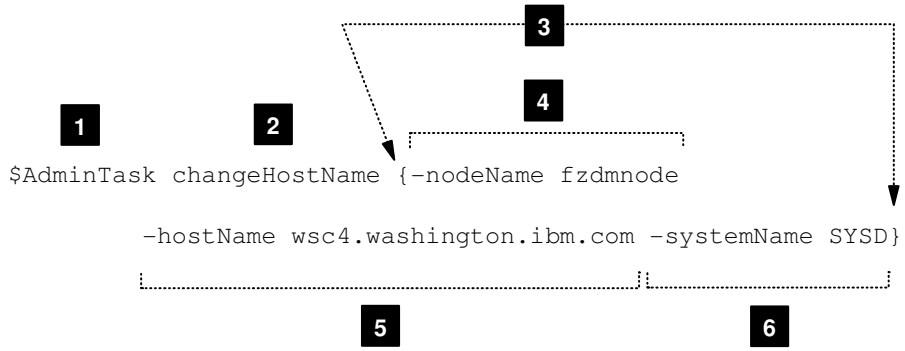
WSADMIN command prompt

Notes:

1. Message indicating that not all WSADMIN commands will work. When in local mode, only those involving changes to configuration files work, but things like starting and stopping servers will not.
2. Message pointing to the $Help object, which we'll not cover in this document.
3. The command prompt. It was here we entered our commands.

Modified host name and system name for DMGR node

We were now ready to change the Deployment Manager node's host name and system name. The command we entered was:



Command entered to change host and system names

Important! The command is entered as *one line*. We broke it here because of space limitations.

Notes:

1. The WSADMIN object that implements the function we wanted. The dollar sign out front tells WSADMIN that the scripting type is JACL.
2. The method on `AdminTask` that is used to change host name and system name
3. Notice the open brace and close brace ... this is what WSADMIN uses to "bracket" the parameters to the `changeHostName` method. If you forget a brace, WSADMIN will let you know that it had trouble "evaluating the JACL expression."
4. The `-nodeName` parameter names the node *long* name that will be the target of the changes. This cell had two nodes, but for now we were operating on just the DMGR node.
5. The `-hostName` parameter provides the *new* host name. Notice that there is no checking for what the existing host name is. This is *not* a "find and replace" operation. This will replace whatever is there with this value.
6. The `-systemName` parameter is what's used to provide the new system name. If by some luck the system name does not need changing, you may omit this parameter and its value. In our case it needed changing because the system name was different.

This function provides no positive feedback ... it simply comes back with the command prompt if things worked okay.

Important! The changes were in a temporary location at this point. We needed to save the changes, which we did with the `$AdminConfig` object we show in a moment.

Modified host name and system name for application server node

This cell had two nodes. We just changed the values in the DMGR node. Now it was time to change the values in the federated application server node.

Note: We continued to use the copy of WSADMIN started from the DMGR node. That meant the changes were made in the DMGR's **master configuration**, *not* the appserver node configuration. That's exactly what we wanted. We then used `syncNode.sh` to synchronize the changes. We'll show that in a moment.

We used essentially the same command as used for the DMGR node, but with the node long name changed:

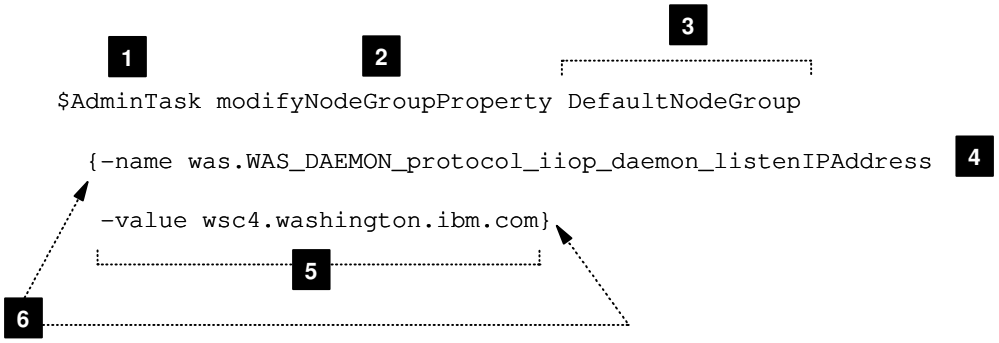
```
$AdminTask changeHostName {-nodeName fznodec
          -hostName wsc4.washington.ibm.com -systemName SYSD}
```

Command entered to change host and system names for federated node

Modified host name for Daemon instance

Daemon servers also make use of a host name value. That information is kept in the "node group" information for the cell.

The command we entered to modify this value was:



Command entered to change Daemon instance host name

Important! The command is entered as *one line*. We broke it here because of space limitations.

Notes:

1. The WSADMIN object that implements the function we wanted. The dollar sign out front tells WSADMIN that the scripting type is JACL.
2. The method on `AdminTask` that is used to modify the Daemon instance information in the "node group" information. It is not important that you know what a "node group" is.
3. The name of the node group that was modified. By default the name will be `DefaultNodeGroup`. Assume the default name unless you know for certain it's something else.
4. The `-name` parameter tells what property to modify. This big long value is a keyword string ... the Daemon instance host name is named on this property in the node group XML file.
5. The `-value` parameter provides the *new* host name. Notice that there is no checking for what the existing host name is. This is *not* a "find and replace" operation. This will replace whatever is there with this value.
6. The braces that enclose the parameter list.

Important! The changes were in a temporary location at this point. We needed to save the changes, which we did with the `$AdminConfig` object we show in a moment.

This command *did* report back success:

```
was.WAS_DAEMON_protocol_iiop_daemon_listenIPAddress
 (cells/fzcell/nodegroups/DefaultNodeGroup|nodegroup.xml#Property_3)
```

Positive confirmation that Daemon property was changed.

Note: The value you see may be slightly different from this.

A less common Daemon change

There's another Daemon property related to the host name, and it is:

```
was.WAS_DAEMON_ONLY_protocol_iiop_listenIPAddress
```

Note: That is not the same as we showed you earlier. It's very close, but it's not the same:

```
Before: was.WAS_DAEMON_protocol_iiop_daemon_listenIPAddress
This:   was.WAS_DAEMON_ONLY_protocol_iiop_listenIPAddress
```

This defaults to * (asterisk) which means the Daemon is to listen across all adapters. And if the configuration you're working with has the default value, then you don't need to worry about this. But if it was changed so the Daemon was bound to a *specific IP address*, then moving the configuration to another system without changing this will result in a problem.

You can check this by driving into the Admin Console of your original configuration:

System Administration ⇨ *Node Groups* ⇨ *DefaultNodeGroup*

If the field is an asterisk then you do not need to worry about this -- the asterisk means the configuration will listen across all available adapters on the target system. But if value is a specific value, then to make this work on the target system you have two choices:

1. Turn it back to an asterisk
2. Provide a specific IP listen address that's applicable to the target system

The command is very similar to before:

```
$AdminTask modifyNodeGroupProperty DefaultNodeGroup

  {-name was.WAS_DAEMON_ONLY_protocol_iiop_listenIPAddress

   -value *}
```

Here we're showing setting the value back to an asterisk. If you wanted to set it to a specific IP address, you would supply that value rather than the asterisk.

Saved configuration changes

A very simple command accomplished this:

```
$AdminConfig save
```

Notice that we made use of a different WSADMIN object -- `AdminConfig`.

Important! The changes we made were in the master configuration only, not the node's actual configuration. If we tried to start the federated node's servers on SYSD at this point it would fail because the node does not yet know about the changes. Using the `syncNode.sh` command to synchronize the node with the master configuration is what we did. We illustrate that on page 13. We had to first start the DMGR on SYSD because `syncNode.sh` requires that the DMGR be up and running for it to be able to get the changes from the master configuration.

Exited WSADMIN

This was done with the `quit` command.

Started DMGR on SYSD -- and it failed!

Note: You may not run into this. We're showing this because it highlights a subtle "gotcha" that happened to us. And it happened because of a change in the way the userid "home" directories were created in V6 and 6.1 vs. earlier versions of WebSphere.

When we tried to start the Deployment Manager, it very quickly failed with very little information as to why. The job output simply said:

```
FZDMGR - STEP WAS EXECUTED - COND CODE 4095
FZDMGR FZDMGR - STEP WAS NOT RUN BECAUSE OF CONDITIONAL EXPRESSION ON STATEMENT 9
```

The step that failed with RC=4095 was one that runs a test to see if OMVS can successfully launch a shell and return. The output from that test went to:

```
/wasv61config/fzcell/fzdmnode/DeploymentManager/properties/service/logs/applyPTF.out
```

In that file we found:

```
FSUM1004 Cannot change to directory </var/WebSphere/home/FZCFG>
```

And that was the problem -- on our Sysplex, each LPAR has a system-specific `/var` directory. When we build the cell initially on `SYSC`, the `BBOSBRAM` job created the home directories on `/SYS`**`C`**`/var/...`, but when we tried to start the server on `SYSD` there was no home directory for `/SYS`**`D`**`/var/...`

The solution was simple -- we ran the `SBBOBRAM` job on `SYSD`. That created the necessary home directories. Had it been a different Sysplex, we could have shipped `SBBOBRAM` over there, or hand-created the directories and provided the proper ownership and permissions.

If you are performing a disaster recovery test and you bring a cell's configuration HFS over to a new Sysplex, you may not remember to create the group ID home directories. We wanted to show what happens so you could be aware of this potential problem.

Note: It *is* possible to create the group ID home directories *inside* the configuration HFS. No matter where the HFS goes, the group ID homes will be there. But that's not the default behavior.

Started DMGR on SYSD

We started the Deployment Manager again. This time it started successfully.

Invoked syncNode.sh to update files for application server node

With the DMGR up and running, we were now able to use `syncNode.sh` to get the changes made to the master configuration synchronized out to the node itself. Remember -- the changes we made with the AdminTask function were done to the master configuration only, *not* the XML files under the `/AppServer` directory of the node.

Here's the process we used to run `syncNode.sh`:

- We changed directories to the `/profiles/default/bin` directory of the federated node. This was under:

  ```
  /wasv61config/fzcell/fznodec/AppServer/profiles/default/bin
  ```

- We made sure we were still operating under the Admin ID. This is particularly important when invoking `syncNode.sh` when global security is enabled. To establish the SSL connection back to the DMGR we needed access to a keyring with the right certificates. The Admin ID by default has all that stuff.
- We checked to see what the SOAP port of the DMGR was -- it was 28810.

  Note: `syncNode.sh` establishes a TCP connection to the SOAP port of the running DMGR. That's what allows it to pull the changed XML files down and synchronize the node.

- We invoked `syncNode.sh` with the following command:

```
 1              2              3
./syncNode.sh wsc4.washington.ibm.com 28810

     -username FZADMIN -password XXXXXX
               4              5
```

Command used to invoke syncNode.sh

Notes:

1. The shell script itself
2. The host name where the running DMGR was to be found. We'd just started it on the SYSD system, which was `wsc4.washington.ibm.com`.
3. The SOAP port of the running DMGR
4. Global security was enabled this cell, so a userid and password was required to authenticate into the SOAP port

- It reported back the following:

```
ADMU0402I: The configuration for node fznodec has been synchronized
with Deployment Manager wsc4.washington.ibm.com: 28810
```

The node was synchronized; the changes were now in the node's configuration as well.

Started Node Agent and application servers

We started the Node Agent and the application servers. The group ID home directory issue reported earlier was not a problem since we'd corrected it for the DMGR.

Alternative - WSADMIN against application server node; no syncNode.sh

We just illustrated using `syncNode.sh` to synchronize changes from the DMGR's master configuration out to the node. But there is a way to change the host names and system names of the federated node without `syncNode.sh` -- it simply means running WSADMIN AdminTask commands against the node *in addition* to against the DMGR:

Running AdminTask against both nodes; no syncNode.sh needed

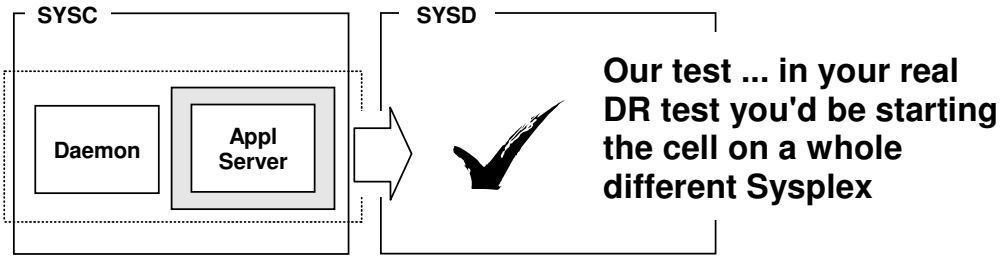
Notes:

- It is *critical* that you invoke the copy of `wsadmin.sh` that's found in each node's configuration structure.
- Invoking the `modifyNodeGroupProperty` to change the Daemon instance name for the federated node is not absolutely necessary, depending on the sequence you use to start the servers. But it's the safe thing to do, just to be sure.

Illustration of Standalone Server cell

Note: This process was very similar to the ND cell process, but not exactly.

Like for the ND illustration, we simply moved the cell from one LPAR to another:



The "Disaster Recovery test" scenario used for the Standalone Server illustration

Here was our "before" and "after" information:

	Before	*After*
Host Name	`wsc3.washington.ibm.com`	`wsc4.washington.ibm.com`
System Name	`SYSC`	`SYSD`

Stopped all servers in Standalone cell

The WSADMIN command is capable of making the XML changes with the servers up, but for the changes to take effect the servers would need to be stopped and restarted. So we'll illustrate making the changes while the servers were down.

Invoked WSADMIN client

The WSADMIN client is the `wsadmin.sh` shell script, which was located at the following location:

```
                1                2                3               4
/wasv61config/fzcell/fznodec/AppServer/profiles/default/bin/wsadmin.sh
```

Location of WSADMIN client shell script

Notes:

1. The mount point we had for our configuration HFS
2. The default home directory for the node, `/AppServer`
3. WebSphere makes use of "profiles," which are different configurations. For z/OS, we only have one profile possible, and it'll always be `/default`. It's important to drive down into the `/profiles/default/bin` directory, and not the higher `/bin` directory.
4. The WSADMIN client shell script itself. This is what we invoked.

Here's what we did to invoke WSADMIN:

```
su fzadmin  1

     2          3
./wsadmin.sh -conntype NONE
```

Commands used to invoke WSADMIN client

Notes:

1. It's always best to run WSADMIN under the authority of the Admin ID. That insures access to all the necessary directories.
2. The shell script itself
3. The parameter `-conntype NONE` specifies that WSADMIN should operate in "local" mode; that is, no connection to a running server and all changes are made directly to the configuration HFS.

??? How did it know which configuration to operate against? It is based on which copy of `wsadmin.sh` is invoked. Each node has its own copy. With `-conntype NONE`, WSADMIN assumes it is to operate against the same node from which it was invoked.

The result we saw was:

```
WASX7357I: By request, this scripting client is not connected to any
server process. Certain configuration and application operations will 1
be available in local mode.

WASX7029I: For help, enter: "$Help help" 2

wsadmin> 3
```

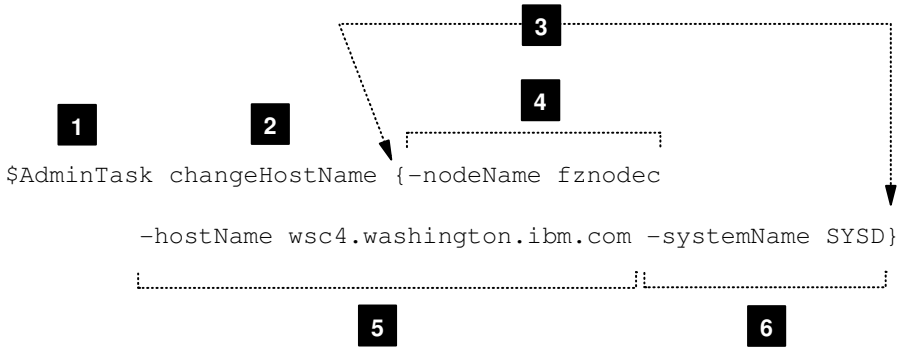
WSADMIN command prompt

Notes:

1. Message indicating that not all WSADMIN commands will work. When in local mode, only those involving changes to configuration files work, but things like starting and stopping servers will not.
2. Message pointing to the $Help object, which we'll not cover in this document.
3. The command prompt. It was here we entered our commands.

Modified host name and system name for server

We were now ready to change the server node's host name and system name. The command we entered was:



Command entered to change host and system names

Important! The command is entered as *one line*. We broke it here because of space limitations.

Notes:

1. The WSADMIN object that implements the function we wanted. The dollar sign out front tells WSADMIN that the scripting type is JACL.
2. The method on `AdminTask` that is used to change host name and system name
3. Notice the open brace and close brace ... this is what WSADMIN uses to "bracket" the parameters to the `changeHostName` method. If you forget a brace, WSADMIN will let you know that it had trouble "evaluating the JACL expression."

4. The `-nodeName` parameter names the node *long* name that will be the target of the changes. A Standalone server has only one node, but this parameter is required nevertheless.
5. The `-hostName` parameter provides the *new* host name. Notice that there is no checking for what the existing host name is. This is *not* a "find and replace" operation. This will replace whatever is there with this value.
6. The `-systemName` parameter is what's used to provide the new system name. If by some luck the system name does not need changing, you may omit this parameter and its value. In our case it needed changing because the system name was different.

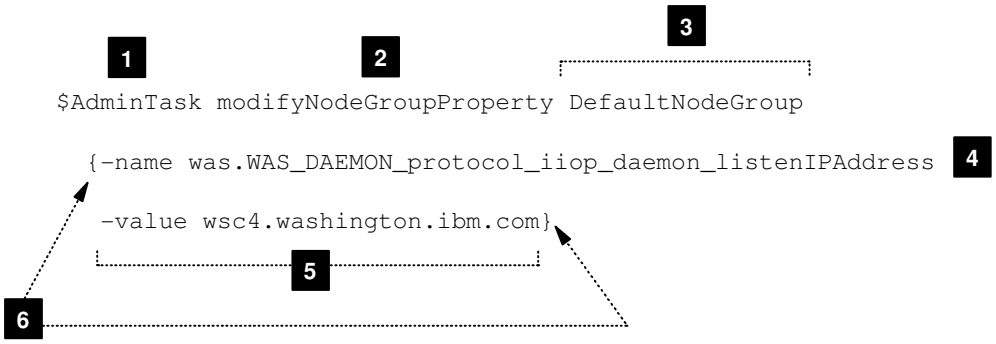
This function provides no positive feedback ... it simply comes back with the command prompt if things worked okay.

Important! The changes were in a temporary location at this point. We needed to save the changes, which we did with the `$AdminConfig` object we show in a moment.

Modified host name for Daemon instance

Daemon servers also make use of a host name value. That information is kept in the "node group" information for the cell.

The command we entered to modify this value was:



Command entered to change Daemon instance host name

Important! The command is entered as *one line*. We broke it here because of space limitations.

Notes:

1. The WSADMIN object that implements the function we wanted. The dollar sign out front tells WSADMIN that the scripting type is JACL.
2. The method on `AdminTask` that is used to modify the Daemon instance information in the "node group" information. It is not important that you know what a "node group" is.
3. The name of the node group that was modified. By default the name will be `DefaultNodeGroup`. Assume the default name unless you know for certain it's something else.
4. The `-name` parameter tells what property to modify. This big long value is a keyword string ... the Daemon instance host name is named on this property in the node group XML file.
5. The `-value` parameter provides the *new* host name. Notice that there is no checking for what the existing host name is. This is *not* a "find and replace" operation. This will replace whatever is there with this value.
6. The braces that enclose the parameter list.

Important! The changes were in a temporary location at this point. We needed to save the changes, which we did with the `$AdminConfig` object we show in a moment.

This command *did* report back success:

```
was.WAS_DAEMON_protocol_iiop_daemon_listenIPAddress
 (cells/fzbasec/nodegroups/DefaultNodeGroup|nodegroup.xml#Property_3)
```

Positive confirmation that Daemon property was changed.

Note: The value you see may be slightly different from this.

Saved configuration changes

A very simple command accomplished this:

```
$AdminConfig save
```

Notice that we made use of a different WSADMIN object -- `AdminConfig`.

No positive confirmation returned. It simply offered back the command prompt.

Exited WSADMIN

This was done with the `quit` command.

Started Standalone server on SYSD -- and it failed!

Note: You may not run into this. We're showing this because it highlights a subtle "gotcha" that happened to us. And it happened because of a change in the way the userid "home" directories were created in V6 and 6.1 vs. earlier versions of WebSphere.

When we tried to start the Standalone Server, it very quickly failed with very little information as to why. The job output simply said:

```
FZSR01C - STEP WAS EXECUTED - COND CODE 4095
FZSR01C FZSR01C - STEP WAS NOT RUN BECAUSE OF CONDITIONAL EXPRESSION ON STATEMENT 9
```

The step that failed with RC=4095 was one that runs a test to see if OMVS can successfully launch a shell and return. The output from that test went to:

```
/wasv61config/fzcell/fznodec/AppServer/properties/service/logs/applyPTF.out
```

In that file we found:

```
FSUM1004 Cannot change to directory </var/WebSphere/home/FZCFG>
```

And that was the problem -- on our Sysplex, each LPAR has a system-specific `/var` directory. When we build the cell initially on `SYSC`, the `BBOSBRAM` job created the home directories on `/SYS`**`C`**`/var/...`, but when we tried to start the server on `SYSD` there was no home directory for `/SYS`**`D`**`/var/...`

The solution was simple -- we ran the `SBBOBRAM` job on `SYSD`. That created the necessary home directories. Had it been a different Sysplex, we could have shipped `SBBOBRAM` over there, or hand-created the directories and provided the proper ownership and permissions.

If you are performing a disaster recovery test and you bring a cell's configuration HFS over to a new Sysplex, you may not remember to create the group ID home directories. We wanted to show what happens so you could be aware of this potential problem.

Note: It *is* possible to create the group ID home directories *inside* the configuration HFS. No matter where the HFS goes, the group ID homes will be there. But that's not the default behavior.

Started Standalone server on SYSD

We started the Standalone Server again. This time it started successfully.

Miscellaneous Information

A unique case -- moving a node but leaving the DMGR where it was configured

In the two scenarios we illustrated earlier in this paper the whole cell was moving from one LPAR to another. Scenario 1 was a simple ND node moving; Scenario 2 was a Standalone server moving.

There's another scenario that's possible -- an ND cell where the federated node moves but the DMGR stays where it is. That implies the use of the `changeHostName` command but not the use of the `modifyNodeGroupProperty DefaultNodeGroup` command.

Why? Because if the DMGR stays where it is, then the DMGR's Daemon stays where it is, and that Daemon's host configuration is the one used for the whole cell (when a single, default node group is present, as is true of most cells).

So the rules of thumb are these:

- If the DMGR *and* a federated node are moving (Scenario 1) then change the Daemon listen IP address value. [4]
- If only the node is moving, but the DMGR stays where it is, then do not change the Daemon listen IP address value.

The Sysplex name in the WebSphere configuration XML

In WebSphere for z/OS V5.x and V6.0, the Sysplex name was not kept in the configuration XML. Moving a configuration from one sysplex to another involved changing host IP names and system names, but not the Sysplex name. It wasn't present.

In V6.1 the Sysplex name was introduced to the configuration XML as well as one non-XML properties file. The use of the Sysplex name is related to the node group [5]. By default there will be one and it'll be called `DefaultNodeGroup`. Most people operate with just that one default node group.

Does that need to change when the cell is moved to another Sysplex?

In most cases you could operate with a mis-match between the configured Sysplex name and the actual Sysplex name and you'd probably never know it. But there are cases where it could become a problem.

The recommendation is to make sure the configured Sysplex name matches actual -- it will avoid confusion later if someone notices it and starts asking questions, and it avoids the potential for some subtle problem to crop up later.

Where the Sysplex name is maintained in the configuration files

It's maintained in two different files:

- `nodegroup.xml` (XML file)
- `node-metadata.properties` (non-XML file)

The `nodegroup.xml` file can be updated with an AdminTask command, just like the system name and IP host name can. That makes insuring the update is made in all the right places must easier. The `node-metadata.properties` file is automatically updated at node startup, so you don't need to worry about that.

---

[4] Unless you've followed the instructions in the WP101140 Techdoc and made the DMGR "mobile" by hiding it behind a DVIPA with Sysplex Distributor. If that's the case, then do not change the Daemon IP listen name since that's configured as the DVIPA and that stays the same regardless of where the DMGR resides.

[5] A node group is a collection of nodes WebSphere considers to have similar characteristics. It uses node groups to understand where the boundary is for things like clustering. By default an all-z/OS cell will have one node group -- `DefaultNodeGroup`, and all the nodes in the cell will belong to that node group.

The AdminTask command to change the configured Sysplex value

The following commands will change the configured value:

```
AdminTask.modifyNodeGroupProperty('DefaultNodeGroup',
                                  '[-name was.sysplexName -value WSCPLEX]')
AdminConfig.save()
```

Note: It's first is a one line command, but we show it here split so the font isn't too small to read.

Notes:

- Change the `DefaultNodeGroup` value to the name of your node group if it's different. But chances are very good that your node group name is `DefaultNodeGroup`.
- Change the `-value WSCPLEX` string to whatever the target Sysplex name is.
- *Synchronize to the nodes so their copies are updated with the new value!*

Example of submitting from JCL batch

If you wish to submit the WSADMIN from a JCL batch job rather than real time from a telnet command prompt, that can be done using BPXBATCH.

```
=COLS> ----+----1----+----2----+----3----+----4----+----5----+
****** ***************************** Top of Data *************
//WSADMIN JOB (ACCTNO,ROOM),REGION=0M,
// USER=MYADMIN,PASSWORD=*******
//STEP1    EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 BPXBATCH SH +
  /wasv61config/mycell/mydmnode/DeploymentManager+          ⇦ Note #1
  /profiles/default/bin/wsadmin.sh +                        ⇦ Note #2
  -conntype NONE +
  -c '$AdminTask changeHostName +
  {-nodeName wpdmnode +
   -hostName wsc4.washington.ibm.com +
   -systemName SYSD}'
 1> /tmp/wsadmin.out +
 2> /tmp/wsadmin.err
/*
```

Notes:

1. There's no space before the + sign because we need to concatenate the next line *directly* onto the first line to form up the directory name where `wsadmin.sh` is found.
2. There *is* a blank before the + sign here, and on the other lines as well, because here we want a blank included in the concatenation.

Document Change History

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

July 7, 2006	Original document.
July 10, 2006	Document number WP100792 added to the document and republished.
August 13, 2006	See "A less common Daemon change" on page 12.
September 20, 2007	Fixed a small typo on one of the command examples. An important "dash" was missing before the operator `hostName`. Also added an example of invoking the command from batch JCL using BPXBATCH.
July 9, 2008	Added a small section on where the Sysplex name is kept and how to change it if needed. See "The Sysplex name in the WebSphere configuration XML" on page 19.
December 16, 2008	Added a quick note in the Executive Overview about how the length of these AdminTask commands might exceed the input field of OMVS.
January 22, 2009	Updated the section on changing the Sysplex name to include information on the value being found in the `node-metadata.properties` file and how to change it there as well. The AdminTask function will change the Sysplex name in the `nodegroup.xml` files, but it apparently does not change it in the properties file. Whether that is a defect is under investigation.
April 21, 2009	Removed the 22-Jan-2009 update on `node-metadata.properties` file. Testing with 6.1.0.23 and 7.0.0.3 showed the earlier problem resolved. That file in each node's configuration file system will be programmatically updated with the actual Sysplex name seen at startup. There's no longer any need to manually edit that properties file. The other XML files still need to be updated using the AdminTask function, as described in this paper.
April 22, 2009	Oops! ☺ Cleaned up a final remnant of the `node-metadata.properties` file issue.
June 26, 2010	Added a note on page 19 about a case where changing the Daemon IP listen address is *not* recommended.

End of Document WP100792