WebSphere Application Server V7: System Management Technical Overview

In this chapter, we provide a technical overview of the system management functionality of WebSphere Application Server. This information can help you understand how system administration occurs. It is particularly useful in a multi-server environment to understand the distributed administration and synchronization topics.

This chapter assumes that you are familiar with the concepts in WebSphere Application Server V7.0: Technical Overview, REDP-4482.

We cover the following topics:

- “System management overview” on page 2
- “System management in a standalone server environment” on page 6
- “System management of multiple standalone servers” on page 7
- “System management in a distributed server environment” on page 10
- “Management of distributed and standalone servers” on page 29
- “Java Management Extensions (JMX)” on page 32
- “Centralized Installation Manager (CIM)” on page 34
- “IBM Support Assistant (ISA) V4” on page 34
System management overview

At first glance, system management concepts in WebSphere® Application Server might seem complex. However, the fact that the system management architecture is based on Java Management Extensions (JMX™) and the fact that WebSphere Application Server provides easy-to-use administration tools, both make it fairly simple to use and understand.

Terminology

There are differences in how WebSphere Application Server handles administration, depending on the environment that you have set up. As you go through this book, you can see the following terms used:

- **Standalone server environment** refers to a single standalone server that is not managed as part of a cell. With the Base and Express packages, this is your only option. You can also create a standalone server with the Network Deployment package.

- **Distributed server environment** refers to the situation where you have multiple servers managed from a single deployment manager in the cell. We also refer to these as *managed servers*. This is only valid with the Network Deployment package.

- **Application server** refers to the process that provides the functions that are required to support and host user applications. An application server can be a *standalone application server*, a *distributed application server* managed by a deployment manager, or an *unfederated application server* that is managed from an administrative agent.

- **Managed processes** refer to the deployment manager, nodes (node agents), and application servers.

- **Flexible management** refers to asynchronous job management through the job manager (available in Network Deployment only) and managing multiple unfederated application servers from an administrative agent (available on all packages).

- **System** refers to one physical computer.

Directory conventions

Throughout this book, we use the following notations when indicating the location of files and commands:

- **install_root** is used to denote the installation directory for a product. The default installation directory locations can be found in:
Profiles

To create an application server, node agent, deployment manager, administrative agent, or a job manager, you must first install the WebSphere Application Server core product files and then create their respective profiles. This is illustrated in Figure 1.

Figure 1  Product files and profiles

When a profile is created, the configuration details for the server are stored in a folder unique to the profile. You can think of the product files as the runtime component, and the profiles as the input for the runtime.
There are seven profile types that can be created, as shown in Figure 2.

Notice that a server always belongs to a node, and the node always belongs to a cell. The purpose of the node and cell is to define the administrative domain. The node and cell are logical concepts used to group servers, and to define their scope. Every time a profile is created, a node is created. Therefore, there is a one to one relationship between a profile and a node.
System management tools

WebSphere Application Server V7.0 provides a variety of administrative tools for configuring and managing your runtime environment. The choice of which combination of administrative tools you want to employ depends on the size and complexity of your runtime environment:

▶ Integrated Solutions Console, referred to as the administrative console:

The administrative console is a browser-based client that uses a Web application running in the Web container to administer WebSphere Application Server.

The administrative console uses Simple Object Access Protocol (SOAP) via HTTP or HTTPS to connect to the admin application in the application server or deployment manager. The admin application connects to the administrative service, which is responsible for modifying the configuration repository using JMX managed beans (MBeans).

▶ WebSphere scripting client (wsadmin):

The wsadmin client is a non-graphical scripting interface that can be used to administer WebSphere Application Server from a command line prompt. It provides multiple options to connect to the Mbeans: SOAP, RMI, JSR160RMI, IPC, and NONE. Note that the RMI connection option is deprecated. The default connection type is SOAP.

The NONE connection type connects directly to the configuration files bypassing the application server. The NONE option allows the administrator to administer the application server even if the application server is not started, or if the embedded HTTP Server or administrative service is not responding. When using the NONE option, the administrator must be on the same system as the application server.

▶ Task automation with ANT:

With Another Neat Tool (ANT), you can create build scripts that compile, package, install, and test your application on WebSphere Application Server.

▶ Administrative applications:

You can develop custom Java™ applications that use the Java Management Extensions (JMX) based on the WebSphere Application Programming Interface (API).
Command line utilities:

WebSphere Application Server provides administrative utilities to help manage your environment. These utilities are called from a command line. They can be used to perform common administrative tasks such as starting and stopping WebSphere Application Server, backing up the configuration, and so on. Command line utilities work on local processes (application servers, nodes, and the deployment manager) only.

There are multiple levels of administration in WebSphere Application Server:

- In the WebSphere Application Server Express and Base packages, you can administer standalone server instances individually.
- In the WebSphere Application Server Express and Base packages, you can administer multiple standalone server instances on a single system using an administrative agent.
- In the WebSphere Application Server Network Deployment package, you can administer an entire cell of application servers using a deployment manager.
- You can administer multiple standalone application servers, administrative agents, and deployment managers using a job manager.

System management in a standalone server environment

A standalone application server provides the necessary capabilities to run Java 2 Platform, Enterprise Edition (J2EE™) compliant applications. A standalone application server is a good starting point for development and test teams. It can also be used for proof of concept or lightweight applications that do not require intensive system resources.

To create a standalone application server, you must create an Application Server profile. The profile defines the application server, node, and cell.

The application server can be administered using the Web-based administrative console, command line utilities, and wsadmin. The administrative console and wsadmin provide remote administration access. All of the configuration data for an application server, including the installed applications, is stored in a configuration repository created when the profile is created.
Figure 3 shows the components of a standalone application server environment.

**System management of multiple standalone servers**

Based on their business requirements, an organization can have multiple standalone application servers installed on the same system or on multiple systems. These servers might be used for development environments, unit testing, regression testing, staging environment, and so forth.

When there are multiple standalone application servers installed, administration of each application server can become difficult. Each application server runs its own administrative service as well as the administrative console application. An administrator might have to juggle multiple consoles. And as the number of standalone servers increases on a given system, the system resources used for administrative functions increase.
An environment with independently managed application servers is illustrated in Figure 4.

With this infrastructure, the administrator must know which administrative port is used for each standalone server. Another drawback to managing standalone servers is that when the server is not running the administrative application and administrative services are not available, to start the application server, or make any changes, the application servers must be locally started.

The administrative agent reduces the problems related to managing multiple standalone servers by providing a centralized point of control for all of the standalone application servers within a system. The administrative agent is responsible for running the administrative application and administrative service to manage all of the servers on the registered node within the given system. This environment is illustrated in Figure 5 on page 9.
After a node containing a standalone server is registered with the administrative agent, the administrative console application and administrative service are stopped on the application server. System resources are efficiently utilized, because the administrative functionality is not enabled on multiple standalone servers. The standalone application server resources are dedicated to running applications.

Another key feature is the ability to administer an application server when it is not running. The administrative agent modifies the standalone server's configuration repository directly using the administrative service. The administrative agent can also start, stop, and create new servers within the managed node.
System management in a distributed server environment

Although a standalone application server is sufficient to run any J2EE compliant application, there are many runtime capabilities that are not addressed, such as what happens when an application server crashes, or needs to be restarted, or if the demand on the application exceeds the resources available on the hardware. A distributed server environment allows the developer to concentrate on solving the business problem at hand, while leaving infrastructure problems to WebSphere Application Server Network Deployment.

A distributed server environment provides the capabilities to create clusters of application servers. Clustering servers provides workload balancing, session data replication, and failover. The application servers are centrally managed using the deployment manager.

To build a distributed server environment, you start by creating a deployment manager profile. The deployment manager is responsible for administering the entire cell. The concept of cells and nodes becomes very important in a distributed server environment. A deployment manager administers one and only one cell.

New in V7: Without the administrative agent, to create a standalone server, you must create an application server profile, which introduces a new cell, a new node, and an increased number of files. The capability to create multiple application servers within an unmanaged node using the administrative agent is a new feature introduced in V7.

The administrative agent, along with multiple standalone servers, is a great starting point for simplifying administration. However, note that features such as failover, workload management, session data replication, and many other features cannot be configured without a distributed server environment.
After the deployment manager is created, the next step is to create a custom profile. This creates a second cell (defaultCell), a node, and a node agent. At this point, you do not have a functioning application server environment. Figure 6 illustrates this temporary stage of the environment.

The next step is to federate the node (NodeA in Figure 6) to the deployment manager’s cell by using the `addNode` command. After being federated, NodeA is no longer part of the defaultCell, but rather it is part of the deployment manager's cell (dmgrCell).

Now that NodeA has been federated, all administration of NodeA is delegated to the deployment manager, and new application servers can be created on the node using the administrative tools for the deployment manager.

This environment is shown in Figure 7 on page 12.

For design considerations such as scalability, caching, high availability, load balancing, failover, disaster recovery security, and serviceability, see *WebSphere Application Server V7: Concepts, Planning and Design, SG24-7708*. 
Centralized changes to configuration and application data

The deployment manager maintains a master repository of all the configuration files. When configuration changes are made at the deployment manager, the changes are first stored in the master repository. Automatic or manual synchronization pushes the changes down to the affected nodes. Synchronization is discussed further in “File synchronization in distributed server environments” on page 23.
Rules for process startup

When a managed server begins its startup, it sends a discovery request message that allows other processes to discover its existence and establish communication channels with the process. This makes it possible to start the processes in a distributed server environment without following a strict order for startup. For example:

- A node agent can be running while the deployment manager is not, and the reverse is true. When the stopped process is started, discovery occurs automatically.
- The deployment manager can be running while a managed server is not, and the reverse is true. The execution of a managed server is not dependent on the presence of a running deployment manager. The deployment manager is only required for permanent configuration changes written to the master repository.

The only rule to remember is that the node agent should be started before any application servers on that node. The node agent contains the Location Service Daemon (LSD) in which each application server registers on startup. However, the node agent is purely an administrative agent and is not involved in application serving functions. Each managed server has the data necessary to start itself.

Example discovery scenarios
In this section, we describe two typical situations that might occur:

**Situation:** The node agent is not running and the deployment manager starts:

1. The deployment manager tries to determine if the node agent is running. The process fails.
2. When the node agent is started, it contacts the deployment manager, creates a communication channel, and synchronizes data.

**Situation:** The node agent starts but no managed servers are started:

1. The node agent knows all about its managed servers and checks whether they are started. If so, it creates communication channels to these processes.
2. When a managed server starts, it checks whether the node agent is started and then creates a communication channel to it.
Distributed process discovery

**Deep-dive:** The distributed process discovery section provides in-depth knowledge that can be useful when debugging or testing, but it is not necessary when trying to understand the overall architecture.

Figure 8 shows an example of the distributed discovery process for a topology containing two nodes that are located on different machines. Note that both node agents in the figure use ports 7272 and 5000. This assumes they reside on separate physical machines. If nodes are located on the same machine, they must be configured to use non-conflicting IP ports. The profile wizard selects non-conflicting ports for you automatically.

```
serverindex.xml
  serverType="Deployment_Manager"
    ... CELL_DISCOVERY_ADDRESS .. port:7277

serverType="Node_Agent"
  ... NODE_DISCOVERY_ADDRESS.. port 7272
  ... NODE_MULTICAST_DISCOVERY_ADDRESS .. port 5000
```

*Figure 8  Distributed discovery process*

Each node agent and deployment manager maintains status and configuration information by using discovery addresses, or *ports*. On startup, processes discover other running components, and create communication channels between them, through the discovery addresses.
During discovery, the following actions occur:

- The master repository located on the deployment manager installation contains the serverindex.xml file for each node. The deployment manager reads this file on startup to determine the host name and IP port of each node agent's NODE_DISCOVERY_ADDRESS.

  The default port for the NODE_DISCOVERY_ADDRESS is 7272. You can verify this by looking at the NODE_AGENT stanza in the serverindex.xml file of each node located at:

  
  dmgr_profile_root/config/cells/cell_name/nodes/node_name/serverindex.xml

  You can also display this port from the administrative console by selecting System Administration → Node agents. Select each node agent and expand Ports under the Additional Properties section.

- The copy of the configuration repository located on each node contains the serverindex.xml file for the deployment manager. The node agent reads this file on startup to determine the host name and IP port of the deployment manager’s CELL_DISCOVERY_ADDRESS.

  The default port for the CELL_DISCOVERY_ADDRESS is port 7277. You can verify this by looking at the DEPLOYMENT_MANAGER stanza in the serverindex.xml file for the deployment manager node located at:

  
  profile_root/config/cells/cell_name/nodes/dmgr_node_name/serverindex.xml

  You can also display this port from the administrative console by selecting System Administration → Deployment manager. Expand Ports under the Additional Properties section.

- The copy of the configuration repository located on each node also contains the serverindex.xml file for the node. Each managed server reads this file on startup to determine the host name and IP port of the node agent's NODE_MULTICAST_DISCOVERY_ADDRESS.

  A multicast address is used to prevent the usage of a large number of IP ports for managed server to node agent discovery requests. Using multicast, a node agent can listen on a single IP port for any number of local servers.

  The default port for the NODE_MULTICAST_DISCOVERY_ADDRESS is 5000. You can verify this by looking at the NODE_AGENT stanza in the serverindex.xml file of the node located at:

  
  profile_root/config/cells/cell_name/nodes/node_name/serverindex.xml

  You can also display this port from the administrative console by selecting System Administration → Node agents. Select the node agent and expand Ports under the Additional Properties section.
Each server has its own copy of the configuration and application data necessary for startup of the runtime and the installed applications.

Configuration and application data repository

The configuration and application data repository is a collection of files containing all the information necessary to configure and execute servers and their applications. Configuration files are stored in XML format, while application data is stored as enterprise archive (EAR) files and deployment descriptors.

Repository directory structure

It is important to know that configuration files defining a runtime environment are stored in profile directories. Each node containing a deployment manager, application server, administrative agent, or job manager has its own profile directory under the install_root/profiles directory.
The repository files are arranged in a set of cascading directories under each profile directory structure, with each directory containing a number of files relating to different components of the cell. You can see this arrangement in Figure 9. The repository structure follows the same format, regardless of whether you have a standalone server environment or a distributed server environment.

**Terminology:** In the rest of this book, when we talk about a specific profile directory, located at, `install_root/profiles/profile_name`, we refer to it as the `profile_root` directory.

When we are speaking of a profile directory for a specific profile, we use the following terms:

- **Deployment manager profile:** `dmgr_profile_root`
- **Administrative agent profile:** `adminAgnt_profile_root`
- **Job manager profile:** `jmgr_profile_root`

The repository files are arranged in a set of cascading directories under each profile directory structure, with each directory containing a number of files relating to different components of the cell. You can see this arrangement in Figure 9. The repository structure follows the same format, regardless of whether you have a standalone server environment or a distributed server environment.

**Figure 9  Repository directory structure**
The profile_root/config directory is the root of the repository for each profile. It contains the following directory structure:

- cells/cell_name/
  This is the root level of the configuration for the cell. The directory contains a number of cell-level configuration files. Depending on the types of resources that have been configured, you might see the following subdirectories:
  - cells/cell_name/applications/ contains one subdirectory for every application that has been deployed within the cell.
  - cells/cell_name/buses/ contains one directory for each service integration bus (bus) defined.
  - cells/cell_name/coregroups/ contains one directory for each core group defined.
  - cells/cell_name/nodegroups/ contains one directory for each node group defined.
  - cells/cell_name/nodes/ contains the configuration settings for all nodes and servers managed as part of this cell. The directory contains one directory per node. Each cells/cell_name/nodes/<node> directory contains node-specific configuration files and a server directory that in turn contains one directory per server and node agent on that node.
  - cells/cell_name/clusters/ contains one directory for each of the clusters managed as part of this cell. Each cluster directory contains a single file, cluster.xml, which defines the application servers of one or more nodes that are members of the cluster.

The overall structure of the master repository is the same for both a standalone server environment and a distributed server environment. The differences are summarized in the following sections.

In a standalone server environment, the structure has the following characteristics:

- The master repository is held on a single machine. There are no copies of this specific repository on any other node.
- The repository contains a single cell and node.
- There is no node agent, because each application server is standalone, so there is no directory for the node agent (nodeagent).
- Clusters are not supported, and therefore, the structure does not contain the cluster’s directory or subdirectories.
In a distributed server environment, the structure has the following characteristics:

- The master repository is held on the node containing the deployment manager. It contains the master copies of the configuration and application data files for all nodes and servers in the cell.
- Each node also has a local copy of the configuration and application data files from the master repository that are relevant to the node.
- When changes are made to the configuration in the master repository, those changes need to be synchronized to the configuration files on the nodes.
- Changes can be made to the configuration files on a node, but the changes are temporary and are overwritten by the next file synchronization from the deployment manager. Permanent changes to the configuration require changes to the file or files in the master repository. Configuration changes made to node repositories are not propagated up to the cell.
- The applications directory of the master repository contains the application data (binaries and deployment descriptors) for all applications deployed in the cell. The local copy of the applications directory on each node only contains the directories and files for the applications deployed on application servers within that node.

Information about the individual files in each of these directories can be found in the topic, "Configuration Document Descriptions," at this Web site:


**Variable scoped files**

Identically named files that exist at differing levels of the configuration hierarchy are termed *variable scoped* files. There are two uses for variable scoped files:

- Configuration data contained in a document at one level is logically combined with data from documents at other levels of the configuration hierarchy. In the case of conflicting definitions, the “most specific” value takes precedence. For example, if an identical entry exists in the files at the cell and node level (as with a variable defined in both the cell and node's variables.xml), the entry at the node level takes precedence.

- Documents representing data that is not merged but is rather scoped to a specific level of the topology. For example, the namestore.xml document at the cell level contains the cell persistent portion of the name space, while the namestore.xml document at the node level contains the node persistent root of the name space.
Application data files

The master repository is also used to store the application binaries (EAR files) and deployment descriptors. The `profile_root/config` directory of the master repository contains the following directory structure used to hold application binaries and deployment settings:

- `cells/cell_name/applications/`

  This directory contains a subdirectory for each application deployed in the cell. Names of the directories match the names of the deployed applications.

**Note:** The name of the deployed application does not have to match the name of the original EAR file used to install it. Any name can be chosen when deploying a new application, as long as the name is unique across all applications in the cell.

- `cells/cell_name/applications/app_name.ear`

  Each application’s directory in the master repository contains these components:

  - A copy of the original EAR, called `app_name.ear`, which does not contain any of the bindings specified during the installation of the application
  - A deployments directory, which contains a single `app_name` directory used to contain the deployed application configuration

- `cells/cell_name/applications/app_name.ear/deployments/app_name`

  The deployment descriptors in this directory contain the bindings specified during application deployment. The deployment directory of each application contains these files:

  - `deployment.xml`

    This file contains configuration data for the application deployment, including the allocation of application modules to application servers, and the module startup order.
- META-INF/

This directory contains these files:

- application.xml - J2EE standard application deployment descriptor
- ibm-application-bnd.xmi - IBM® WebSphere-specific application bindings
- ibm-application-ext.xmi - IBM WebSphere-specific application extensions
- was.policy - Application-specific Java 2 security configuration

This file is optional. If not present, the policy files defined at the node level apply for the application.

The subdirectories for all application modules (Web archive files (WARs) and Enterprise JavaBeans (EJB™) Java archive files (JARs)) are contained in the was.policy file along with each module's deployment descriptors. The subdirectories for each module do not contain application binaries (JARs, classes, and JavaServer Pages (JSPs)), only deployment descriptors and other configuration files.

The installation of an application onto a WebSphere Application Server application server results in these actions:

- The storage of the application binaries (EAR) and deployment descriptors within the master repository.
- The publishing of the application binaries and deployment descriptors to each node that will be hosting the application. These files are stored in the local copy of the repository on each node.

Each node then installs applications ready for execution by exploding the EARs under the profile_root/installedApps/cell_name/ directory in the following manner:

- profile_root/installedApps/cell_name/

This directory contains a subdirectory for each application deployed to the local node. The name of each application's directory reflects the name under which the application is installed, not the name of the original EAR. For example, if an application is called myapp, the installedApps/cell_name directory will contain a myapp.ear subdirectory.
profile_root/installedApps/cell_name/app_name.ear/

Each application-specific directory contains the contents of the original EAR used to install the application:

- The deployment descriptors from the original EAR. These descriptors do not contain any of the bindings specified during application deployment.
- All application binaries (JARs, classes, and JSPs).

Figure 10 summarizes how the node’s local copy of the repository contains the application’s installed deployment descriptors, while the directory under installedApps contains the application binaries.
By default, a WebSphere Application Server application server executes an application by performing the following tasks:

1. **Loading the application binaries stored under:**
   
   
   
   
   `profile_root/installedApps/cell_name/app_name.ear/`
   
   You can change this location by altering the Application binaries setting for the enterprise application or by altering the $(APP_INSTALL_ROOT) variable setting.

2. **Configuring the application using the deployment descriptors stored under:**
   
   
   
   `profile_root/config/cells/cell_name/applications/app_name.ear/deployments/app_name`
   
   You can change this by modifying the Use configuration information in binary setting for the enterprise application. This is the Use Binary Configuration field on the application installation and update wizards.

   By default, the setting is not enabled. Enabling it specifies that you want the application server to use the binding, extensions, and deployment descriptors located in the application EAR file rather than those stored in the deployments directory.

**File synchronization in distributed server environments**

The file synchronization service is the administrative service responsible for keeping the configuration and application data files up-to-date that are distributed across the cell. The service runs in the deployment manager and node agents, and ensures that changes made to the master repository are propagated out to the nodes, as necessary. The file transfer system application is used for the synchronization process. File synchronization can be forced from an administration client, or can be scheduled to happen automatically.

During the synchronization operation, the node agent checks with the deployment manager to see if any files that apply to the node have been updated in the master repository. New or updated files are sent to the node, while any deleted files are also deleted from the node.

Synchronization is one-way. The changes are sent from the deployment manager to the node agent. No changes are sent from the node agent back to the deployment manager.
Synchronization scheduling
The scheduling of file synchronization is configured using an administrative client. There are two available options:

- **Automatic synchronization:**
  Synchronization can be made to operate automatically by configuring the file synchronization service of the node agent. The settings allow you to:
  - Enable periodic synchronization to occur at a specified time interval
    By default, this option is enabled with a time interval of one minute.
  - Enable synchronization at server startup
    The synchronization occurs before the node agent starts a server. Note that if you start a server using the `startServer` command, this setting has no effect.

- **Explicit/forced synchronization:**
  Synchronization can be explicitly forced at any time via use of an administrative client.

**Tip:** In a production environment, the automatic synchronization interval needs to be increased from the one minute default so that processing and network overhead is reduced.

How files are identified for synchronization

**Deep-dive:** This section provides in-depth knowledge that can be useful when debugging or testing, but it is not necessary when trying to understand the overall architecture.

When synchronization occurs, WebSphere must be able to identify the files that have changed and therefore need to be synchronized. To do this, WebSphere uses the following scheme:

- A calculated digest is kept by both the node agent and the deployment manager for each file in the configuration they manage. These digest values are stored in memory. If the digest for a file is recalculated and it does not match the digest stored in memory, this indicates the file has changed.

- An *epoch* for each folder in the repository and one for the overall repository are also stored in memory. These epochs are used to determine whether any files in the directory have changed. When a configuration file is altered through one of the WebSphere administration interfaces, then the overall repository epoch and the epoch for the folder in which that file resides are modified.
Manually updating a configuration file does not cause the digest to change. Only files updated with administration clients are marked as changed. Manually updating the files is not recommended, but if you do, a forced synchronization will include manually updated files.

- During configuration synchronization operations, if the repository epoch has changed since the previous synchronize operation, individual folder epochs are compared. If the epochs for corresponding node and cell directories do not match, the digests for all files in the directory are recalculated, including that changed file.

**Ensuring that manual changes are synchronized**

Manually changing configuration files is not recommended. Only change configuration files manually as a diagnostic measure or on the rare occasion that you need to modify a configuration setting that is not exposed by the administration clients.

The following topic lists several configuration files that have settings not exposed in the administration clients. In the event that you find it necessary to edit a file manually, this information can help make sure that you do not lose your changes. Refer to *Configuration Document Descriptions* in the Information Center at this Web site:


Manual editing has several drawbacks, including these:

- When using **wsadmin** and the administrative console, you have the benefit of a validation process before the changes are applied. With manual editing, you have no such fail-safe.

- Updates made manually are not marked for synchronization and will be lost at the next synchronization process unless you make them in the master repository and manually force synchronization.

If a change to a configuration file is made by editing the file, the digest for the file is not recalculated, because epochs for the directories continue to match and the synchronization process does not recognize that the files have changed.

However, manual edits of configuration files in the master cell repository can be picked up if the repository is reset so that it rereads all the files and recalculates all of the digests. You can reset either the master cell repository epoch or the node repository epoch:

- Resetting the master cell repository causes any manual changes made in the master configuration repository to be replicated to the nodes where the file is applicable.
Resetting the node repository causes any manual changes to the local node files to be overwritten by whatever is in the master cell repository, regardless of whether the cell repository was changed or not. Any manual changes in the master repository will be picked up and brought down to the node.

The main difference between cell reset and node reset is that cell reset is likely to impact the entire cell, not just one node.

This holds true for changes to installed applications as well. They are treated the same as other configuration files in the repository. For each installed application, there is an EAR file in the repository and configuration files associated with the deployment of the application.

If you manually change the EAR file and reset the master cell repository, the changed EAR file is replicated out to the nodes where it is configured to be served and is expanded in the appropriate location on that node for the application server to find it. The application on that node is stopped and restarted automatically so that whatever is changed is picked up and made available in the application server.

**Important:** Manually changing the EAR file is best performed by advanced users. Otherwise, unpredictable results can occur.

If you manually edit one of the deployment configuration files for the application and reset the repository, that change is replicated to the applicable nodes and is picked up the next time the application on that node is restarted.

**Resetting the master cell repository**

**Note:** The use of `wsadmin` is covered in Chapter 8, *Administration with scripting*. The only thing that you might need to know about `wsadmin` to complete these tasks is to start `wsadmin` on the SOAP connector port of the process on which you want to run the commands. The default is to start to port 8879. If the process you are connecting to has a different port number specified, start `wsadmin` with the `-port` argument.

To perform a reset of the master cell repository, perform the following actions:

1. Open a command prompt and change to the `dmgr_profile_root\bin` directory and start a `wsadmin` session. Note that the deployment manager must be running. Use the following command:

```
cd dmgr_profile_root\bin
wsadmin
```
2. Enter the following statements:

```
wsadmin> set config [AdminControl queryNames *:*.,type=ConfigRepository,process=dmgr]

wsadmin> AdminControl invoke $config refreshRepositoryEpoch
```

You can see a number returned by the refreshRepositoryEpoch operation, for example, 1047961605195, as shown in Example 1.

**Example 1  Resetting the master cell repository**

```
dmgr_profile_root\bin> wsadmin
WASX7209I: Connected to process "dmgr" on node DmgrNode using SOAP connector;  The type of process is: DeploymentManager
WASX7029I: For help, enter: "$Help help"

wsadmin> set config [AdminControl queryNames *:*.,type=ConfigRepository,process=dmgr]

WebSphere:name=repository,process=dmgr,platform=common,node=DmgrNode,version=5.0,type=ConfigRepository,mbeanIdentifier=repository,cell=DmgrCell,spec=1.0

wsadmin> AdminControl invoke $config refreshRepositoryEpoch
1237317922687
wsadmin>
```

This resets the entire cell repository digest set. On the next synchronize operation, all files in the master cell repository will have their digests recalculated. Any manual changes will be replicated to the applicable nodes.

**Resetting the node repository**

There are multiple ways to reset a node repository for synchronization:

- In a *wsadmin* session connected to the deployment manager or node agent, enter the following commands:

```
wsadmin> set config [AdminControl queryNames *:*.,type=ConfigRepository,process=nodeagent]

wsadmin> AdminControl invoke $config refreshRepositoryEpoch
```

This resets the node digest set. Any file that does not match what is in the repository is overwritten.

Example 2 gives an overview of resetting the node repository.
**Example 2  Resetting the node repository**

```bash
profile_root\bin>wsadmin -port 8883
```

WASX7209I: Connected to process "nodeagent" on node AppSrvrNode using SOAP connector; The type of process is: NodeAgent
WASX7029I: For help, enter: "$Help help"

wsadmin>set config [$AdminControl queryNames */*,type=ConfigRepository,process=nodeagent]
WebSphere:name=repository,process=nodeagent,platform=common,node=AppSrvrNode,version=5.0,type=ConfigRepository,mbeanIdentifier=repository,cell=DmgrCell

wsadmin>$$AdminControl invoke $config refreshRepositoryEpoch 1237319314359

| From the deployment manager administrative console, select **System Administration → Nodes** to see a list of the nodes in the cell. Notice the Synchronize and Full Resynchronize buttons on the page. The Synchronize button causes a normal synchronize operation with no rereading of the files. The Full Resynchronize button is the reset and recalculate function. Select the node or nodes to be updated with manual changes, then click the **Full Resynchronize** button.
| Use the **syncNode** command. This command is a standalone program that runs separately from the node agent. It has no cache of epoch values that could be used for an optimized synchronization, therefore it performs a complete synchronization. For this same reason, if you restart a node agent, the very first synchronization that it performs will always be a complete synchronization. Note that this action requires the node agent to be stopped.

The **syncNode** command resides in the `bin` directory of the base install. To use the **syncNode** command, type these commands from the command line:

```bash
cd profile_root\bin
csyncNode cell_host
```

Example 3 shows the use of the **syncNode** command.

**Example 3  Using the syncNode command**

```bash
profile_root\bin>stopnode
ADMU0116I: Tool information is being logged in file profile_root\logs\nodeagent\stopServer.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU3100I: Reading configuration for server: nodeagent
ADMU3201I: Server stop request issued. Waiting for stop status.
```
ADMU4000I: Server nodeagent stop completed.

profile_root\bin>syncnode sysvm2
ADMU0116I: Tool information is being logged in file
profile_root\logs\syncNode.log
ADMU0128I: Starting tool with the AppSrv01 profile
ADMU0401I: Begin syncNode operation for node AppSrvrNode with
Deployment Manager sysvm2: 8879
ADMU0016I: Synchronizing configuration between node and cell.
ADMU0402I: The configuration for node AppSrvrNode has been synchronized
with Deployment Manager sysvm2: 8879

The repository is flexible in that there is no predefined list of document types that
it permits. You can add any file you want. Perhaps you have some unique
configuration data that needs to be used on all nodes. You can put it in the
config/cells/cell_name folder and it will be synchronized to all nodes. If it
applies to just one node, you can put it in the folder corresponding to that node
and it will be synchronized only to that node. The same applies for any additional
documents in a server level folder.

For example, under normal circumstances, all application files are packaged in
the EAR file for the application. However, consider a configuration file specific to
an application. Any changes to that file will require that you update the EAR file
and synchronize the entire application.

One possibility is to put a properties file in the application deployment directory in
the master configuration repository, so that it is replicated automatically to all
nodes where the application is installed but the entire EAR is not replicated. Then
you can have an ExtensionMBean update the properties file in the master
repository and normal synchronization will replicate just those changes out to the
nodes without the need to synchronize the whole EAR and restart the
application.

Management of distributed and standalone servers

It is possible to encounter a scenario where there might be multiple distributed
environments, each managed by their own deployment manager. With multiple
deployment managers, they must be administered individually and there is no
way of coordinating management actions between the different distributed
environments. Distributed environment administration performance is affected by
low latency networks because file synchronization between the deployment
manager and node agent are dependent on network communication.
The job manager can be used to administer multiple distributed environments as well as standalone servers. The job manager administers the environment asynchronously using the concept of jobs. Because jobs are submitted asynchronously, a low latency network is sufficient. This is can be useful when the environment is distributed over distant geographical areas.

The job manager is only available with WebSphere Application Server Network Deployment and WebSphere Application Server for z/OS.

To administer a distributed environment, the deployment manager is registered with the job manager. To administer standalone servers, the nodes managed by the administrative agent are registered with the job manager. This relationship between the job manager and the environments it can interact with is shown in Figure 11.

![Figure 11 Flexible management](image)

The job manager administers the registered environments by submitting jobs that perform tasks, for example:

- Start and stop servers
- Create and delete servers
- Install and uninstall applications
Start and stop applications
- Run wsadmin scripts
- Distribute files

The job manager has a repository for its own configuration files, which are related to security, administration of job manager, configurations, and so on; however, it does not maintain a master repository the way a deployment manager does. Rather, the job manager allows the administrative agents and deployment managers to continue managing their environment as they would have had they not been registered with the job manager. The job manager simply provides another point of administration. This is illustrated in Figure 12.

![Diagram](image1.png)

**Figure 12  Job manager administration environment**

The job manager can administer multiple administrative agents and deployment managers. Conversely, each administrative agent and deployment manager can be registered with multiple job managers.
Java Management Extensions (JMX)

Deep-dive: Extensive knowledge of JMX is not required to administer WebSphere Application Server. However, familiarity with basic concepts, such as MBeans, can be useful when you are writing scripts for wsadmin.

The system management functionality of WebSphere Application Server is based on the use of Java Management Extensions (JMX). JMX is a framework that provides a standard way of exposing Java resources, for example, application servers, to a system management infrastructure. The JMX framework allows a provider to implement functions, such as listing the configuration settings, and allows users to edit the settings. It also includes a notification layer that can be used by management applications to monitor events, such as the start-up of an application server. The use of JMX opens the door to third-party management tool providers. Users of WebSphere are no longer restricted to IBM-supplied management tools.

JMX is a Java specification (JSR-003) that is part of Java 2 Platform, Standard Edition (J2SE™) 1.5. A separate specification defines the J2EE management API (JSR-77) for managing a J2EE conforming application server. WebSphere Application Server provides managed objects (MOs) as defined in the JSR-77 specification and hence is manageable from third-party management products that deliver J2EE management capabilities.

IBM WebSphere Application Server V6, V6.1, and V7 implement JMX 1.2, while Version 5.x implements JMX 1.1. Due to the evolution of the JMX specification, the serialization format for JMX objects differs between the two specifications, such as javax.management.ObjectName. The WebSphere Application Server JMX runtime has been enhanced to be aware of the version of the client with which it is communicating. It makes appropriate transformations on these incompatible serialized formats to allow the different version runtimes to communicate with each other. This makes it possible for a V5.x administrative client to call a V7 deployment manager, node, or server. Similarly, a V7 administrative client can call a V5.x node or server.

For more information about JMX architecture, see this Web site:
JMX MBeans

Resources are managed by JMX MBeans. These are not EJBs, but simple Java beans that conform to certain design patterns outlined in the JMX specification.

Providers that want to instrument their systems with JMX need to provide a series of MBeans. Each MBean is meant to wrap, or represent, a certain runtime resource. For example, in order to expose an application server as a manageable resource, WebSphere needs to provide an application server MBean.

External applications can interact with the MBeans through the use of JMX connectors and protocol adapters. Connectors are used to connect an agent with a remote JMX-enabled management application. This form of communication involves a connector in the JMX agent and a connector client in the management application.

Each JMX-enabled JVM™ contains an MBean server that registers all the MBeans in the system. It is the MBean server that provides access to all of its registered MBeans. There is only one MBean server per JVM.

WebSphere Application Server provides a number of MBeans, each of which can have different functions and operations available. For example:

- An application server MBean might expose operations such as start and stop.
- An application MBean might expose operations such as install and uninstall.

JMX usage scenarios

You might encounter several of the more common JMX usage scenarios:

- Internal product usage:
  All WebSphere Application Server administration clients use JMX:
  - WebSphere administrative console
  - wsadmin scripting client
  - Admin client Java API

- External programmatic administration:
  In general, most external users will not be exposed to the use of JMX. Instead, they will access administration functions through the standard WebSphere Application Server administration clients.

  However, external users will need to access JMX in the following scenarios:
  - External programs written to control the WebSphere Application Server runtime and its resources by programmatically accessing the JMX API.
Third-party applications that include custom JMX MBeans as part of their deployed code, allowing the applications' components and resources to be managed through the JMX API.

Centralized Installation Manager (CIM)

CIM provides the capabilities to install, uninstall, and update the WebSphere Application Server environment from a deployment manager. After a deployment manager has been created, CIM can be used to install the WebSphere Application Server binaries on remote servers on the network.

There is no need for an agent to be running on the remote server in order to transfer the binary files. Rather, the CIM uses a native protocol to transfer the binaries over to the remote machine. After the binaries are transferred, custom profiles can be created on the remote server and federated into the deployment manager's cell.

CIM can also be used to download the update installer, interim fixes, and fix packs for WebSphere Application Server V6.1 and greater. When the fix packs have been downloaded using the CIM, the fix packs can be transferred to all remote servers and applied using the update installer. CIM can also be used to uninstall binaries from a remote server.

IBM Support Assistant (ISA) V4

IBM Support Assistant V4.0 (ISA) is a free tool provided by IBM to troubleshoot a WebSphere Application Server environment. ISA is composed of three parts:

- IBM Support Assistant Workbench
- IBM Support Assistant Agent Manager
- IBM Support Assistant Agent

The IBM Support Assistant Workbench is an Eclipse-based client application that provides features, such as:

- Search capabilities
- Product information
- Media viewer
- Data collection
- Guided troubleshooter
The majority of the workbench features are only available on the local machine where the workbench is installed, which means, all data that is collected must be manually transferred to the system where the workbench is installed. To extend the capabilities built into the workbench to import and export data to remote systems, the ISA manager and agent have to be installed.

The IBM Support Assistant Agent Manager needs to be installed only once within your network. After being installed, the workbench can authenticate itself against the manager, and collect data from remote servers through the use of the IBM Support Assistant Agent.

The IBM Support Assistant Agent is installed on all systems that you are interested in troubleshooting remotely. After installation, the agent must be registered with the ISA agent manager. The agent manager then exposes the agent to the workbench, which is now capable of transferring files to the agent, remotely collecting data from the agent, and also gathering inventory reports.

Figure 13 on page 36 shows IBM Support Assistant.
For installation instructions and more details about the IBM Support Assistant, check the following Web site:

http://www.ibm.com/software/support/isa/

If you are new to using IBM SupPort Assistant, we recommend that you visit the IBM Education assistant tools regarding IBM SupporT Assistant for help in getting started:

http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/topic/com.ibm.iea.selfassist/selfassist/ISAv41_Task.html
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright International Business Machines Corporation 2009. All rights reserved.
This document REDP-4569-00 was created or updated on October 13, 2009.

Send us your comments in one of the following ways:

- Use the online Contact us review Redbooks form found at: ibm.com/redbooks
- Send your comments in an email to: redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD  Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM®
Redbooks (logo) ®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

EJB, J2EE, J2SE, Java, JMX, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.