Many thanks to ... well, the list is long. From the Washington Systems Center: Mike Cox for the vision, Mike Kearney for all the security implications of this, as well as John Hutchinson, Lee-Win Tai, and Bob Teichman.

From WebSphere development: Rohith Ashok.
## Table of Contents

**Executive Overview** .......................................................... 3
  - Scope and objective of this document .................................. 3
  - Keys to making this work .................................................. 3
  - Planning ................................................................................. 4
  - Service levels ....................................................................... 4
  - Security ................................................................................ 4
  - "RACF" = SAF for this document ........................................... 5
  - Miscellaneous things ........................................................... 5
  - Minimum service level recommended ..................................... 5
  - Disclaimer ............................................................................. 6

**Revisiting Important WebSphere Configuration Points** ............. 7
  - Standalone servers and Network Deployment configurations ....... 7
  - The Deployment Manager and the cell .................................... 7
  - Names and the naming convention ........................................ 8
    - Meaningful naming convention .......................................... 9
      - Where our earlier short name convention starts to break down 10
    - Insuring unique names ..................................................... 10
  - Building nodes and adding nodes to the cell ......................... 11
  - New in Version 6 -- the "node group" .................................... 12
    - Node Group long name .................................................... 13
    - Node Group short name .................................................. 13
    - Summary of Node Groups ............................................... 14
  - Implications of turning on global security .............................. 14
    - External user registry -- LDAP ........................................ 14
      - Configuring an LDAP server .......................................... 14
      - Loading information into LDAP server .............................. 14
      - Configuring the cell's security settings to use LDAP as user registry 15
      - Combination of local OS repositories and LDAP user registry 15
    - J2EE role enforcement .................................................... 15
    - Certificates, Certificate Authorities, digital signatures, keyrings, keyfiles ... 17
      - When the cell is all-z/OS .............................................. 19
      - When the cell is heterogeneous ....................................... 19
      - Who plays the role of Certificate Authority? .................... 20
      - Where the CA resides .................................................. 20
      - Where are the server certificates generated and where are they signed? 21
      - Copying CA public/private key to another system not a good idea 22
      - When two separate Sysplex environments are part of the heterogeneous cell 22
  - Illustration of a Test Cell Built at the Washington Systems Center 25
    - Sequence of construction ............................................... 25
    - Standalone server on WSCPLEX ........................................ 26
    - DeploymentManager on xLinux ......................................... 27
      - ORB character encoding ................................................ 27
    - Federated z/OS node into xLinux DMGR ............................... 28
    - Prepared for global security ............................................. 29
      - Planned the distribution of server and CA certificates ........ 29
      - Generated server certificate for Deployment Manager .......... 32
      - Export certificates from z/OS ........................................... 33
        - Exporting the CA certificate ........................................ 33
        - Exporting the server certificate and private key ............... 33
        - FTPped files to the xLinux box where DMGR was defined .... 34
      - Imported downloaded certificates into new keyfiles on xLinux system 34

© 2007, IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

Version Date: Wednesday, September 12, 2007
Where does the value of the realm name come from?

What is the realm name?

Status of cell

Federate z/OS node into xLinux DMGR

Started Standalone Server

Set final few properties

Important: verified node and Node Agent operational before saving settings

Configured cell to use LDAP as user registry

Turning on global security

Generated LTPA keys

Important: verified node and Node Agent operational before saving settings

Set final few properties

Stopped all servers and restarted

Status of the cell

Standalone server on WPLEX

Preliminary planning

Would RRSF have made this easier?

Copied Security Domain and Standalone Server saved variables to WPLEX

Coordinated CA certificate and server certificate

Exported CA certificate and server certificate

Imported CA certificate

What about server certificate?

ISPF panels and the generated jobs

Security Domain panels and jobs

Standalone Server panels and jobs -- generated but not run

Manually created controller ID and connected to configuration group

Imported server certificate

Modified BBOWBRAK member

Run the rest of the customized jobs to create Standalone Server

Started Standalone Server

Federate z/OS node into xLinux DMGR

Status of cell

Appendix A - Miscellaneous Updates and Information

Notes on cross-Sysplex cells and SAF REALM names in 6.1

What is the realm name?

Where does the value of the realm name come from?

Considerations
Executive Overview

When WebSphere Application Server V5 for z/OS came out back in 2004, it had one notable restriction: the cell had to exist entirely on z/OS. Further, it had to exist entirely within the same Sysplex.

With WebSphere for z/OS Version 6 that restriction no longer exists\(^1\). Now a WebSphere cell can span Sysplex environments, and it can span operating systems and have nodes on z/OS, Linux, Unix and Windows. We'll call this kind of configuration a heterogeneous cell:

The interaction between the Deployment Manager and the nodes is done over a TCP network. Node Agents on the nodes serve on each node to make whatever configuration changes are made through the Admin Console on the DMGR. So the basic architecture was always there to support this. But some of the operational details weren't quite in place ... until now.

**Scope and objective of this document**

This document will convey essential concepts and key points about heterogeneous cells, and how one is built. The first section of the document covers some important high-level issues that we faced and issues you may face, and the second section covers the creation of an actual heterogeneous cell built at the IBM Washington Systems Center.

When writing a paper like this it's always difficult to know where to draw the line between conveying conceptual information and actual keystroke-level instructions. This paper won't attempt to do the detailed step-by-step instructions. We'll leave that level of detail to the product-specific documentation (for instance, the documentation for keytool, which is used to create and manage keyfiles and trustfiles on the distributed platform).

**Keys to making this work**

If we had to limit this list to just three elements, those three would be:

- Planning
- Service levels
- Security

---

\(^1\) Provided you're at the proper level of maintenance. See “Minimum service level recommended” on page 5.
Planning

Proper planning of a WebSphere Application Server cell was always a good idea, but with a heterogeneous cell like this the need for good planning is amplified. Two main areas of planning are needed:

1. **Naming** -- WebSphere imposes restrictions on duplicate names for some things. So building a heterogeneous cell involves a deliberate intent to do so. A cell that simply evolves over time, without much forethought, will probably be bad experience for you.

   **Key:** A good naming convention, properly planned and carefully followed will be essential. So too will be the carefully consideration of where certificates are generated and signed, and how they'll be propagated to the server system where they'll be used.

2. **Digital Certificates** -- for a heterogeneous cell to work with SSL enabled will require a coordinated set of certificates throughout the cell. When the cell is "all-z/OS" the customized jobs take care of much of this coordination. But when the cell jumps platforms it requires some manual planning and implementation.

This document covers both topics.

Service levels

The Deployment Manager node must be at a service level equal to or higher than any of the application server nodes.

**Note:** If a node is at a higher level than the DMGR node, the Node Agent will detect this and fail to initialize.

That's the case when the cell is on the same z/OS Sysplex, and is the case when the cell is across multiple platforms. When the cell is entirely on z/OS the coordination of the service level was relatively easy: service came out for WebSphere on z/OS and you simply made sure the service was applied to the Deployment Manager node first.

But when the cell is heterogeneous there's a complicating factor: service releases for z/OS and the distributed platforms do not necessarily come out at the same time. Thankfully the "service pack" number -- for instance, 6.0.1.2 -- is coordinated across all platforms. That means that 6.0.1.2 is the "same code level" on z/OS, Linux, AIX or Windows.

**Key:** Always remember this: apply service first to the DMGR, then the other nodes. Never apply service to an application server node so that it becomes a higher level than the Deployment Manager.

Security

If global security is turned off, the act of building a heterogeneous cell is fairly simple. Federating a node into a Deployment Manager cell is essentially the same process, whether the node is the same type as the DMGR's or a different one. But when security is on, things get more interesting.

When a WebSphere cell is contained entirely within a z/OS Sysplex, we have the luxury of using the MVS "Security Access Facility" (SAF) as the common user registry across the cell\(^2\). All the nodes in that cell have access to SAF, and the SAF database is shared across the Sysplex:

---
\(^2\) Not to mention a common repository for digital certificates. You'll see as you read this document that planning and coordinating the digital certificates plays a significant role in the success of the final heterogeneous cell.
But when the cell is configured across *multiple operating system platforms*, then we no longer have that luxury. WebSphere running on a distributed platform can't access the SAF interface. We then must use some other user registry, one that's accessible to both z/OS nodes and distributed platform nodes. An LDAP directory is an example of an external user registry.

**Note:** This is true for cells that span Sysplex environments as well, unless the SAF repository on both Sysplexes had the *same information* about the cell ... IDs, keyrings, certificates, etc.

Once the configuration jumps platforms, several security-related things bubble to the surface. These are covered in more detail in "Implications of turning on global security" starting on page 14.

"**RACF**" = SAF for this document

The term RACF will be used in many places throughout this document as a generic reference to the product that implements security policies behind the SAF interface. Other products provide similar functionality. The use of the term "RACF" does not imply exclusively RACF ... it's just a simpler term than "SAF product functionality."

**Miscellaneous things**

In addition to the broader topics listed above, two rather arcane issues popped up in the building of our "proof of concept" cell at the Washington Systems Center:

- **ORB character encoding** -- it turned out the *orb.properties* file of the distributed platform WebSphere DMGR had to be modified so the ORB there could talk to the ORB on z/OS. See "ORB character encoding" on page 27 for more.

- **Unrestricted policy files** -- the *keytool* utility provided by WebSphere on the distributed platforms does not by default have the more powerful encryption files. That's due to export limitations on that technology. But you can get the files from the web. This issue came up because we exported a server certificate from RACF on z/OS using PKCS12 encryption and needed to give keytool on the distributed box the ability to crack open the private key file. See "Unrestricted policy files" on page 35 for more.

**Minimum service level recommended**

Version 6.0.2 is the minimum level that supports heterogeneous cells.
Disclaimer

I have attempted to capture the key considerations in building a heterogeneous cell. I've offered some examples of commands and jobs used at the Washington Systems Center to provide some reality to the discussion. I believe I have captured everything accurately, but I can't be certain that you won't run into different things when building a heterogeneous cell. To the extent inaccuracies exist in this document, the responsibility falls on me rather than those who provided assistance to me in writing this document.

Please send corrections to me at dbagwell@us.ibm.com
Revisiting Important WebSphere Configuration Points

You may already be aware of some of these points, but reiterating them here sets a common base for the discussion and understanding of the issues surrounding the construction of a heterogeneous cell.

**Standalone servers and Network Deployment configurations**

WebSphere for z/OS Version 6 has two basic configuration forms, regardless of the platform on which WebSphere is running:

- Standalone Server configuration
- Network Deployment configuration

In some ways the difference between them is simply a matter of degree:

<table>
<thead>
<tr>
<th><strong>Standalone Server</strong></th>
<th><strong>Network Deployment</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Daemon</td>
<td>Daemon</td>
</tr>
<tr>
<td>AppServer</td>
<td>AppServer</td>
</tr>
<tr>
<td>CR</td>
<td>CR</td>
</tr>
<tr>
<td>CR</td>
<td>CR</td>
</tr>
<tr>
<td>CR</td>
<td>SR</td>
</tr>
</tbody>
</table>

*Standalone server as compared to the Network Deployment configuration -- z/OS example*

<table>
<thead>
<tr>
<th><strong>Similarities</strong></th>
<th><strong>Differences</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Both consist of a cell and nodes</td>
<td>Standalone Server is limited to a single server while a Network Deployment configuration may have many servers</td>
</tr>
<tr>
<td>Both have a Daemon</td>
<td>Standalone Server is limited to a single node while a Network Deployment configuration may have many nodes</td>
</tr>
<tr>
<td>Both run the Admin Console</td>
<td>Standalone Server must by definition reside entirely on a single operating system instance while a Network Deployment configuration may span many</td>
</tr>
<tr>
<td></td>
<td>Can't cluster servers in a Standalone Server while clustering is supported in a Network Deployment Configuration</td>
</tr>
</tbody>
</table>

The list item under "differences" marked with a small star emblem is the key to understanding why only Network Deployment configurations may be considered for a heterogeneous cell. That's the only configuration type that may span operating system instances.

**Notes:**

- Why the term "operating system instance?" Terms like "servers" or "boxes" get a bit ambiguous in this age of partitions and virtualized systems. But "operating system instance" is more precise in telling the story of where WebSphere runs.
- The ability to split a ND configuration across multiple operating system instances is done at the node level. A node must, by definition, be contained within a single operating system. That's why a Standalone Server -- which consists of one and only one node -- can't span multiple operating system instances.

**The Deployment Manager and the cell**

The Deployment Manager (DMGR) serves as the central repository of information about the cell as a whole. It is where the administrative application runs (the "Admin Console"), and it is where the "master configuration" resides. Fortunately, the DMGR is not a single point of failure.
The steady-state function of the individual nodes and servers is largely unaffected by the absence of the Deployment Manager.

**Note:** Some things, like performance monitoring, requires the DMGR to be up and running. But almost all other things can continue on even if the DMGR is stopped. Configuration changes are generally restricted when the DMGR is down, but not entirely so -- the scripting tool WSADMIN could be used to make changes to the configuration even if the DMGR is not present.

There really isn't anything in the architecture of WebSphere that says that the Deployment Manager has to be on the same server or even the same platform type as the application server nodes. Prior to V6 the restriction found with WebSphere on z/OS was largely a tactical issue having to do specific technical difficulties that hadn't yet been ironed out.

So with V6 there are now many different topologies that are possible:

---

**Note:** And combinations of these. The example scenario shown under "Illustration of a Test Cell Built at the Washington Systems Center" starting on page 25 is a combination of the multi-Sysplex topology (upper-right corner) and the DMGR on distributed topology.

**Names and the naming convention**

If you're familiar with the importance of a good naming convention for an all-z/OS configuration, you'll not be surprised to hear it's as important -- if not more important -- to have a good naming convention planned in advance before attempting a heterogeneous cell.
There are really two issues here:

1. Coming up with a naming convention that meaningfully captures all the information you want to convey in the names
2. Making sure you don’t duplicate names that WebSphere requires be unique

**Meaningful naming convention**

WebSphere has a lot of things that require names -- cells, nodes, servers, applications, resources ... the list goes on and on. Furthermore, WebSphere for z/OS has all the same names as distributed WebSphere plus a set of "short" names.

*Note:* "Short" names are a part of WebSphere for z/OS because the underlying operating system has length limitations for certain things. Chief among these are the MVS JOBNAME, JCL start procedure names and the parameter string passed in on the MVS START command.

While it’s possible to put together a WebSphere configuration without planning the names in advance, it gets more difficult the more complex the configuration. It is recommended that a naming convention is planned out prior to building a configuration, and particularly a configuration that spans different platforms.

When considering the naming convention, the z/OS "short" names pose a particular challenge. They are typically limited to either 7 or 8 characters. It becomes difficult to convey a lot of information in so few characters. That makes each character valuable -- devoting too much of the limited characters to some piece of information will severely limit you elsewhere.

*Note:* Long names are less of a concern -- they are not limited to 7 or 8 characters, so it’s far easier to convey information in the name with that many characters to work with.

What information is important to include in the short name convention? Opinions differ, but there seems to be a consensus on a few key things:

- All short names for a cell should start with the same characters. This provides a way to relate all the names within a cell to one another. The number of characters used for this purpose is up to you, though most naming convention examples use two characters for this purpose.
- Some indication of the type of server the name represents -- Daemon server, Deployment Manager, Node Agent or Application Server.
- A “system identifier” to indicate which MVS image a server is running on. This is particularly useful for Node Agents and servers that are clustered.

The naming convention for the server JOBNAME, advocated in the WP100367 white paper and used in the spreadsheet planning tool provided in the PRS1331 techdoc, is this:

```
1 2 3 4 5 6 7
M Y S R 0 1 A
```

- **Cell identifier**
- **Server Identifier**
  - DEMN = Daemon
  - DMGR = Deployment Manager
  - AGNT = Node Agents
  - SR## = Application Servers, where ## is some numeric differentiator
- **System Identifier**

An example of a naming convention for the WebSphere for z/OS JOBNAME
Notes:

- The eighth character is reserved for WLM, which will append an "S" for servant regions and an "A" for controller adjunct regions.
- The white paper and spreadsheet referenced earlier can be found at:
  
  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100367
  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS1331

Some feel that the name should include other things, though there's some disagreement over the importance of these:

- A "W" to indicate "WebSphere"
- A number to indicate the version of WebSphere
- A indication of the application running in the server. This is usually only in cases where each server will host a single application.

The key here is not to say that one naming convention is clearly superior to another, but rather that a well-thought-out naming convention is important. In coming up with a naming convention, it's wise to "paper test" the convention by coming up with different scenarios and seeing if the naming convention can handle it. No naming convention is perfect -- each will have limitations. It's best to know the limitations up-front, rather than after the fact.

Where our earlier short name convention starts to break down

As mentioned earlier, WebSphere for z/OS Version 5 was limited to a single Sysplex environment. The naming convention we came up with for V5 never bothered to include a "Sysplex Identifier." It had a single-character "System Identifier," but assumed (correctly for V5) that the cell would be contained within the same Sysplex.

But with V6, a cell can span Sysplexes. So a decision needs to be made whether the Sysplex on which the server runs should be part of the name, and if so, how that information will be provided.

Note: When a cell spans over to non-z/OS platforms, those platforms won't use short names. Short names are only found on z/OS. Therefore, the distributed platform boxes will have only long names. Long names can be much longer than 8 characters, so there's ample space to include some indication of where the node or server is running.

There are at least a couple of ways to make room for a Sysplex identifier:

- Reduce the "server identifier" from four characters to three, and use the extra character for the Sysplex Identifier
- Reduce the "cell identifier" from two characters to one

Or you may simply decide to forego having a "Sysplex Identifier."

Insuring unique names

Important: WebSphere requires that some names be unique. This includes short names as well as long names. Some names must be unique within the cell, other names must be unique between cells. To insure uniqueness between cells, it's best to start all names with a "cell identifier" and provide each cell with a unique set of characters.

What follows is a discussion about how strict that uniqueness requirement appears to be. This discussion is offered only for cases where you're forced to re-use a name for some reason. But again, strive for unique names within and between cells.
The following table summarizes the uniqueness requirements for the names:

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Uniqueness Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Long</td>
<td>All</td>
<td>Unique within the Sysplex.</td>
</tr>
<tr>
<td>Cell Short</td>
<td>z/OS</td>
<td>Unique within the Sysplex.</td>
</tr>
<tr>
<td>Node Long</td>
<td>All</td>
<td>Unique within the cell.</td>
</tr>
<tr>
<td>Node Short</td>
<td>z/OS</td>
<td>Unique within the cell.</td>
</tr>
<tr>
<td>Node Group Long Note</td>
<td>All</td>
<td>Unique within the cell.</td>
</tr>
<tr>
<td>Node Group Short Note</td>
<td>z/OS</td>
<td>Unique within the cell.</td>
</tr>
<tr>
<td>Server Long</td>
<td>All</td>
<td>Unique within the node.</td>
</tr>
<tr>
<td>Server Short</td>
<td>z/OS</td>
<td>Unique within the cell.</td>
</tr>
<tr>
<td>Cluster Transition Name</td>
<td>z/OS</td>
<td>Unique within the Sysplex.</td>
</tr>
</tbody>
</table>

**Notes:**
- This table is built on what I believe is correct information. When building a heterogeneous cell, you should strive to insure unique names. This table is meant to provide information when you're faced with having duplicate names. Again, there may be more of a restriction imposed by WebSphere than is represented here.
- See "New in Version 6 -- the "node group"" starting on page 12 for more information on "Node Groups."

**Building nodes and adding nodes to the cell**

The method used to construct a node for a heterogeneous cell is no different from the method used if the cell was going to be homogeneous. And the method used to add the node is no different. This is all due to the inherent distributed architecture of WebSphere, where the interaction between Deployment Manager and nodes is across an IP network, and "end points" are defined as host/port pairs. It doesn't matter whether the host/port is on the same server or in the next room, on the same type of server or a different type of server-- as long as it can be reached via the IP network.

**Note:** That statement is quite supportable if global security is turned off. But things get more complicated when global security is turned on. When that's the case, some degree of coordination is required above and beyond the simple ability to reach the port over the IP network. Specifically, the coordination of keyrings and CA certificates is necessary, as well as the use of some kind of "external" user registry such as LDAP. More on this coming up.

So, for example, if a heterogeneous cell's Deployment Manager is on a distributed server and you wish to build a z/OS application server node, you would do it in the same way you would for an all-z/OS solution: run through the ISPF panels, generate the customized jobs, create the node, then federate the node into the running Deployment Manager. You can federate the node with either the customized JCL, the `addNode.sh` shell script, or do it from the DMGR's Admin Console.

For nodes on distributed platforms being federated into a z/OS DMGR, you would construct the node through the normal WebSphere install-shell process. Then you could federate into the DMGR with `addNode.sh`, or you could federate from the DMGR's Admin Console.

The key message here is that with the exception of the security implications, the rest of this is pretty much business as usual.
New in Version 6 -- the "node group"

Note: This topic is really somewhat of a "gee whiz" thing. It's not central to main thrust of this paper -- security definitions are the main thrust -- but it's something new with V6 and therefore we felt you may wish to know what "node groups" are all about.

A "node group" is a logical collection of nodes. WebSphere uses "node groups" to understand the limits of where clusters can be built. In WebSphere V6 -- for all platforms -- clusters may only be built across nodes that are part of the same "node group."

From a z/OS perspective, the key is this: a WebSphere for z/OS "node group" must be contained within a given Sysplex. If a cell is built across two Sysplexes, it will necessarily mean two node groups: one node group on the first Sysplex and a different node group on the second Sysplex:

When cell spans multiple Sysplex it means separate "node groups"

What that means is that a cluster of servers on WebSphere for z/OS must be contained within the boundaries of the Sysplex. A cluster can't span from one Sysplex to another, and it can't span from a z/OS platform to a non-z/OS platform.

You may not even be familiar with "node groups." That's because in WebSphere for z/OS Version 6, a default node group is automatically built when you construct a Network Deployment configuration. That default node group is called DefaultNodeGroup.

Note: Further, the initial default node group will be known as the "Sysplex Node Group," and it's this node group to which the Daemon structure is related. You may add additional node groups if you wish, but only one -- the first one -- will be the "Sysplex Node Group." You can tell which one is the "Sysplex Node Group" by looking at the Admin Console information for the Node Group: "System Administration" ⇒ "Node Groups." The "Sysplex Node Group" will be the one with the "Sysplex Name" information box.

Knowing this is not really critical to the task of building a heterogeneous cell. Just FYI.

When you build a Network Deployment configuration on a single Sysplex environment, and you federate a node into the Deployment Manager cell, by default the node that's joining is added to the DefaultNodeGroup. This is a good design ... it means that by default a cluster can be built to span any nodes within a Network Deployment configuration that's built on a single Sysplex environment.

But what about a cell that spans multiple platforms, including z/OS? Or, to make things really interesting, a cell that spans multiple Sysplexes? We saw in the table under "Insuring unique
names” starting on page 10 that a Node Group long name must be unique within a cell, as should the Node Group short name. What's the story behind those?

**Node Group long name**

Two nodes on two different Sysplexes will probably each use DefaultNodeGroup for its respective node group long name. How does WebSphere insure uniqueness when those two nodes are federated into the same cell? It renames the DefaultNodeGroup. And it uses the Sysplex name as a root for the new node group name:

<table>
<thead>
<tr>
<th>Before federation</th>
<th>After federation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Group: DefaultNodeGroup</td>
<td>Node Group: WSCPLEX_NodeGroup</td>
</tr>
<tr>
<td>Sysplex: WSCPLEX</td>
<td>Sysplex: WSCPLEX</td>
</tr>
</tbody>
</table>

How DefaultNodeGroup name is handled when two nodes from different Sysplexes federated

**Node Group short name**

Only Node Groups on z/OS will have short names. Node Group short names are not something you specify -- they're something that's created for you by default. And the default short name for the initial node group will be set equal to the cell short name.

Here's where things get interesting. Recall how an application server node is first built on z/OS: it starts out as a "Standalone Server" and then it is federated into the Deployment Manager cell. When a Standalone Server is built, a cell long and short name is required. We often state the cell name given to a Standalone Server is temporary; that is, it is used only up until the node is federated. Then the cell name used is the DMGR's cell name.

While that's true, it does not follow that the Standalone Server cell names simply vanish. The cell short name lives on ... in the form of the Node Group short name for the DefaultNodeGroup.  In the previous section we saw how WebSphere renames DefaultNodeGroup so the uniqueness of the node Node Group long name can be assured. What about the uniqueness of the short name?

That's responsibility falls on you.

Here's how you can be assured of unique Node Group short names when building a heterogeneous cell that involves nodes on different Sysplexes (as does our sample scenario):

> When building Standalone Servers that will be federated into the Deployment Manager cell, be sure to provide "temporary" cell short names that are unique.

**Note:** In other words, don't tag the temporary cell short name for a Standalone Server on Sysplex X with XXCELLT and the cell short on Sysplex Y as XXCELLT. That's the same value for both, and the default Node Group short name for each will be the same, which means when those Standalone Servers are federated into the same cell it'll violate the rule that they must be unique within the cell. Values of XXCELXX and XXCELY, for example, will provide uniqueness when the nodes are federated.

Can a Node Group's short name be changed after-the-fact? Not through the Admin Console. But it's a relatively easy thing to change using the WSADMIN scripting interface. Here’s the command you can use:

```bash
$AdminTask modifyNodeGroup DefaultNodeGroup { -shortName XXXXXX }
```

Where XXXXXX is the new short name for the node group. So if you find yourself having accidentally set the "temporary cell short name" to the same thing on different Sysplex environments, you should use this WSADMIN command to change the Node Group short name prior to federating the Standalone Server into the DMGR cell.
Summary of Node Groups

We provided this section on Node Groups for several reasons:

- To alert you to what a Node Group is. They're new in V6.
- To help you understand where Node Groups come from, and what happens to them when nodes from different Sysplex environments come into a single heterogeneous cell.
- To flag the important point about making sure Standalone Server temporary cell names are unique when they ultimately will be federated into a single DMGR cell.

Implications of turning on global security

This is the real "heart" of the issue of building heterogeneous cells. As mentioned earlier, if global security is turned off for the cell, then building the heterogeneous cell is really a matter of "business-as-usual" federation of nodes into a Deployment Manager cell, with some attention paid to uniqueness of names and the service levels of the DMGR and the various nodes.

But when that "big switch" of global security is turned on, then we must pay particular attention to several key things, most notably: an external user registry, the avoidance of SAF as the J2EE role enforcement mechanism, and the coordination of certificates between the various servers.

External user registry -- LDAP

When a cell is heterogeneous, an external user registry is required so a common repository for user and group information can be accessed by all servers in the cell. This is required so the servers can authenticate each other when they do the SSL handshake.

Note: As will be explained next, this is in addition to the local OS repository for user/group information.

This involves several steps:

- Configuring an LDAP server
- Loading the group and user information into the LDAP server
- Make the cell aware of the LDAP server (configure cell to use specific LDAP user registry)

Configuring an LDAP server

WebSphere recognizes seven different "flavors" of LDAP that might be used:

- IBM Tivoli Directory Server
- SecureWay
- SUN One
- Domino
- Active Directory
- eDirectory
- "Custom"

For installing and doing basic configuration of each, consult that product's documentation. What necessary is a working LDAP server that's on the network, accessible by other servers, and ready to accept the data about userids and groups.

Loading information into LDAP server

Once the LDAP server is ready to accept input, you should load information about all the userids for the cell as well as the "WebSphere Config Group".
This is a good spot to emphasize the importance of having a common "Config Group" for the entire cell -- z/OS nodes as well as non-z/OS nodes. So when configuring userids and groups on the various platforms, be sure to use a single, common, and consistently spelled name for the WebSphere configuration group.

The information can be supplied to LDAP in a batch format (using LDIF commands), or they can be entered individually through the command line interface, or you may use one of the several free LDAP browser/editors that exist.

Here's a brief example of how the input data might look:

Base distinguished name (suffix) is dc=fnf,dc=com

LDAP Group
dn: cn=wscfg1,dc=fnf,dc=com

LDAP Users (member of the above group)
dn: cn=wsadmin,dc=fnf,dc=com
uid: wsadmin
userpassword: wsadmin

In this example the WebSphere Configuration Group is wscfg1 and the admin ID wsadmin is a member of that group. In reality the number of users would be greater -- all the server IDs that would also be members of the wscfg1 group would be included as well.

Configuring the cell's security settings to use LDAP as user registry

The only way WebSphere will use the LDAP server as its user registry is if you tell it to. This is done through the Admin Console, under "Security." For an illustration of how this was done for the heterogeneous cell configured at the Washington Systems Center, see "Configured cell to use LDAP as user registry" starting on page 38.

Combination of local OS repositories and LDAP user registry

We've already stated that when a cell spans the z/OS platform and other platforms, an external user registry (such as LDAP) will be necessary. But having an external user registry does not eliminate the need to have local OS repositories as well. So the "BRAK" jobs that are run to create RACF profiles z/OS are still necessary:

- We still need RACF to understand the user and group structure so the identity of started tasks can be properly assigned, and access to the local file system can be controlled.
- We still need RACF to know how to assign identity to started tasks, which is done through STARTED profiles.
- We still need RACF to have properly defined CBIND and SERVER profiles.
- We still need RACF to have information on keyrings and certificates

So a heterogeneous cell will use a combination of local OS security information as well as an external user registry in LDAP. The customized JCL jobs (the "BRAK" jobs) that create security profiles for users and groups, STARTED profiles, CBIND and SERVER still need to be run on z/OS.

J2EE role enforcement

The J2EE application architecture provides something called "J2EE roles," which are levels of authority designed within an application. Not all applications make use of this feature. But some applications have a need to have some users with more authority than other users. An example of this is the hypothetical bank application where tellers are permitted...
access to a few functions, the teller supervisor to more functions, and the branch manager perhaps given access to still more.

**Note:** A "real life" example of this is the WebSphere Admin Console, which is a J2EE application. It has four "roles" defined: Administrator, Operator, Configurator and Monitor. Each role has different degrees of authority, with Administrator given full authority. By default the WebSphere Admin ID has Administrator authority.

When a user attempts to use a function of a J2EE application that has roles defined, there needs to be a check performed to see if the user has authority to perform the role. This is known as "role enforcement" and involves comparing the user's identity against a defined list of roles and who's permitted to do which role. The question boils down to this: where is that list maintained?

When WebSphere is on the z/OS platform, there are two choices: SAF (with RACF being IBM's product behind the SAF interface), or in the application's "bindings."

**Note:** An application's "bindings" are the set of XML files that are packaged with the application. These XML files tell the story about the application. WebSphere (or any J2EE server) uses the bindings to know how to install and operate the application. J2EE role definitions may be provided in these bindings.

But the distributed platform WebSphere has no such choice ... J2EE role enforcement on those platforms must be done through the application bindings.

Here's why this issue is being brought up: the EJB role enforcement option in z/OS is a cell-wide setting. If you are building a heterogeneous cell and you know that cell will have a combination of z/OS nodes and distributed platform nodes, you must construct your z/OS node so that EJB role enforcement is done through application bindings. EJB role enforcement must not be done through SAF in a heterogeneous cell.

Where is this property set for z/OS? It's defined in the "Security Domain." Specifically, panel 2 of 2 of the Security Domain definitions:

```
Security Domain Configuration (2 of 2)

Specify the following to customize the security domain to be selected when configuring one or more servers or cells, then press Enter to continue.

SSL Customization

Certificate authority keylabel...........: WebSphereCA
Generate certificate authority (CA) certificate: Y
Expiration date for CA authority: 2010/12/31
Default RACF keyring name............: WASKeyring
Enable SSL on location service daemon: N

Additional z/OS Security Customization Options

Generate default RACF realm name: N
Default RACF realm name ....: WSCPLEX

Use SAF EJBRLE profiles to enforce J2EE roles: Y

Make certain this is set to "N". Make certain all nodes constructed for z/OS for this cell use this Security Domain.
```

The z/OS Security Domain panel 2 of 2, where the "J2EE role" enforcement option is set

With respect to the note in the little box in the picture, what the second sentence is getting at is this: when constructing a Network Deployment configuration on z/OS, the first thing built is the "Security Domain." This produces a batch RACF script, which creates the
profiles for the cell-wide things such as the Admin ID, the configuration group and, as this picture shows, the SSL customization and J2EE role setting.

When other nodes are built for this cell, the Security Domain "saved variables" are loaded into the ISPF panels for the new node. This is an important step because what it does is help insure consistency between all nodes for those key cell-wide things.

We'll finish with these key points:

- If a node will be part of a heterogeneous cell, make certain the Security Domain "Use SAF EJBROLE" question is set to "N"
- Use the saved variables for that Security Domain for all nodes you construct that will be part of the heterogeneous cell. This would include application server nodes and a Deployment Manager node.

Certificates, Certificate Authorities, digital signatures, keyrings, keyfiles ...

Note: This is one of those topics that can become very complicated very quickly. But it's necessary to understand some the essential concepts of this to understand why certain things are done when constructing a heterogeneous cell.

When a cell is enabled for global security, one of the effects is to turn on the use of SSL as a mechanism to protect data traveling on the network. SSL stands for "Secure Socket Layer" and is a standard widely used to encrypt data at the TCP socket layer between client and server. You use SSL each day, no doubt: the little lock symbol on Internet Explorer means the requests flow over a protected SSL link.

There's no way to not use SSL when global security is enabled. The two -- global security and SSL -- are inseparable.

Further, in order for an SSL connection to be successfully established, the two sides of the communication need to do a little "hand-shaking" to make sure the other can be trusted. This is done by exchanging "certificates" -- long strings of digital data -- to see who the other party is and whether they can be trusted.

But a certificate by itself is not enough. To be trusted that certificate sent by a server or a client needs to be "signed" by a "Certificate Authority" (or a CA for short). What a Certificate Authority does by signing a server or client certificate is vouch for the authenticity of the original certificate.

Note: Here's where the topic could go deep into the encryption methods used when a CA signs a digital certificate. It involves the use of "public" and "private" keys and a very sophisticated encryption algorithm. We won't go into that level of detail.

For the recipient of a CA-signed certificate to really trust the CA's signature, that recipient needs to have a copy of the CA's "public key" available to it. The public key is used to decrypt the digital signature and verify that it was properly signed by the CA.

Note: And there is the fascinating aspect of asymmetrical encryption -- the digital signature is encrypted using the CA's "private" key, which they keep very private and jealously protect. The only way to decrypt the signature is with the CA's "public" key, which is distributes widely. Data encrypted with a private key can only be decrypted with the associated public key; data encrypted with the public key can only be decrypted with the private key. It's a remarkably elegant solution.
The word-picture just painted looks something like this:

The high-level relationship between a CA, server certificates and the CA's private/public keys

Notes:
1. The Certificate Authority uses its private key (a very carefully guarded string of digital data which only the CA has a copy of) to encrypt and "sign" the certificate for Server A.
2. When Server A wishes to establish an SSL connection with Server B, Server A sends its CA-signed server certificate over to Server B.
3. Server B receives A's certificate and uses the CA's public key to decrypt the signature so it can validate that the signature was indeed issued by CA XYZ.

Note: Please understand that the actual mechanics are far more sophisticated than what's pictured here. This is the high-level picture. It'll serve as the starting point for getting a few very key concepts on the table. Those concepts will be illustrated over and over in the description of the sample heterogeneous cell and how it was built.

As you look at that picture, consider the following very important points:

- **Certificate Authority** -- one must exist somewhere. There are two basic ways this can be accomplished: an external CA (like VeriSign) or an internal CA like RACF. We cover that in a bit more detail starting on page 20. The key point here is that you can't do SSL without the server certificates being signed by a CA. So a CA must exist somewhere in the picture.

- **Server Certificates** -- these identify the server and are what the CA signs. These need to be generated. Server certificates may be generated by RACF on z/OS, or keytool on the distributed platform. The key point here is that they need to be generated and signed by a CA. Server certificates reside in the "keyring" (a RACF thing on z/OS) or a "keyfile" (a file on a distributed platform).

- **CA signatures on server certificates** -- it's not enough to simply generate the server certificates. For SSL to work, they need to be signed by a Certificate Authority.

- **Public Key available to receiver of CA-signed certificate** -- the only way to crack open the CA's digital signature at the other end of an SSL connection is to have access to the CA's public key. Without that, the signed server certificate is useless. So when a decision is made regarding the CA to be used, that CA's public key should be made available in the keyrings (on z/OS) or "trustfiles" (on distributed) of all the other servers within the WebSphere cell.

Now let's turn to a closer look at some of the issues that bubble out of those four key points.
When the cell is all-z/OS

By default, the customized jobs for an all-z/OS WebSphere cell assumes that an internal CA will be employed. (See "Who plays the role of Certificate Authority?" on page 20 for more on that topic.)

The generation of the CA certificates and the various server certificates is all done within the various "BRAK" jobs that are run for each node. The "BRAK" jobs are generated by the ISPF customization dialogs and contain RACF commands such as these:

```racf
/* Creating SSL keyrings for WebSphere Deployment Manager */
RACDCERT ADDRING(WASKeyring) ID( ZZDCRU )
RACDCERT ADDRING(WASKeyring) ID( ZZDSRU )
/* Generating certificates for WebSphere Deployment Manager */
RACDCERT ID (ZZDCRU) GENCERT SUBJECTSDN(CN('ZZDCRU.ZZDMGR') O('IBM') OU('ZZ'))
   NOTAFTER(DATE(2010/12/31))
/* Connecting Certificates to the Deployment Manager keyring */
RACDCERT ID(ZZDCRU) CONNECT (LABEL('DefaultWASdmgrCert') RING(WASKeyring ) DEFAULT)
RACDCERT ID(ZZDSRU) CONNECT (RING(WASKeyring) LABEL('WebSphereCA.ZZ') CERTAUTH)
```

Here's what the numbered blocks are illustrating:

1. A keyring is built for the controller ID and the servant ID
2. A server certificate is built and signed with the CA certificate
3. The server certificate is added (or "connected") to the controller keyring
4. The CA certificate is connected to the controller and servant keyrings

The CA certificate was created in a different BRAK job -- the one for the Security Domain. That RACF command is not shown in the picture above.

In short, the customized BRAK jobs take care of everything based on an assumption of an internal CA.

When the cell is heterogeneous

What would happen if you copied these BRAK jobs to a distributed platform and tried to run them? It wouldn't work. RACF commands are not understood on those platforms. So when the cell is heterogeneous, we can't rely on some customized job to do all the work. We need to get more involved.
Who plays the role of Certificate Authority?

When thinking about the construction of a heterogeneous cell, one of the questions we need to iron out is who'll play the role of the Certificate Authority (CA). A CA is required, as we noted earlier.

There are two basic approaches:

1. Use an "external" (and well known) Certificate Authority such as VeriSign. This is the recommended approach for applications that will be used by people in the wider population at large.

   Why? Because well-known external authorities such as VeriSign have their CA certificates shipped with popular browsers such as Internet Explorer, Netscape and Firefox. It'll know to "trust" a server certificate signed by one of those well-known CA's. But, if the browser sees a server certificate signed by a CA it doesn't recognize (an "internal" CA, for instance), it'll pop a window asking the user (the human) to indicate their trust that it's okay to proceed.

2. Be your own "internal" CA and create your own CA certificates and use them to sign the server certificates. This is what the WebSphere for z/OS customized "BRAK" jobs do by default.

   Why? This is the default behavior because it allows a cell to be built quickly and without having to coordinate with an external CA. Being your own internal CA also involves no additional charge. It's a perfectly good way to establish a secure cell for small-scale and internal uses ... but as cited before, is probably not recommended for "real world" applications that are exposed to the population at large.

Where the CA resides

If you used an external CA like VeriSign, getting the server certificates "signed" involves sending a certificate request to VeriSign, who signs the certificate and sends it back to you. But if you're acting as your own CA (as we did for our test cell), then the server certificates are signed by our own internal CA.

So the question becomes: what server platform in the heterogeneous cell will possess the CA's public/private key pair and act as the CA for the purpose of signing the server certificates?

From a purely technical of view, any platform in the cell could serve as the CA for signing server certificates. Server certificates can be generated and signed on one platform and then exported from that platform and imported into a different platform.

For the purposes of our test cell, we used RACF on the WSCPLEX Sysplex as our CA.

Why? Two reasons: one, because we're the Washington Systems Center and z/OS is our thing ☺. But the other reason is because we built the first node of the cell on that z/OS Sysplex, and by default the BRAK jobs created a CA certificate and public/private key pair as well as the server certificates for that node. So, since we started on that platform and the CA public/private key pair was already there, we decided to make that platform where our CA would reside.

But again, technically there's no reason it has to be on z/OS.

However, once we settled on a given platform for the CA, that raised the next question.
Where are the server certificates generated and where are they signed?

The server certificates need to be generated somewhere. It's not a difficult process -- on z/OS it involves a RACF command; on the distributed platform it involves using the keytool utility to generate the certificate.

But to be useful, a server certificate needs to be signed by a CA. Therefore, the question came up: where would we generate the server certificates for the various platforms that make up our heterogeneous cell? There are two basic choices here:

1. Generate the server certificates on the same platform as the CA and sign them there. Then export the signed server certificate from the CA platform and take it to the server platform where it can be imported and used for the server.

   ![Diagram of certificate authority flow]

   Generating and signing certificates on the platform where the CA resides

   **Note:** This is what we did for the test cell we built. As mentioned under "Where the CA resides" on page 20, we had decided to use RACF on the WSCPLEX z/OS system to act as our CA. Therefore, we decided to use RACF on that z/OS system to act as the place to generate the certificates for all the servers and the place where they'd be signed. From there the certificates were exported from RACF on z/OS and imported on the other platform.

2. Generate the certificates on the platform where they'll be used, then export the unsigned certificate and import it to the platform where the CA resides. The CA would then be used to sign the server certificate. To get the signed certificate back to the original platform then requires the export from the CA platform and then back into the server platform.

   ![Diagram of certificate authority flow]

   Generating the server certificate on the server platform, then signing it on CA platform
This is essentially the process that would be used if an external, well-known CA was part of the plan. However, since we had decided on an internal CA, it didn't make sense to create extra work. We decided to generate and sign the certificates on the platform where the CA resided.

**Copying CA public/private key to another system not a good idea**

It's technically possible to copy a CA's public/private key pair to another system, effectively making a copy of the CA there. It's tempting to think this would be an easy way to generate and sign all the different server certificates where they'll be used. But this is bad practice for two reasons:

1. The CA's *private* key is a valuable thing, and should be guarded carefully. Having multiple copies of it scattered across different systems just increases the chance of a security breach.

2. Doing so could result in multiple server certificates having the same serial number, which is not the ideal.

**Key:** Once an internal CA has been established on a given server platform, leave it there. That then becomes the place where server certificates are signed by the CA. Do not copy the CA's public/private keys to another system unless you intend to move the CA there.

**When two separate Sysplex environments are part of the heterogeneous cell**

When a heterogeneous cell contains z/OS nodes from two separate Sysplex environments, a certain amount of care needs to be taken to insure the RACF information is properly defined on both. There are three basic ways to look at this:

1. When the RACF databases are entirely separate from one another
2. When the RACF databases are kept in sync using RR SF (RACF Remote Sharing Facility)
3. When the RACF database is shared between the two Sysplexes

**Completely separate RACF data bases**

There's a temptation to think that some of the customized jobs from the first Sysplex can simply be re-run on the second Sysplex to make things work. That's not a recommended approach for several reasons:

- If all the customized jobs from the first Sysplex were re-run on the second, it would result in the creation of an nearly identical configuration, with the same node name and server names, the same userids and everything else. That would result in trouble federating if a node by the same name -- the node from the first Sysplex -- was already federated into the Deployment Manager cell.

- The internal Certificate Authority created would have the same label, but the keys would be different.

**Note:** When a CA certificate is created with the RACDCERT CERTAUTH command in the Security Domain job, the public/private keys are generated at that time using a sophisticated algorithm that insures the key pair is unique. Re-running the job on another Sysplex would result in a new (and unique) key pair.

It's possible, therefore, that SSL connection requests would fail if a server certificate signed by the CA on Sysplex A was inspected with the CA Certificate generated on Sysplex B. The public key of Sysplex B's CA wouldn't be able to unlock the encryption generated by the private key of the CA on Sysplex A.

The important thing to grasp here is this: when a heterogeneous cell is planned which will incorporate nodes on different Sysplexes, and the RACF databases are
completely separate, it'll involve some coordination of customized job creation between one Sysplex and the other. Specifically:

- You'll need to determine ahead of time which Sysplex environment will be the first one to have a node built there. The other Sysplex will then piggy-back off some of the key variables (Config Group ID, Admin ID). This would be done by copying over the "saved variables" from the Security Domain customization.

- When the Security Domain BRAK job is run on the second Sysplex you'll *not* want to generate the CA certificate there. You could -- it wouldn't hurt anything provided it wasn't used -- but as we stated earlier it would result in a CA certificate with the same label as the one on the first Sysplex, but with different keys. Too much opportunity for confusion with that arrangement.

- When the BRAK job for the node is run on the second Sysplex, you'll want to strip out the commands that generate and sign the server certificates. As we stated earlier, you'll want to establish a single location for the CA and generate and sign the server certificates there.

  Note: But the commands that create the keyrings you'll want to leave in the BRAK job. When you're ready to import the signed server certificates the key rings will be already there. We illustrate which commands to strip out and which to keep in the section titled "ISPF panels and the generated jobs" starting on page 45.

  You'll definitely want to run the BRAK job for both the Security Domain and the WebSphere node being built. That's what creates the userids, groups and other necessary profiles on the local OS.

  Note: Unless you've got some kind of RACF database sharing or synchronization system set up between the different Sysplex environments. Then you may or may not run the BRAK job, depending on how much piggy-backing of profiles created on the first Sysplex you're planning on. More on this coming up.

Finally, you'll need to import into the second Sysplex's environment the *public key* of the CA -- not the private key, but the public key -- and connect that to all the server keyrings in the node.

See "Illustration of a Test Cell Built at the Washington Systems Center" starting on page 25 for a real-world scenario where this was done. The heterogeneous cell built at the Washington System Center had nodes from two separate Sysplexes where the RACF databases were entirely isolated from one another.

**Mirrored using RRSF**

One way to keep the RACF databases in two separate Sysplexes coordinated is to use RRSF -- RACF Remote Sharing Facility. In a nutshell, that facility watches for changes made in one RACF database and it mirrors the change over on the other.

  Note: It's a feature-rich product ... it can be configured to do one-way mirroring, two-way mirroring, and filter on which profiles are mirrored and which are not. We won't go into all the details on RRSF in this white paper.

With RRSF in play it's possible to simplify the construction of a heterogeneous node when two nodes are on different Sysplex environments. Depending on the nature of your configuration plans, with security profiles automatically mirrored over it may be possible to eliminate the need to run the BRAK jobs on the second Sysplex.
Note: “Depending on the nature of your configuration plans”... what we mean by this is that if you planned from the start to simplify your security profiles down to a single set of userids (one for controllers, one for servants, one for Daemons), and you made the other profiles generic (STARTED, CBIND and SERVER), it would be possible to create all the necessary profiles on Sysplex A and have them mirrored over to Sysplex B with RRSF. But there's a "gotcha"... that's next.

The reason this topic is raised is because of one aspect of RRSF that's not widely understood. It is this: **RRSF will not copy the private key for a certificate.** This is by design. Only public keys are copied over.

Is this a problem? Yes and no.

- For the CA certificate, no. We've stated earlier that it's best to have the CA reside in a single spot. For that certificate you don't want the private key copied over. But you do need the public key certificate copied out and in the keyrings of all the other servers. So here RRSF's behavior would work for you.

- But for server certificates, you need both the public and private keys over on the other Sysplex. Relying on RRSF to do this will result in problems. Only the public key will go, and that won't allow for proper SSL initialization. So if the server certificate generation and signing is done on one Sysplex, the only way to get those server certificates over to the other is to export the certificate from one Sysplex using PKCS12 encoding, and then importing it on the other.

**Bottom Line:** Don't rely on RRSF to copy your certificates from one Sysplex to another. Plan on the following:

- Have your CA on one Sysplex but not both
- Generate and sign all certificates where the CA resides
- Export server certificates using PKCS12 encoding
- Export CA public certificate using PKCS7 encoding
- Import certificates on other Sysplex
- Connect to the appropriate keyrings

**Completely shared using data sharing**

It is possible to have RACF database sharing between different systems not in the same Sysplex. This was commonly done in the "old days" before the advent of true data sharing in a Parallel Sysplex. It may still be done today between systems that are separate monoplexes.

If that's the case, then profiles created on one system are automatically available to the other system. That would include digital certificates, including private keys. In this sense, it would be just like how such profiles are automatically available between systems within a Sysplex.

**Note:** You can't share RACF between two Sysplexes that themselves are using Parallel Sysplex data sharing. However, if no Sysplex data sharing is used (for instance, monoplexes), then cross-system RACF data base sharing is permitted.

Just because the RACF database is shared doesn't mean you can automatically forego running the customized jobs or the BRAK jobs. You will most certainly need to run most of the customized jobs -- that's what builds the node configuration structure. The node BRAK jobs may or may not need to be run, depending on how much you're relying on "generic" profiles and a simplified userid set. (The Security Domain BRAK job would only need to be run once, just like is the case in a Parallel Sysplex with data sharing.)
To validate that a heterogeneous cell was possible, and more importantly -- to flush out all the "gotchas" that might exist along the way, a heterogeneous cell was built at the Washington Systems Center in Gaithersburg, Maryland. The topology of the cell looked like this:

The heterogeneous "xxcell" at the WSC

Sequence of construction

We'll use this simplified picture to represent the heterogeneous cell:

Simplified schematic of heterogeneous cell

The order in which this cell was built was:

- Standalone server on WSCPLEX built first
- Deployment Manager on xLinux built next
- Standalone federated into the Deployment Manager cell
- Work done to prepare for global security
- Global security turned on
- Standalone server on WPLEX created and federated into DMGR cell

Note: Does the order of construction matter? Not really ... prior to turning on global security the cell could have been built in a different sequence. We started with the node on z/OS to establish the starting point for the security profiles, most notably the CA certificate.
Standalone server on WSCPLEX

The Standalone server on WSCPLEX was built in the standard fashion:

- The ISPF panels were used to define a Security Domain. The customized jobs were run to create the RACF profiles for the Security Domain.

  **Note:** This job contained the command to generate the CA certificate. This established (for our configuration) RACF on WSCPLEX as the Certificate Authority.

- The SDCFG file from the Security Domain was brought into the ISPF panels for the Standalone server.

  **Note:** This is standard procedure. It insures that the Security Domain information is carried into the Standalone server's configuration.

- The customized jobs for the Standalone server were generated and run to create the Standalone server.

It was, really, a fairly standard construction of a Standalone server in preparation for federation into a Deployment Manager. A few notes about this:

- We didn't really take much care in our naming convention. Our objective was to put together a heterogeneous cell with security enabled and see what issues popped up. We followed the basics of our own naming standard recommendations -- all names start with a common "cell identifier" ("XX" in this case), and all names unique -- but beyond that we didn't try to modify the convention to provide a platform or Sysplex identifier.

  **Note:** That's why you're not seeing too much detail about the actual names used for this cell. It works, but it's not pretty in all areas. The important point for you is to craft a naming convention to that does what you need it to do. See "Meaningful naming convention" on page 9 for more.

- Because we knew this Standalone Server would get federated into a Deployment Manager cell, the cell name we used was a "temporary" name. This is standard procedure. But we planned ahead and made sure that the temporary name we gave this Standalone Server would be different from that provided for the Standalone Server on WPLEX. See "Node Group short name" on page 13 for a discussion of why this was necessary.

- An important aspect of this was what we discussed back under "J2EE role enforcement" on page 15. Knowing that this cell would be heterogeneous and span z/OS and non-z/OS platforms, the "Use SAF to enforce J2EE roles" question was answered "N".

  **Additional z/OS Security Customization Options**
  
  Generate default RACF realm name: N
  Default RACF realm name ....: WSCPLEX
  
  Use SAF EJBROLE profiles to enforce J2EE roles: N
  
  Enable SAF authentication using LTPA or ICSF login tokens: Y

  **Answered "N" to the "Use SAF EJBROLE profiles to enforce J2EE roles"**

  **Note:** See "J2EE role enforcement" on page 20 for a little more on this topic.

- Security was not on for this Standalone server.
At this point our configuration looked like this:

![Diagram showing the configuration](image)

**Status of the heterogeneous cell**

**DeploymentManager on xLinux**

The standard install shell for the Deployment Manager on xLinux was used to create the DMGR there. The cell name provided here was the cell name that would be used for the heterogeneous cell.

The status of the cell at this point was as follows:

![Diagram showing the status](image)

**ORB character encoding**

The issue here is that WebSphere V6 on z/OS understands **ISO8859-1** encoding (single character) and **UCS2** encoding (wide) only. WebSphere V6 on the distributed platform defaults to **UTF-8** and **UTF-16**.

To allow the Deployment Manager running on xLinux to communicate properly with the nodes on z/OS, it was necessary to go into the `<node_root>/java/jre/lib/orb.properties` file on the DMGR box and tell it to use **ISO8859-1** and **UCS2**. By so doing, WebSphere on the two platforms could talk to one another and understand what in the world each was saying.

**Notes:**
- WebSphere on z/OS will be compatible with **UTF-8** in the future.
- Any time a heterogeneous cell has nodes on z/OS and distributed, the distributed server nodes will have to be modified to use **ISO8859-1** and **UCS2**.

The file is located in the following location:

`<node_root>/java/jre/lib/orb.properties`

We did the following:
Changing the ORB character encoding

And of course a change like that required that the DMGR be stopped and restarted.

Federated z/OS node into xLinux DMGR

The Standalone server node on z/OS was then federated into the Deployment Manager cell running on zLinux. There are three different ways this can be accomplished:

1. Use the ISPF option on z/OS to create a customized federation batch job. This creates a JCL job that invokes BPXBATCH that then runs the `addNode.sh` shell script with all the parameters you specified in the ISPF panels. This method is useful in that you can easily control the values for the Node Agent -- name, ports, etc. The DMGR must be running for this process to work.

2. Use the `addNode.sh` shell script on z/OS directly. This is essentially the same thing as option #1, except the parameters are not built for you. The minimum required parameters are the host and port where the running Deployment Manager can be reached. That'll result in a Node Agent created with default values. The DMGR must be running for this process to work.

3. Use the "Add Node" function of the DMGR's Admin Console to "go and get" the node from the Standalone server. To use this function the Standalone server must be running. You provide the host and port where the Standalone server can be reached, and the DMGR remotely invokes the `addNode.sh` processing to federate the node.

They all accomplish the same thing. With global security off at this point, the federation was little different from the federation of a node all within a z/OS system. Again, the architecture of WebSphere is distributed and as long as there's a TCP network between the DMGR and the node being federated, it should work.
The status of the heterogeneous cell now looked like this:

---

**Status of the heterogeneous cell**

**Prepared for global security**

The preparation work needed before global security could be turned on revolved around two main things: the establishment of an external user registry and the creation and distribution of the certificates. There were a few other small things that popped up as well.

**Planned the distribution of server and CA certificates**

As stated earlier, if global security were left off, the construction of a heterogeneous cell would be a fairly straight-forward process. By turning it on, the use of SSL then comes into play, and SSL requires the coordination of certificates. This was explained back under “Certificates, Certificate Authorities, digital signatures, keyrings, keyfiles ...” starting on page 17.

For this cell we decided to use an internal CA, which is the default assumed by WebSphere for z/OS. Further, we decided to establish our CA on the z/OS platform on the WSCPLEX, the first node we built (see “Where the CA resides” on page 20). That means we would use RACF on z/OS to generate certificates and sign them with the CA certificate also created by RACF on that platform.

We then took a look at our present cell and conducted an inventory of what certificates were presently created, and which ones were still needed:
Initial status of certificates in the cell

Notes:

1. On z/OS the certificates are stored in RACF and are associated with a "keyring." Keyrings are identified with an ID. Our Standalone Server (which was then federated) was built using the default BRAK jobs, which means that a keyring was built for controller ID and a different keyring was built for the servant ID. The Node Agent and the application server controller both used the same controller ID, and thus both used the same keyring. The default assumption made by WebSphere for z/OS is that all the controllers in the node will use the same ID, and all servants in the node would use the same ID, though different from the controller ID.

   Note: That's the default behavior. Some people opt to have unique IDs for each controller and servant; others opt to simplify things even more than the default and have one ID for both controllers and servants. We won't plumb the depths of the pros and cons of those here. Suffice to say the default behavior is as we show you the picture above.

The RACF command used to create a keyring looked something like this:

```
RACDCERT ADDRING(WASKeyring) ID( XXACRU )
```

2. The keyring associated with the controller ID holds the server certificate, the server public and private keys and the CA's public key certificate. The server certificate was generated and signed when the BRAK job for the Standalone Server was run. The commands used to generate the server certificate, sign it and connect it to the ring looked something like this:
This generates the server certificate and signs it with the CA certificate, which was created when the Security Domain BRAK job was run.

Note: Once the server certificate is generated, it's connected to the controller keyring.

Finally, the CA's public key certificate is connected to the controller's keyring as well.

3. The CA's public key certificate is also connected to the servant's keyring. This is something new in V6. Back in V5 the servant didn't have a keyring, but in V6 it does. The command to connect the CA certificate to the servant's keyring is like the one to connect it to the controller keyring. The only difference is the ID( ) field and the ID that's named in there.

4. RACF was holding the internal Certificate Authority with its public and private keys. This CA was generated in the Security Domain's BRAK job with a command that looked something like this:

RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('WAS CertAuth for Security Domain') OU('XXCELL.WebSphere for zOS')) WITHLABEL('WebSphereCA') TRUST NOTAFTER(DATE(2010/12/31))

5. The Deployment Manager on the xLinux server had, by default, two keyfiles created for it with operational certificates in them. On non-z/OS platforms the files are known as "keyfiles" (for server certificates) and "trust files" (for CA certificates), but the concept is identical to "keyrings" on the z/OS platform. We chose to disregard these files for two reasons:

   - The keyfiles and the certificates inside are shipped with WebSphere, not generated at the time of installation. Therefore, the certificates in there are the same as any other default WebSphere installation out in the world. The z/OS CA certificate was generated when the security domain BRAK job was run, and the keys in that certificate are unique.

   - We already established the z/OS platform as the place where our CA would reside and where certificates would be generated and signed. If we maintained these default key files, it would imply a second CA in the picture. That's technically feasible, but it complicates matters because it requires two CA public key certificates to be placed in all the server keyrings (or keyfiles) in the cell. So we opted to discard these and bring new certificates over from the z/OS platform.

   Note: This implies telling the Deployment Manager to use new keyfiles. We discuss that under "Configured Deployment Manager to use new key files" on page 37.

In summary, because we started the construction of this heterogeneous cell on the z/OS platform, and because we decided to have our CA on that z/OS system, the keyrings and certificates for that node were all properly defined.

Note: That won't be the case when we build our node over on the other Sysplex. We had to coordinate the movement of certificates from WSCPLEX over to WPLEX when we built that node. See "Standalone server on WPLEX" starting on page 42.

The Deployment Manager, however, needed some keyfiles and certificates. That was next.
Generated server certificate for Deployment Manager

The Deployment Manager server needs its own certificate along with a private/public key pair. As stated back under "Where the CA resides" on page 20, we had decided to make RACF on the z/OS system in WSCPLEX the Certificate Authority for the whole cell. That meant for our cell we needed to generate the certificate and export it.

An example of the format of the RACDCERT command needed to generate a server certificate was found in the BRAK job for the Standalone Server. We took that command and modified it to create a server certificate for the Deployment Manager:

```
RACDCERT ID (XXACRU) GENCERT SUBJECTSDN(CN('XXDMGR.NULL.WASHINGTON.IBM.COM')
O('IBM') OU('XXCELL')) WITHLABEL('DefaultWASCert.DMGR')
SIGNWITH(CERTAUTH LABEL('WebSphereCA.XXCELL'))
NOTAFTER(DATE(2010/12/31))
```

Command to generate the DMGR server certificate

**Notes:**

1. Since we were using RACF on z/OS as the tool to generate the certificates, it required a valid ID value. The value XXACRU was the ID created for the Standalone Server (now federated) controller. We could have given a different ID here -- say, XXDCRU for Deployment Manager CR user -- but that would have meant creating that ID prior to trying to issue this RACDCERT command.

   Instead, we simply used the already-existing XXACRU ID for this command. The certificate generate was different from the server certificate for the application server (generated when the Standalone Server BRAK job was run) because the CN value of this command was changed to make it unique.

2. For the CN value we used the host name of the xLinux box where the DMGR was running.

3. The WITHLABEL field is the label of the server certificate being generated. We took the default value of DefaultWASCert and tagged DMGR on the end. When this certificate was imported into the key files on the xLinux box, the keytool utility asked for the label again. So in truth this label generated here on z/OS was somewhat meaningless, except that it the label should be unique in RACF on the z/OS system.

4. The label WebSphereCA.XXCELL is the label of the Certificate Authority that will sign this server certificate. The CA was created when the Security Domain BRAK job was run. The default value created by WebSphere for z/OS is WebSphereCA.xxxxx, where "xxxxx" is the "Security Domain Identifier" provided on the ISPF panels (if one is provided). We took the default when we created our Security Domain, therefore the value for the CA was the default -- WebSphereCA.XXCELL.

The result of this was a server certificate created and stored in RACF that was intended for the Deployment Manager on xLinux. The next step was to export two certificates from RACF on z/OS and take them to the xLinux box:

- The just-created server certificate along with its private key
- The CA certificate with its public key (but not the private key)
Export certificates from z/OS

Here's a picture overview of what was done:

Exporting and downloading certificates to xLinux server where DMGR was running

The "exporting" of a certificate from RACF is really little more than the writing of the certificate out to a sequential data set. The certificate may then be binary FTPed to wherever it needs to go.

Exporting the CA certificate

The CA certificate was exported without its private key. This is standard practice with the CA's public certificate, which is the one that's widely distributed. That means the certificate is exported without a password, in a format known as "DER encoded PKCS7".

Here's the command we used:

```
RACDCERT CERTAUTH EXPORT (LABEL('WebSphereCA.XXCELL'))
    DSN('XXCELL.CACERT.PKCS7') FORMAT(CERTDER)
```

Exporting the server certificate and private key

Exporting the server certificate required that the private key be exported as well. That means the data set exported certificate (and key) needs to be password protected. This is the command we used to export the certificate and key:
RACDCERT  ID(XXACRU)  EXPORT(LABEL('DefaultWASCert.DMGR'))
DSN('XXCELL.DMCERT.PKCS12')  FORMAT(PKCS12DER)  PASSWORD('xxxxxx')

Command to export the server certificate from RACF

Notes:
1. The ID for which the server certificate was initially created. (See "Generated server certificate for Deployment Manager" on page 32.)
2. The label of the certificate to be exported. This had to match exactly the label that was applied when the certificate was created.
3. The name of the sequential data set that will be used to hold the exported certificate. This does not need to be pre-allocated; RACF will allocate automatically.
4. The format of the file ... PKSC12DER implies “DER encoded PKCS12”.
5. The password to be used in encrypting the certificate information in the sequential file.

FTPed files to the xLinux box where DMGR was defined

The two sequential data sets created in the previous two steps were FTPed to the xLinux box in binary mode.

Note: PKCS7 files can, in theory, be transferred in ASCII. But the rule of thumb “always binary” is easier to remember.

Imported downloaded certificates into new keyfiles on xLinux system

The z/OS system uses RACF as its repository for certificate and key information, but the distributed systems (including the xLinux box we used as our Deployment Manager) use files called "key files." There are actually two such files that go hand-in-hand:

• KeyStore -- for private certificates and keys
• TrustStore -- for CA certificates

Note: RACF has no such distinction. There, a "keyring" can hold either a private certificate, a CA certificate, or both. In fact, the keyring for the application server controller in our test cell had just that -- it's own private server certificate and a copy of the CA's public certificate.

The way you get certificates and keys into those files is by using a tool called keytool, which is supplied with the IBM JDK that ships with WebSphere. You can find that tool down under the /bin directory of Java.

Default key files and new key files

As mentioned earlier, WebSphere on the distributed platform ships with two key files by default:

dummyKeyStore.jks
dummyTrustStore.jks

But we chose not to use those files. They contained certificates shipped with WebSphere, and the same certificates are shipped with every copy of WebSphere. Those files work fine when first setting up WebSphere, but beyond initial tests you should create new files and use different certificates and keys. That's exactly what we did. We ended up creating two new files:
xxcellKeyStore.jks
xxcellTrustStore.jks

Note: The name didn't really matter, though the extension should be jks as shown. Whatever the name is, the files should be properly pointed to in the Admin Console. We show that under "Configured Deployment Manager to use new key files" on page 37.

How did we create those files? With keytool ... when we imported the certificates. We'll show how that was done under "Imported certificates into new key files" on page 36. First, an explanation of something we encountered regarding keytool's ability to decrypt the strong encryption created by RACF.

Unrestricted policy files

Note: This information is only applicable when exporting from RACF and downloading PKCS12 files. That would include any private certificates with a private key, but would not include a CA public certificate. So, depending on where your CA is located, you may not need to do this. For example, if you used keytool to generate your CA certificate on the distributed box, you would not need to do this.

It turned out that certain encryption "policy files" shipped with the JDK of WebSphere were "restricted" -- that is, not quite so powerful. That was so the WebSphere product could more easily adhere to encryption technology export regulations. Due to these "less powerful" (or "restricted") policy files, the keytool utility was unable to import the PKCS12 file we exported from RACF. The error symptom was this:

```
keytool error (likely untranslated): com.ibm.security.pkcsutil.PKCSException: Private key decryption error: (java.lang.SecurityException: Unsupported keysize or algorithm parameters)
```

The solution is to download the "Unrestricted JCE policy files" from the IBM website and overlay the existing files.

Where to get "Unrestricted JCE policy files"

IBM supplies a website to download a zip file that contains the two files you need. However, you'll need to register before you can download the files. The registration is free and immediate, but it does require supplying your e-mail address and creating a userid/password. The website is:

```
```

Select the JDK level you're presently using (JDK 1.4.1 or JDK 1.4.2), then scroll down and click on "IBM SDK Policy Files". Then work through the registration and get to the point where you can download the ZIP file with the files.

For us the file name was unrestrict142.zip.

Where policy files exist in JDK directory

There are two policy files we had to replace:

```
local_policy.jar
US_export_policy.jar
```

They're in the /lib/security directory of the JDK you use to invoke keytool.

Note: The ZIP file you download will contain those two files. The objective was to replace the existing files with the ones in the downloaded ZIP file.
**Replaced files**

We first backed up the existing JAR files and then copied in the new, "unrestricted" files.

**Imported certificates into new key files**

The certificates -- downloaded from RACF on z/OS -- were ready to be imported into the key files. We had two certificates:

<table>
<thead>
<tr>
<th>Certificate</th>
<th>Format</th>
<th>Destination File</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA Certificate with public key</td>
<td>PKCS7</td>
<td>xxcellTrustStore.jks</td>
</tr>
<tr>
<td>Server Certificate with public and private key</td>
<td>PKCS12</td>
<td>xxcellKeyStore.jks</td>
</tr>
</tbody>
</table>

**Import of server certificate**

Here’s an example of the command used to import the server certificate. Remember, the server certificate came down as a PKCS12 -- *password protected* -- file.

```
keytool -import -file <downloaded file> -keypass <aaaa>
-pkcs12 -keystore xxcellKeyStore.jks -storepass <bbbb>
```

**Notes:**

- The entire string is entered as one command at the command prompt of the server operating system
- `<download file>` is the name of the file on the distributed platform after it was FTPed from z/OS.
- `<aaaa>` was the password we supplied on the RACF command to export the file. This was used by keytool to unlock the protected private key inside the file.
- `<bbbb>` was the password we wished to be used to lock the keyfile itself. This password will be supplied to WebSphere through the Admin Console so WebSphere can open and use the certificates.

**Import of server certificate**

And here’s an example of the command used to import the CA certificate. That was a PKCS7 file and as such had no password protection.

```
keytool -import -file <downloaded file> -keystore
xxcellTrustStore.jks -storepass <bbbb>
```

**Notes:**

- The entire string is entered as one command at the command prompt of the server operating system
- `<download file>` is the name of the file on the distributed platform after it was FTPed from z/OS.
- `<bbbb>` was the password we wished to be used to lock the keyfile itself. This password will be supplied to WebSphere through the Admin Console so WebSphere can open and use the certificates.
Configured Deployment Manager to use new key files

This setting is found under the properties for the "SSL Configuration Repertoire" for the Deployment Manager. That's found under "Security" ⇒ "SSL".

Keyfile and Trustfile definitions in Admin Console

Notes:

1. The default key file name will be DummyServerKeyFile.jks. Rather than use that file, we created a new key store file called xxcellKeyStore.jks and placed it in the same directory. See "Imported downloaded certificates into new keyfiles on xLinux system" on page 34 for information on the creation of that new key file.

2. The "Key file password" field is used to specify the password established for the key file (xxcellKeyStore.jks in our cell). This is the password used when creating the key files through keytool back under "Imported downloaded certificates into new keyfiles on xLinux system" on page 34. The files are password protected to keep those who have simple file permission access to the file from getting into file and messing around with the keys. WebSphere needs to get into the file, however, so the password to the key file is presented here.

   Note: The Trust File has the same password protection mechanism.

3. The format of the key stores is JKS, so the "Key file format" value here was set to the same.

4. On the distributed platform the CA certificate is kept in a separate file called a "Trust file." On z/OS we have just a keyring, which can hold both a personal certificate and a CA certificate, but on distributed they're split out like this.

5. The file path to the key file and the trust file contains a variable -- ${USER_INSTALL_ROOT}. The value of that variable will be set when you initially configure the node. On z/OS it'll be the configuration HFS mount point plus /DeploymentManager for a DMGR or /AppServer for an application server node. On the xLinux box we used for our DMGR, the value was:

   /opt/IBM/WebSphere/AppServer/profiles/xxdmgr01

   The important thing is that path to the key file and trust file accurately and properly point to where those files really are. That includes the contents of the variable. So if you're not sure what the value of that variable is, you should verify its contents. Go to "Environment" ⇒ "WebSphere Variables" and scope the variables to the node for which the SSL repertoire applies. Then scroll down and look at the contents to the USER_INSTALL_ROOT variable.

© 2007, IBM Americas Advanced Technical Support
Washington Systems Center, Gaithersburg, MD

- 37 -

Section: WSC Sample Illustration
Version Date: Wednesday, September 12, 2007
Configured LDAP server and loaded user data

As noted under "External user registry -- LDAP" on page 14, when a cell is heterogeneous an external user registry is required. Contrast this with a cell that's entirely on z/OS: there RACF can be used as a the common user registry. But WebSphere on xLinux can't access RACF, so some common registry is required.

For this cell, we built an LDAP server to act as the common user registry. Some properties of that LDAP server:

<table>
<thead>
<tr>
<th>Server userid</th>
<th>cn=xxadmin,ou=xxcell,o=wsc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>9.82.24.153</td>
</tr>
<tr>
<td>Port</td>
<td>389</td>
</tr>
<tr>
<td>Base distinguished name (DN)</td>
<td>ou=xxcell,o=wsc</td>
</tr>
<tr>
<td>Bind distinguished name (DN)</td>
<td>cn=root</td>
</tr>
</tbody>
</table>

**Note:** That information is what we defined in the Admin Console to tell the cell to use that copy of LDAP. More under "Configured cell to use LDAP as user registry" on page 38.

Here's what was loaded into the LDAP server for the XXCELL:

**Notes:**
- The blank cn= values actually contained some of our personal userids. We added those as a test. Rather that expose that IDs in a white paper, I deleted them from this bitmap.
- The need for the Admin ID should be obvious: when global security is turned on the userid and password given will be authenticated through LDAP. But what about the controller and servant IDs? Yes ... when the servers make administrative requests of each other (using JMX over SOAP with SSL) they pass an LTPA token that contains their identity. For example, the Node Agent will try to synch with the DM, so it sends an LTPA token and a synch request. The server that receives the request goes to LDAP to retrieve the list of the groups that that identity belongs to. The controller ID is for certain needed; I'm less certain about the servant ID. But it doesn't hurt to load those as well.

**Configured cell to use LDAP as user registry**

Once the LDAP server was up and operational, loaded with the user data, it was time to set the cell's "Active User Registry" to LDAP. This setting can be found under "Security" ⇒ "Global Security." It involved two steps:

1. Changed the "Active User Registry" setting to "Lightweight Directory Access Protocol (LDAP) user registry":

![Diagram of the LDAP server structure with notes regarding blank cn= values and users and group defined to LDAP.](image-url)
Set Active User Registry to LDAP

2. Set the "User Registry" properties for LDAP:

- **General Properties**
  - **Server user ID**: cn=wwadmin,ou=wwcell,o=wsi
  - **Server user password**: ********
  - **Type**: Custom
  - **Host**: 19.62.24.151
  - **Port**: 389
  - **Base distinguished name (DN)**: ou=wwcell,o=wsi
  - **Bind distinguished name (DN)**: cn=root
  - **Bind password**: ********
  - **Search timeout**: 120 seconds
  - **Reuse connection**: 
  - **Ignore case for authorization**: 
  - **SSL enabled**: 
  - **SSL configuration**: %dmnode/DefaultSSLSettings

**Set LDAP properties**

**Note:** The information populated here is what was captured when the LDAP server was configured. See "Configured LDAP server and loaded user data" on page 38.

**Turning on global security**

With the LDAP server created and loaded with user data, and the LDAP server configured into the DMGR's security settings, the final steps could be taken to enable and turn on global security.

**Generated LTPA keys**

This was a relatively simple process involving specifying a password for the keys that WebSphere would generate:
Setting the password for the LTPA keys

**Note:** The value you supply is not tied to any of the IDs created earlier. This is just a string of characters used to password-protect the LTPA keys that are generated. You can supply any string that you want.

**Important: verified node and Node Agent operational before saving settings**

It was (and is) very important to be sure the Node Agents for all the nodes in the cell are up and ready to accept synchronization when the "Enable global security" setting is saved. If the Node Agent is *not* up when the change is saved, then the node will have no knowledge of global security having been set for the cell.

**Note:** This is true of all Network Deployment configurations, not just heterogeneous cells.

If the "global security = on" setting is saved only to the DMGR, but not to the nodes, the problem that occurs is this:

- The DMGR is restarted to pick up changes. It starts and sees that global security is enabled. It then assumes SSL will be used for all connections.
- The node -- if the settings were *not* successfully synchronized out -- starts up and has no knowledge of global security having been turned on.
- The DMGR demands and SSL connection, the Node Agent has no idea how to respond.

The key to avoiding this problem is to make certain that when the global security settings are saved in the DMGR, that the settings are successfully synchronized out to *all current nodes in the cell*. In our cell, that meant one application server node: the one on the z/OS system in the WSCPLEX Sysplex.

**Note:** In our cell, the z/OS node on the WPLEX had not yet been built. That was okay: it’s possible to federate a node into a cell where global security is already turned on. In that case the federation process insures the proper synchronization of the global security settings to the newly federated node.
**Set final few properties**

The final few checkboxes and drop-down lists were set.

<table>
<thead>
<tr>
<th>General Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Enable global security</td>
</tr>
<tr>
<td>☐ Enforce Java 2 security</td>
</tr>
<tr>
<td>☐ Enforce fine-grained JCA security</td>
</tr>
<tr>
<td>☐ Use domain-qualified user IDs</td>
</tr>
<tr>
<td>* Cache timeout</td>
</tr>
<tr>
<td>✓ Issue permission warning</td>
</tr>
<tr>
<td>Active protocol</td>
</tr>
<tr>
<td>Active authentication mechanism</td>
</tr>
<tr>
<td>Active user registry</td>
</tr>
<tr>
<td>☐ Use the Federal Information Processing Standard (FIPS)</td>
</tr>
</tbody>
</table>

**Enabling global security**

We saved the changes and synchronized out to the node.

**Stopped all servers and restarted**

To activate the global security settings, it was necessary to stop the servers and restart them. The Deployment Manager (on xLinux) was stopped, as was the Daemon, Node Agent and Application Server on the z/OS system in the WSCPlex Sysplex. Our entire cell was stopped.

Then the order of restart was:
- Deployment Manager first
- Node Agent second (the Daemon was started automatically at this point)
- Application Server third

**Status of the cell**

Here's a picture of the status of the cell at this point:
The final step was to construct the second application server node. This was to reside on z/OS as well, but a z/OS system on a different Sysplex. That’s important. Because had the second application server node been built on the same Sysplex -- WSCPLEX -- then much of what we’ll discuss here wouldn’t have been an issue.

**Why?** Because if the second node was on the same Sysplex, then we would have had access to all the common security profiles through the shared RACF on that Sysplex. The most significant of which would have been the CA certificate and its private key. But the WPLEX was entirely separate Sysplex from WSCPLEX, so the issue of where the CA resides and where certificates are generated and signed applies. As stated under “Where the CA resides” on page 20, we chose to create and maintain our internal CA on z/OS on WSCPLEX. Therefore, we’ll need to generate the WPLEX certificates on WSCPLEX and then export/import them. In concept this is the same thing we did for the DMGR running on xLinux. The difference will be the mechanics of importing the certificates -- on xLinux we used `keytool`; on z/OS we’ll use RACF.

**Preliminary planning**

The objective was to build a Standalone server on another Sysplex so that when that node is federated into the heterogeneous cell -- with global security on -- things would work okay. To accomplish that we had to insure a few things:

- The "temporary" cell names for this Standalone Server on **WPLEX** would be different from those used on **WSCPLEX**. See "Node Group short name" on page 13 for more on why.
The "Config Group" was the same on the second Sysplex environment as was used on the first. It's theoretically possible to have a different Config Group, but that introduces more complexity, such as keeping the group/id relationships correct in LDAP.

The "Use SAF EJBRoles to enforce J2EE roles" question was answer "N" for the WPLEX node as it was for the WSCPLEX node.

A single CA was used for the whole cell, and that CA resided over on the WSCPLEX Sysplex.

The issues sited for the first three bullets were accomplished by copying the Security Domain "saved variables" file from WSCPLEX to WPLEX and using that as input to the Security Domain ISPF panels. That insured commonality between the two security environments.

Some editing of the BRAK jobs was necessary to make sure things like a CA certificate and server certificates weren't re-created on WPLEX. We planned on copying them over from WSCPLEX. See "ISPF panels and the generated jobs" on page 45 for more.

The same controller and servant ID would be used for the servers on WPLEX (our second Sysplex) as was used on WSCPLEX (our first Sysplex). This was done to keep things simple. Unique IDs could have been used, but it would have required generated a new server certificate for the new controller ID.

Note: By using the same ID on both Sysplexes it allowed us to simply export the server certificate for XXACRU from WSCPLEX and import it over on WPLEX. By loading the saved variables for the Standalone Server from WSCPLEX, we insured the ID values would be the same.

The common CA public key certificate was properly distributed to the various keyrings on the WPLEX Sysplex node. This was a manual operation. See "Coordinated CA certificate and server certificate" on page 44.

Would RRSF have made this easier?

Somewhat, but it would not have eliminated all the manual coordination. The groups, ID, keyrings and other profiles would have been copied over.

Note: But the BRAK jobs would still have to be run. Some of the profiles -- CBIND and SERVER specifically -- would have been unique, based on the "cluster transition name."

But what would not have been copied is the server certificate private key. That's by design. That means SSL for the servers on the second (or RRSF target) Sysplex would not have worked. The workaround is to manually bring the server certificate with its public key over, which is what we did.

See "Mirrored using RRSF" on page 23 for more on the RRSF question.

See "Coordinated CA certificate and server certificate" on page 44 for more on the export and import of the server and CA certificates across the z/OS systems.

Copied Security Domain and Standalone Server saved variables to WPLEX

This was not a strictly required step. We could have written the information from the Security Domain on a piece of paper and re-entered it all. But copying the saved variables helped insure consistency of key information. Two sequential data sets copied over:

- The SDCFG file for the Security Domain
- The SAVECFG file for the Standalone Server
Coordinated CA certificate and server certificate

Since we planned to use the same CA on WPLEX as on WSCPLEX, and the same userid for controllers and servers on WPLEX as on WSCPLEX, this allowed us to simply export the existing certificates and carry them over to WPLEX:

<table>
<thead>
<tr>
<th>Certificate</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA Certificate with public key</td>
<td>PKCS7</td>
</tr>
<tr>
<td>Server Certificate with public and private key</td>
<td>PKCS12</td>
</tr>
</tbody>
</table>

The export commands were essentially the same as we used back when we exported the certificates to take them to the Deployment Manager (see “Generated server certificate for Deployment Manager” starting on page 32).

In picture format, here’s what we did:

Export of CA public key certificate and server public/private key pair

Exported CA certificate and server certificate

The commands used to export the certificates where:

```
RACDCERT CERTAUTH EXPORT(LABEL('WebSphereCA.XXCELL'))
DSN('XXCELL.CACERT.PKCS7') FORMAT(CERTDER)
```

Command to export the CA certificate from RACF

Note: That's the exact same command used to export the CA certificate to take it to the DMGR server. In fact we didn't actually export it a second time. We simply used the exported certificate from before.
This was not the same command used to export the server certificate for the DMGR. If you recall, we created a new server certificate for the DMGR and changed a few things in the LABEL value. Here we're exporting the certificate that was originally generated when the BRAK job for the Standalone Server on WSCPLEX was executed.

**Imported CA certificate**

The exported CA certificate was brought into RACF on the WPLEX with a command that looked something like this:

```
RACDCERT CERTAUTH ADD('XXCELL.CACERT.PKCS7') TRUST
WITHLABEL('WebSphereCA.XXCELL')
```

It'll be important to bring the CA certificate in with a label identical to label that will be assumed when the BBOWBRAK job is generated for this Standalone Server ... that is, unless you plan on manually connecting this CA certificate to all the keyrings.

The label the BRAK job will assume will be determined by the value specified in the Security Domain definition, from panel 2 of 2 of that definition. The "XXCELL" portion of the command shown above is the "Security Domain Identifier" from panel 1 of 2. That may or may not be used, depending on whether you answered "Y" to the "Use security domain identifier" question on panel 1. If so (and we did), then the format of the label will be as shown above, with "WebSphereCA" being the default label.

**What about server certificate?**

The server certificate had to wait a bit ... the import of that certificate required an owner ID, and that ID -- the controller ID for our Standalone Server configuration -- hadn't yet been created (because we hadn't yet run the BRAK job). Further, the group to which the ID was a member wasn't yet created. So the import would have failed at this point.

Here's the sequence of activities that worked:

- Generated and run the security domain jobs (with one alteration ... explained on page 46)
- Generated but did not run the Standalone Server jobs (except the BBOCBRAJ job that generates the RACF commands for use by BBOCBRAK)
- Inspected the commands generated and copied the command to create the controller userid and connect it to the Config Group.

**Note:** The Config Group is created when the Security Domain job is run.

- Manually created the controller ID and connected it to the group
- Imported the server certificate
- Run the BBOCBRAK job (with a few modifications ... see page 47)

**Note:** The command in the BRAK job to create the ID will fail (duplicate error), but that's okay. More importantly, all the commands to create the keyrings and connect the CA certificate to the rings will succeed.

**ISPF panels and the generated jobs**

It was still necessary to go through the ISPF panels and generate customized jobs to build the Standalone Server on the WPLEX system. The configuration HFS had to be built and populated, and all the necessary local security information had to be created.
There were two sets of ISPF dialogs and customized jobs: the Security Domain and the Standalone Server itself. Each had unique elements to it, including the editing of the BRAK jobs to remove certain commands.

**Security Domain panels and jobs**

Using the SDCFG saved variables from WSCPLEX, the Security Domain jobs were created. Nothing on the panels were changed -- the identical information was used to generate the jobs for WPLEX as was used to generate the jobs for WSCPLEX.

But the BBOSBRAK member in the DATA data set was edited to remove the small section of RACF commands used to "Create CA Certificate for WebSphere Security Domain." Back under "Coordinated CA certificate and server certificate" on page 44, we had brought that CA certificate over to this Sysplex's RACF.

The section removed looked something like this:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('WAS CertAuth for Security Domain')  OU('XXCELL.WebSphere for zOS'))
WITHLABEL('WebSphereCA')  TRUST NOTAFTER(DATE(2010/12/31))
```

With that removed, the job was run. This created the basic security infrastructure in support of the next set of customized jobs.

**Note:** Would it have hurt had that command been run? No. But it would have meant that both Sysplexes would have had CA certificates with the same label, but different keys. That would have been very confusing. Much better to stick to our plan and maintain a single CA on our original Sysplex.

**Standalone Server panels and jobs -- generated but not run**

The saved variables for the Security Domain and the Standalone Server for WSCPLEX were loaded into the dialogs on WPLEX. We then ran through the panels making changes where necessary:

- The SYSPLEX and system names were changed to reflect the new environment
- The node long and short name was made unique
- The server long and short name was made unique
- The temporary cell long and short name for the Standalone Server was made unique from what was used before.
- The IP host names were changed to those of the WPLEX system

**Note:** See "Insuring unique names" starting on page 10. The point here is that those things that would normally require to be unique -- names, IP host values, system names, Sysplex names -- were updated. All the other stuff was left as it was.

**Manually created controller ID and connected to configuration group**

As we mentioned in the previous section, we had to create the XXACRU ID manually so the server certificate could be imported and owned by that ID. The Security Domain job had to be run first to create the configuration group.

Running the Standalone Server's BBOCBRAJ job will generate the BBOWBRAK member in the DATA data set. That member contains all the RACF commands that will be executed by BBOCBRAK. We noted the command used to create the controller ID and connect it to the config group, and we manually executed those commands:
ADDUSER XXACRU DFLTGRP(XXCFGP) OMVS(UID(2500) HOME(/tmp)
PROGRAM(/bin/sh)) NAME('WAS APPSVR CR') NOPASSWORD NOOIIDCARD
CONNECT XXACRU group(XXCFGP)

**Imported server certificate**

With the ID created, the server certificate could be imported. Here's an example of the command used:

```
RACDCERT ID(XXACRU) ADD('XXCELL.SRVCERT.PKCS12) TRUST
WITHLABEL('DefaultWASCert.XXSR01') PASSWORD('xxxxxx')
```

**Notes:**

- It's a PKCS12 file, so the key is encrypted with a password. The "xxxxxx" in the example command is meant to be the password used when the certificate was exported from RACF.
- The `WITHLABEL` value should be set equal to the label assumed by the BRAK job. If you're unsure, you should look in the generated `BBOWBRAK` member. Look for the string "Connecting Server Certificates to their keyrings" and note the LABEL value on the `RACDCERT` command that follows.

**Modified BBOWBRAK member**

The generated `BBOWBRAK` member has a command in it to generate the server certificate and sign it with the CA certificate. We didn't want that command run -- we just imported the properly signed server certificate. So we edited the `BBOWBRAK` member and stripped out the following command:

```
RACDCERT ID (XXACRU) GENCERT SUBJECTSDN(CN('XXACRU.XXSR01')
O('IBM') OU('XXCELL')) WITHLABEL('DefaultWASCert.XXSR01')
SIGNWITH(CERTAUTH LABEL('WebSphereCA.XXCELL'))
NOTAFTER(DATE(2010/12/31))
```

All the other commands -- the creation of keyrings and the connection of certificates to the keyrings -- were all necessary things. It was important to get the CA certificate and the server certificate into RACF on WPLEX from RACF on WSCPLEX prior to running the job so all those commands would execute properly.

**Notes:**

- If a command to connect a certificate to a keyring is attempted and the certificate isn't yet defined in RACF, that command will fail.
- Since we manually created the controller ID, the line from `BBOWBRAK` that tried to do that again simply failed with a duplicate error. That was okay.

**Run the rest of the customized jobs to create Standalone Server**

Standard procedure here ... submit the job, look for RC=0 ... submit the next, etc.

**Started Standalone Server**

A Standalone Server must be started at least once before it will be allowed to be federated into a Deployment Manager cell. This was also a good way to validate the proper configuration of the server.

Once it successfully started once, it was stopped.

**Federate z/OS node into xLinux DMGR**

The ISPF panels for federating a Standalone Server were run and a customized `BBOWADDN` job was created. The host and port information for the DMGR running on the xLinux box was provided so a SOAP connection could be made to the DMGR to process the federation request.
Recall that global security was on for the Deployment Manager. This meant several things:

- The customization panel for the federation job had to be informed that global security was on, and the Admin ID and password had to be supplied.

- The BBOWADDN job had to be run under the Admin ID, not a UID=0 ID.

The first bullet is what provides the ability to authenticate into the SOAP port of the DMGR. The second bullet is what allows the batch job process to establish an SSL connection to the DMGR. In order to establish that SSL connection, the ID under which the federation job runs must have the public key certificate of the CA that signed the DMGR certificate.

The BRAK job for the Standalone Server would have connected that CA certificate -- which we imported from WSCPLEX -- to the WebSphere Admin ID. Another ID may not have that certificate. That's why it's a documented requirement to use the WebSphere Admin ID.

The act of federation created the Node Agent and automatically started the Node Agent. The global security settings were automatically propagated to the node.

**Status of cell**

The heterogeneous cell was complete:
Appendix A - Miscellaneous Updates and Information

Notes on cross-Sysplex cells and SAF REALM names in 6.1

Prior to WebSphere 6.1, it was only possible to create a heterogeneous cell that spanned two or more Sysplexes if the cell used LDAP as its registry. In WebSphere 6.1, it's possible to use LocalOS for a cross-Sysplex cell, subject to numerous caveats. The reason this is possible is because the value of the realm name can now be controlled.

What is the realm name?

When a user accesses a protected application, authentication of the user is driven by WebSphere. If authentication is successful, the user's browser receives an LTPA token in the form of a cookie. The LTPA token contains the user's credentials, and is how WebSphere recognizes the user as the user clicks through the pages of the application.

Among the information contained in the LTPA token is the name of the registry used by the WebSphere cell. This registry name is referred as the realm name for the cell.

Once authenticated, when a user makes a request of a server, the user's LTPA token is forwarded by the browser to the server. When examining the LTPA token, one of the things the server checks is whether the realm name in the user's LTPA token matches the realm name used by the server. If they don't match, the server rejects the LTPA token and prompts the user to login.

Servers use LTPA tokens to authenticate one another in an ND cell. A Deployment Manager will pass its LTPA token along with administrative requests to the Node Agent, for example. If the LTPA token is rejected, administrative functions between the servers will fail.

It is essential that all of the servers in a cell use the same realm name. This is true whether the cell is distributed or not.

Where does the value of the realm name come from?

If a cell uses an LDAP registry, the LDAP server’s host name:port is used as the realm name. The LDAP server’s host name:port is specified in the admin console when the cell is configured to use LDAP, and the value of the realm name is established as a cell-wide WebSphere property. Since the choice of an LDAP registry is cell-wide, all the servers in the cell use the same LDAP registry, and the cell has a consistent view of the user registry and the realm name.

If the cell uses LocalOS (SAF) as the user registry, how WebSphere determines the value of the realm name depends upon the level of WebSphere.

Before WebSphere for z/OS V6.1, the Daemon IP name was used as the realm name.

In WebSphere 6.1, the realm name is determined from the APPLDATA segment of the SAFDFLT profile in the RACF REALM class. If that profile is not defined, or has no APPLDATA value, WebSphere determines the realm name using the previous method.

Considerations

What are the considerations for creating a cross-Sysplex heterogeneous cell using LocalOS as the registry?

1. The most important consideration is that the WebSphere cell must have a common view of security on all systems.

This means that although the cell might be spread across multiple Sysplexes, using multiple RACF security databases, a common set of users, groups and resources (e.g. EJBROLES, certificates and keyrings) must be maintained by the security administrator.
This is probably easier said than done. For instance, the RACF Remote Sharing Facility (RRSF) can be used to synchronize most RACF profiles across Sysplexes, but RRSF does not propagate all changes. Most notably, RRSF does not propagate digital certificate private keys. So if a certificate and private key is created on one Sysplex, the certificate will be propagated to the other RACF databases but the private key will not. For this reason, certificates and their private keys are best exported from one system and added to the others, just as you would do with a cross-platform heterogeneous cell.

2. Before a server is federated into a cell, the LTPA key must be exported from one server, ideally the Deployment Manager, and imported into the server to be federated, just as you would do with a cross-platform heterogeneous cell.

3. Before a server is federated into a cell, the realm name must be set to a common value across the cell and the server to be federated. By default, WebSphere uses the daemon IP name is used as the realm name. If the REALM class is active in RACF, WebSphere V6.1 will use the value of the APPLDATA segment in the REALM class profile SAFDFLT.

Example: the following RACF commands activate the REALM class and set the realm name to WAS.REALMNME

```
SETR CLASSACT(REALM) RACLIST(REALM)
RDEFINE REALM SAFDFLT APPLDATA('WAS.REALMNME') OWNER(USER0) UACC(READ)
SETR RACLIST(REALM) REFRESH
```

Remember that this profile must be identically defined in each RACF database that the cell spans. Keep in mind that once the SAFDFLT profile is defined, all WebSphere for z/OS V6.1 servers configured for LocalOS in that Sysplex will use that profile to determine their realm name.
Document Change History

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 14, 2005</td>
<td>Original document.</td>
</tr>
<tr>
<td>September 16, 2005</td>
<td>Number WP100644 assigned to document.</td>
</tr>
<tr>
<td>September 19, 2005</td>
<td>Added a few clarifications, based on a review by Mike Cox:</td>
</tr>
<tr>
<td></td>
<td>- Emphasized required that DMGR node <em>must</em> be equal or higher service level.</td>
</tr>
<tr>
<td></td>
<td>- Emphasized minimum service level is 6.0.2</td>
</tr>
<tr>
<td></td>
<td>- Added a rather significant section on Node Group short names, and the requirement to maintain their uniqueness. See &quot;Node Group short name&quot; on page 13.</td>
</tr>
<tr>
<td></td>
<td>- Added a section on setting the ORB character encoding of the distributed platform box to ISO8859-1 so it would be compatible with the z/OS platform. See &quot;ORB character encoding&quot; on page 27.</td>
</tr>
<tr>
<td>September 12, 2007</td>
<td>Added &quot;Appendix A&quot; to incorporate information on SAF REALM names in WebSphere for z/OS Version 6.1.</td>
</tr>
</tbody>
</table>

End of WP100644