BEA WebLogic Server®

Deploying Applications to WebLogic Server

Version 9.0 BETA
Revised: December 15, 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

8. weblogic.Deployer Command-Line Reference

Introduction and Roadmap

The following sections describe the audience for and organization of this document:

- "Document Scope and Audience" on page 1-2
- "Guide to this Document" on page 1-2
- "Related Documentation" on page 1-3
- "New and Changed Deployment Features in This Release" on page 1-3

Document Scope and Audience

This document is intended for Administrators who want to deploy Java 2 Platform, Enterprise Edition (J2EE) applications or application modules to WebLogic Server. This document assumes that you are working in a production environment, which is generally characterized by multiple WebLogic Server instances or clusters running on multiple machines. It also assumes that you have one or more application module archive files that have been tested and are ready to deploy on a production server.

If you are an engineer, you may need to perform activities such as:

- Deploying an application in a development environment
- Packaging an application for delivery to an Administrator or Deployer
- Exporting the configuration of an application for deployment to a testing, staging, or production environment

See *Developing WebLogic Server Applications* for information about performing these development-related tasks.

Guide to this Document

- This chapter, Introduction and Roadmap, describes the organization of this document and highlights new deployment features introduced in BEA WebLogic Server 9.0.
- Understanding WebLogic Server Deployment provides an overview of deployment features used in WebLogic Server.
- Preparing Applications and Modules for Deployment explains how to prepare application and module files for deployment to WebLogic Server.
- Configuring Applications for Production Deployment how to configure an application for deployment to a specific WebLogic Server environment.
- Deploying Applications and Modules describes basic and advanced techniques for deploying applications to Weblogic Server.
- Updating Applications in a Production Environment explains how to safely update, redeploy, and reconfigure applications that you have deployed to a production environment.
- Managing Deployed Applications describes common tasks that an Administrator performs when managing deployed applications and modules.

- weblogic.Deployer Command-Line Reference provides a complete reference for the `weblogic.Deployer` tool syntax.

Related Documentation

For additional information about deploying applications and modules to WebLogic Server, see these documents:

- *Developing WebLogic Server Applications* describes how to deploy applications during development using the `wldeploy` Ant task, and provides information about the WebLogic Server deployment descriptor for Enterprise Applications.
- The WebLogic Server J2EE programming guides describe the J2EE and WebLogic Server deployment descriptors used with each J2EE application and module:
  - *Developing Web Applications, Servlets, and JSPs for WebLogic Server*
  - *Programming WebLogic Enterprise JavaBeans (EJB)*
  - *Programming WebLogic Server Resource Adaptors*
  - *Programming WebLogic Web Services for WebLogic Server*
- *Programming WebLogic JDBC* describes the XML deployment descriptors for JDBC application modules.
- *Programming WebLogic JMS* describes the XML deployment descriptors for JMS application modules.

New and Changed Deployment Features in This Release

The following sections describe new deployment features introduced in WebLogic Server 9.0.

J2EE 1.4 Deployment Implementation (JSR-88)

In J2EE 1.4, the JSR-88 specification defines a standard API for application configuration and deployment to a target application server environment. WebLogic Server 9.0 supports JSR-88 as described in "Deployment Standards" on page 2-3. See the J2EE v1.4 Documentation for more information about the JSR-88 deployment specification.

Exporting Applications for Deployment to Multiple Environments

The basic JSR-88 configuration process provides a simple way for standardized deployment tools to deploy J2EE applications on multiple application server products. However, it does not help in the process of migrating an application's configuration from one environment to another within an organization. The WebLogic Server 9.0 deployment API extends JSR-88 to provide support for exporting an application's configuration for deployment to multiple WebLogic Server environments, such as testing, staging, and production domains. See "WebLogic Server Deployment Features" on page 2-6.

New Application Directory Structure for Deployment

WebLogic Server introduces a new application directory structure for storing deployment plans and generated WebLogic Server deployment descriptor files along with the J2EE application archive or exploded archive directory. The directory structure separates generated configuration files from the core application files, so that configuration files can be easily changed or replaced without disturbing the application itself. See "Creating an Application Installation Directory" on page 3-9.

Production Redeployment and Application Versioning

WebLogic Server enables you to redeploy a new version of a production application without affecting existing clients to the application, and without interrupting the availability of the application to new client requests. See "Updating Applications in a Production Environment" on page 6-1.

Version Designation

To support the production redeployment strategy, WebLogic Server now recognizes a unique version string entry in an Enterprise Application's `MANIFEST.MF` file. When a redeployment operation is requested, WebLogic Server checks the version string to determine if a new version of the application should be deployed.

If an application does not specify a version string in the manifest file, you can specify one during deployment or redeployment using the `-appversion` option to `weblogic.Deployer`.

Production redeployment is also performed automatically if an application supports production redeployment and its deployment configuration is updated with changes to resource bindings.

This occurs even if no version string is specified in the application's manifest file. See "Updating Applications in a Production Environment" on page 6-1.

Changes to Application Naming Attributes

The `Name` attribute, retrieved from `AppDeploymentMBean`, now returns a unique application identifier consisting of the deployed application name and the application version.

Applications that require only the deployed application name must use the new `ApplicationName` attribute instead of the `Name` attribute.

Distributing Applications to Administration Mode

Distributing an application now copies deployment files to target servers and validates the deployment, and places the application in Administration mode. Administration mode restricts access to the application to a configured Administration channel. You can distribute a new application into a production environment (or distribute a new version of an application) and perform final testing without opening the application to external client connections. See "Distributing Applications to a Production Environment" on page 5-17 and "Distributing a New Version of a Production Application" on page 6-11.

JDBC and JMS Application Modules

JMS and JDBC configurations in WebLogic Server 9.0 are now stored in XML documents that conform to the appropriate WebLogic Server schema for the resource: `weblogic-jmsmd.xsd` or `weblogic-jdbc.xsd`. You create and manage JMS and JDBC resources either as system modules, similar to the way they were managed prior to version 9.0, or as application modules, similar to standard J2EE modules.

A JMS or JDBC application module can be deployed as a stand-alone resource, in which case the resource is available to the servers or cluster targeted during the deployment process, or as part of an Enterprise application. An application module deployed as part of an Enterprise Application is available only to the enclosing application (an *application-scoped resource*). Using application-scoped resources ensures that an application always has access to required resources, and simplifies the process of deploying the application into new environments.

In contrast to system modules, application modules are owned by the developer who created and packaged the module, rather than the Administrator who deploys the module. This means that the Administrator has more limited control over JDBC and JMS application modules. When deploying an application module, an Administrator can change resource properties that were specified in the module, but cannot add or delete resources. System modules are created by the Administrator via the WebLogic Administration Console, and can be changed or deleted as necessary by the Administrator.

For more information, see:

- Configuring and Managing WebLogic JMS
- Configuring and Managing WebLogic JDBC
- Deploying Applications in *Deploying Applications to WebLogic Server*

New WebLogic Server Deployment API

WebLogic Server includes a new deployment API to implement and extend the JSR-88 deployment specification. All WebLogic Server deployment tools, such as the Administration Console, `weblogic.Deployer` tool, and `wldeploy` Ant task, use the deployment API to configure, deploy, and redeploy applications in a domain. You can use the deployment API to build your own WebLogic Server deployment tools, or to integrate WebLogic Server configuration and deployment operations with an existing JSR-88-compliant tool.

The new WebLogic Server deployment API is provided in the following packages:

- `weblogic.deploy.api.model` provides the WebLogic Server implementation of, and extensions to the `javax.enterprise.deploy.model` package. This package contains the interfaces used to represent the J2EE configuration of a deployable object (an Enterprise Application or standalone module).
- `weblogic.deploy.api.spi` provides the implementation classes required to configure applications for deployment to WebLogic Server targets. This package enables a deployment tool to represent the WebLogic Server-specific deployment configuration for an Enterprise Application or standalone module.
- `weblogic.deploy.api.shared` provides classes that represent the WebLogic Server-specific deployment commands, module types, and target types. These objects are shared between the model and SPI packages.
- `weblogic.deploy.api.tools` provides WebLogic Server helper classes that you can use to easily configure applications and perform deployment operations.

See the WebLogic Server 9.0 API Reference.

DeployerRuntimeMBean Deprecated

The `weblogic.management.runtime.DeployerRuntimeMBean` API is deprecated in this release. Deployment tools should use the new `weblogic.deploy.api` packages for all application configuration and deployment tasks. See the WebLogic Server 9.0 API Reference.

Alternate Deployment Descriptors Deprecated

The use of alternate deployment descriptors for deploying applications modules is deprecated in this release. Use JSR-88-style deployment configuration and deployment plans, instead of alternate deployment descriptors, to maintain custom configurations outside of a packaged application. See "Configuring Applications for Production Deployment" on page 4-1.

weblogic.Deployer Syntax for -redeploy *module-uri* Deprecated

This release deprecates the use of the `weblogic.Deployer -redeploy` *`module-uri`* syntax for redeploying a single module in a deployment. Instead, use production redeployment or use the `-redeploy -targets` *`module@target`* syntax, as described in "Using Partial Redeployment for J2EE Module Updates" on page 6-17.

Formal Classification for WebLogic Server Deployment Descriptor Properties

Each WebLogic Server 9.0 deployment descriptor property is now formally classified into one of the following four categories: non-configurable, dependency, declaration, or configurable. These categories are used during the configuration export process to select properties to expose as variables in the deployment plan. See Exporting an Application for Deployment to New Environments in *Developing Applications for WebLogic Server*.

New Target Types for Deployable Units

WebLogic Server 9.0 introduces a new JMS Server deployment target that you can select when deploying JMS modules to a domain. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.

New Deployment Configuration Tools

WebLogic Server includes these new deployment tools:

- `weblogic.Configure`—The `Configure` utility generates WebLogic Server deployment plans and deployment descriptors as necessary to configure an application. It also exports deployment descriptor properties to deployment plan variables using the Formal Classification for WebLogic Server Deployment Descriptor Properties. See the weblogic.Configure Command Line Reference in *Developing Applications for WebLogic Server*.
- WLST—The BEA WebLogic Scripting Tool (WLST) provides commands for configuring applications and performing deployment operations, as well as other WebLogic Server administration tasks. See WLST Deployment Objects in *WebLogic Scripting Tool*.

Changed Deployment Tools

These tools have been changed to support the new configuration and deployment features:

- weblogic.Deployer
- wldeploy
- Administration Console

weblogic.Deployer

`weblogic.Deployer` has been updated to use the new deployment API to perform all deployment-related operations. `weblogic.Deployer` provides new command-line options for:

- Identifying and updating deployment plans
- Specifying the version of a deployment plan
- Setting version information for production redeployment, and setting application retirement policies
- Deploying J2EE libraries and optional packages, and specifying version string information

See the weblogic.Deployer Command-Line Reference.

wldeploy

The `wldeploy` Ant task has been updated to support the same new features introduced with `weblogic.Deployer`. See the wldeploy Ant Task Reference in *Developing Applications with WebLogic Server*.

Administration Console

The Administration Console contains new deployment assistants for installing applications, new screens for configuring applications, and new controls for performing deployment operations and redeploying production applications.

WebLogic Server 6.x Deployment Protocol Not Supported

The WebLogic Server 6.x single-phase deployment protocol, deprecated in versions 7.x and 8.x, is no longer supported in WebLogic Server 9.0. All deployments now use the two-phase deployment protocol.

CHAPTER 2

Understanding WebLogic Server Deployment

The following sections provide an overview of BEA WebLogic Server™ deployment:

Overview of the Deployment Process

The term *application deployment* refers to the process of making an application or module available for processing client requests in a WebLogic Server domain. For an Administrator, application deployment generally involves the following tasks:

1. Preparing applications and modules for deployment. WebLogic Server enables you to deploy applications either as archived JAR files, or as exploded archive directories. WebLogic Server 9.0 also introduces the concept of an application installation directory, which helps you organize deployment files and deployment configuration files for easy deployment using WebLogic Server tools. Before deploying an application, an Administrator typically prepares the deployable modules by creating an application installation directory and copying the application archive file into the appropriate subdirectory.

2. Configuring the application or module for deployment to the WebLogic Server environment. Administrators typically receive a new application (or a new version of an application) from their development team, and must deploy the application to a staging or production environment that differs from the environments used during development and testing. WebLogic Server 9.0 helps you easily configure an application for a new target domain *without* having to manually edit deployment descriptor files provided by development. Configuration changes for a specific deployment environment are stored in a new configuration artifact—a *deployment plan*—which can be stored and maintained independently of the deployment files provided by the development team.

3. Deploying the application to WebLogic Server—After preparing both the deployment and configuration files, applications are distributed to target servers in a WebLogic Server domain and made active for processing client requests.

Some BEA deployment tools, such as the Administration Console, automatically guide you through each of the above steps to help you quickly deploy applications to a WebLogic Server domain. Other tools, such as the `weblogic.Deployer` tool, provide more control over individual steps in the deployment process, but require additional conceptual knowledge about how WebLogic Server performs each step. This document focuses mainly on using the `weblogic.Deployer` tool, which provides access to many advanced deployment features. However, the Administration Console is also documented where it provides unique functionality. For example, only the Administration Console provides interactive deployment configuration for applications and modules.

After an application has been deployed to one or more WebLogic Server instances, an Administrator typically performs several common maintenance tasks, such as:

- Upgrading a deployed application to a newer version
- Reconfiguring applications to make better use of system resources
- Taking an application off-line for server or application maintenance
- Changing the order of deployment at server startup time
- Tracking and managing long-running deployment operations

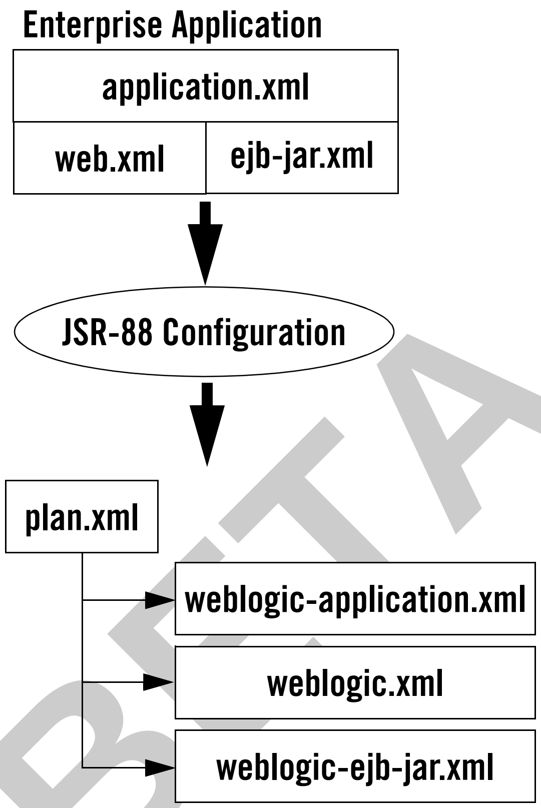
The above topics are described in "Updating Applications in a Production Environment" on page 6-1 and in "Managing Deployed Applications" on page 7-1.

Deployment Standards

WebLogic Server 9.0 implements the J2EE 1.4 specification. J2EE 1.4 includes a deployment specification, JSR-88, that describes a standard API used by deployment tools and application server providers to configure and deploy applications to an application server.

WebLogic Server 9.0 provides implementations for both the JSR-88 Service Provider Interface (SPI) plug-in and model plug-in to comply with the J2EE 1.4 deployment specification. A basic JSR-88 deployment tool can use the WebLogic Server plug-ins (without using WebLogic Server extensions to the API) to configure, deploy, and redeploy J2EE applications and modules to WebLogic Server. The WebLogic Server configuration generated by a JSR-88 configuration process is stored in a deployment plan and one or more generated WebLogic Server deployment descriptor files, as shown in the figure below. WebLogic Server deployment descriptors are generated as needed to store WebLogic Server configuration data.

Figure 2-1 Configuring Applications with JSR-88



The WebLogic Server deployment plan generated by a pure JSR-88 deployment tool identifies the WebLogic Server deployment descriptors that were generated for the application during the configuration session.

Although JSR-88 provides a simple, standardized way to configure applications and modules for use with a compliant application server, the specification does not address many of the deployment features that were available in WebLogic Server 8.1, or newly introduced in WebLogic Server 9.0. For this reason, WebLogic Server provides important extensions to the JSR-88 specification to support capabilities described in "WebLogic Server Deployment Features" on page 2-6.

Deployment Terminology

The following WebLogic Server deployment terms are used throughout this document:

application module—An XML document that configures JMS or JDBC resources. Application modules can be deployed as standalone modules in which case their resources are bound to the global JNDI tree. Application modules can also be bundled as part of an Enterprise Application and scoped within the application itself; in this case, the modules are referred to as application-scoped modules.

application installation directory—A WebLogic Server directory structure designed to help organize deployment files and generated deployment configuration artifacts for an application or module. Also referred to as an *application root directory*.

application version—A string value that identifies the version of a deployed application. Compatible applications that use version strings can use the WebLogic Server production redeployment strategy.

deployment configuration—The process of defining the deployment descriptor values required to deploy an application to a particular WebLogic Server domain. The deployment configuration for an application or module is stored in three types of XML document: J2EE deployment descriptors, WebLogic Server descriptors, and WebLogic Server deployment plans.

deployment descriptor—An XML document used to define the J2EE behavior or WebLogic Server configuration of an application or module at deployment time.

deployment plan—An XML document that defines an application's WebLogic Server deployment configuration for a specific WebLogic Server environment. A deployment plan resides outside of an application's archive file, and can apply changes to deployment properties stored in the application's existing WebLogic Server deployment descriptors. A deployment plan enables an Administrator to easily change an application's WebLogic Server configuration for a specific environment *without* modifying existing deployment descriptors. Multiple deployment plans can be used to reconfigure a single application for deployment to multiple, differing WebLogic Server environments.

distribution—The process by which WebLogic Server copies deployment source files to target servers for deployment.

production redeployment—A WebLogic Server redeployment strategy that deploys a new version of a production application alongside an older version, while automatically managing HTTP connections to ensure uninterrupted client access.

staging mode—The method WebLogic Server uses to make deployment files available to target servers in a domain. Staging modes determine whether or not files are distributed (copied) to target servers before deployment.

Security Roles Required for Deployment

The built-in security roles for “Admin” and “Deployer” users allow you to perform deployment tasks using the WebLogic Server Administration Console.

WebLogic Server Deployment Features

BEA Weblogic Server supports the following advanced deployment features to help you reliably deploy and manage applications in a production environment.

Rich Deployment Configuration

Whereas the JSR-88 deployment specification enables you to generate vendor-specific descriptor values necessary for deploying an application, WebLogic Server extensions to JSR-88 allow you to configure many additional deployment properties, including:

- The names of external resources required for the application to operate
- The declared names of services provided in a deployed application, which other applications may reference for their own use
- Tuning properties that control the performance and behavior of the application on WebLogic Server

Easy Deployment to Multiple Environments

WebLogic Server extensions to the JSR-88 deployment API make it easy to reconfigure an application as needed for deployment to multiple, different WebLogic Server environments, without unpacking or otherwise modifying the deployment source files. Migration of applications is accomplished by exporting an application’s deployment configuration to a deployment plan during development.

To export an application’s configuration, developers use the new `weblogic.Configure` tool and select WebLogic Server deployment descriptor properties that need to change when the application is deployed into another environment. For example, global resource names and tuning parameters typically differ between environments because different domains provide different types and quantities of resources to deployed applications. `weblogic.Configure` creates variable definitions in a deployment plan which act as placeholders for the selected descriptor properties. Deployment tools such as the Administration Console automatically present deployment plan variables to a deployer for customizing before deployment.

Variable definitions allow an Administrator or other deployer to easily change the values of those descriptor properties as needed for different server domains, without changing the actual deployment descriptors files in the application itself. By setting empty (null) variable definitions for required descriptor properties, a developer can require valid entries to be filled in before an application can be deployed.

See "Configuring Applications for Production Deployment" on page 4-1 for information about configuring applications before deployment with the Administration Console. See Exporting an Application for Deployment to New Environments in *Developing Applications for WebLogic Server* for more information about using the `weblogic.Configure` tool.

Administration Mode for Isolating Production Applications

Distributing an application copies deployment files to target servers, validates the deployment configuration, and places the application into Administration mode. Administration mode restricts access to an application to a configured Administration channel. You can distribute an application into a production environment (or distribute a new version of an application) without opening the application to external client connections.

While in Administration mode, you can connect to an application only via a configured Administration channel. This allows you to perform final ("sanity") checking of the distributed application directly in the production environment without disrupting clients. After performing final testing, you can either undeploy the application to make further changes, or start the application to make it generally available to clients.

See "Distributing an Application" on page 5-17.

Deployable JDBC and JMS Application Modules

As described in "JDBC and JMS Application Modules" on page 1-5, JDBC and JMS resources can now be stored as application modules, which can be deployed standalone to multiple servers or clusters or included within an Enterprise Application as application-scoped resources. Standalone JDBC and JMS application modules makes it easy to replicate resources in multiple WebLogic Server domains. Application-scoped resource modules make it possible to include all of an Application's required resources within the application module itself, for maximum portability to multiple environments. See *Developing Applications with WebLogic Server* for more information about using application-scoped resources. See *"Deploying JDBC and JMS*

Application Modules" on page 5-8 to deploy standalone or application-scoped resources to WebLogic Server.

Module-Level Deployment and Redeployment for Enterprise Applications

WebLogic Server enables you to target individual modules of an Enterprise Application to different server targets, or to deploy only a subset of available modules in an Enterprise Application. This provides flexible packaging options, allowing you to bundle a group of related modules together in an Enterprise Application, but deploy only selected modules to individual servers in a domain.

Safe Redeployment for Production Applications

WebLogic Server enables you to safely update and redeploy a new version production application without affecting current HTTP clients to the application. Production redeployment helps you roll out bug fixes or new functionality without application downtime, and without creating redundant servers in order to roll out the changes. See "Updating Applications in a Production Environment" on page 6-1.

Overview of Deployment Tools

Weblogic Server provides the following tools to help you configure and deploy applications.

weblogic.Deployer

`weblogic.Deployer` provides a command-line based interface for performing both basic and advanced deployment tasks. Use `weblogic.Deployer` when you want command-line access to WebLogic Server deployment functionality, or when you need to perform a deployment task that is not supported using the Administration Console.

Administration Console

The Administration Console provides a series of web-based Deployment Assistants that guide you through the deployment process. The Administration Console also provides controls for changing and monitoring the deployment status, and changing selected deployment descriptor values while the deployment unit is up and running.

Use the Administration Console when you need to perform basic deployment functions interactively and you have access to a supported browser.

WLST

The WebLogic Scripting Tool (WLST) is a new command-line interface that you can use to automate domain configuration tasks, including application deployment configuration and deployment operations. See *WebLogic Scripting Tool* for more information.

Deployment Tools for Developers

WebLogic Server provides several tools for deploying applications and standalone modules:

- `wldeploy` is an Ant task version of the `weblogic.Deployer` utility. You can automate deployment tasks by placing `wldeploy` commands in an Ant `build.xml` file and running Ant to execute the commands.
- weblogic.Configure is a command-line tools that enables developers to export an application's configuration for deployment to multiple WebLogic Server environments.
- The deployment API allows you to perform deployment tasks programmatically using Java classes.
- The `applications` domain directory allows you to deploy an application quickly for evaluation or testing in a development environment.

CHAPTER 3

Preparing Applications and Modules for Deployment

The following sections provide a basic overview of key BEA WebLogic Server™ deployment topics:

Overview of Preparing Applications and Modules

WebLogic Server supports deployment of standard J2EE modules and applications, as well as JDBC and JMS resource modules, as described in "Supported Deployment Units" on page 3-2. When preparing supported applications and modules for deployment, an Administrator has several options with regard to how the deployment files are arranged:

- **Archive file or Exploded Archive Directory.** Weblogic Server enables you to deploy the application source files either from a Java archive, or from an exploded archive directory. Using an exploded archive directory helps you easily update portions of the application after deployment, but is not recommended for production environments. If you choose to use an exploded archive directory, you may be required to manually unpack a previously-archived deployment. See "Deployment Archive Files Versus Exploded Archive Directories" on page 3-6 and "Exploded Archive Directories" on page 3-7.

- **Application Installation Directory or other deployment directory.** WebLogic Server 9.0 introduces a new application installation directory structure that helps you organize multiple versions of an application's source and configuration files in a standard way. Regardless of whether you deploy from an archive file or exploded archive directory, BEA recommends that you copy deployment files into the appropriate subdirectories of an application installation directory. Doing so ensures that the Administration Console can easily find deployment configuration files, such as WebLogic Server deployment descriptors and deployment plans, and can easily distinguish between multiple versions of the same application.

  An application installation directory is not required; you can also deploy archive files and exploded archive directories from any directory that is accessible to the Administration Console.

Supported Deployment Units

A *deployment unit* refers to a J2EE application (an Enterprise Application or Web Application) or a standalone J2EE module (such as an EJB or Resource Adapter) that has been organized according to the J2EE specification and can be deployed to WebLogic Server.

For each type of deployment unit, the J2EE specification defines both the required files and their location in the directory structure of the application or module. Deployment units may include Java classes for EJBs and servlets, resource adapters, Web pages and supporting files, XML-formatted deployment descriptors, and even other modules.

J2EE does not specify *how* a deployment unit is deployed on the target server—only how standard applications and modules are organized. WebLogic Server supports the following types of deployment units:

- "Enterprise Application" on page 3-3
- "Web Application" on page 3-3
- "Enterprise JavaBean" on page 3-3
- "Resource Adapter" on page 3-4
- "Web Service" on page 3-4
- "J2EE Library" on page 3-4

- "Optional Package" on page 3-5
- "JDBC and JMS Modules" on page 3-5
- "Client Application Archive" on page 3-6

Enterprise Application

An Enterprise Application consists of one or more of the following J2EE applications or modules:

- Web Applications
- Enterprise Java Beans (EJB) modules
- Resource Adapter modules

An Enterprise Application is packaged as a JAR file with an `.ear` extension, but is generally deployed as an exploded EAR directory. An exploded EAR directory contains all of the JAR, WAR, and RAR modules (also in exploded format) for an application as well as the XML descriptor files for the Enterprise Application and its bundled applications and modules. See *Developing WebLogic Server Applications* for more information.

Web Application

A Web Application always includes the following files:

- A servlet or JSP page, along with any helper classes.
- A `web.xml` deployment descriptor, a J2EE standard XML document that configures the contents of a WAR file.

Web Applications may also contain JSP tag libraries, static .html and image files, supporting classes and `.jar` files, and a `weblogic.xml` deployment descriptor, which configures WebLogic Server-specific elements for Web Applications. See *Developing Web Applications for WebLogic Server* for more information.

Enterprise JavaBean

Enterprise JavaBeans (EJBs) are reusable Java components that implement business logic and enable you to develop component-based distributed business applications. EJB modules are packaged as archive files having a `.jar` extension, but are generally deployed as exploded archive directories. The archive file or exploded archive directory for an EJB contains the compiled EJB classes, optional generated classes, and XML deployment descriptors for the EJB.

See *Programming WebLogic Server Enterprise JavaBeans* for more information on the different types of EJBs.

Resource Adapter

A Resource Adapter (also referred to as a connector) adds Enterprise Information System (EIS) integration to the J2EE platform. Connectors provide a system-level software driver that WebLogic Server can use to connect to an EIS. Connectors contain both the Java classes, and if necessary, the native components required to interact with the EIS. See *Programming WebLogic Server J2EE Connectors* for more information.

Note: Resource Adapters cannot be deployed as exploded archive directories.

Web Service

A Web Service is a set of functions packaged into a single entity that is available to other systems on a network, and can be shared by and used as a component of distributed Web-based applications. Web Services commonly interface with existing back-end applications, such as customer relationship management systems, order-processing systems, and so on.

A Web Service module may include either Java classes or EJBs that implement the Web Service. Web Services are packaged either as Web Application archives (WARs) or EJB modules (JARs) depending on the implementation. See *Programming WebLogic Web Services* for more information.

J2EE Library

A J2EE library is a standalone J2EE module, or multiple J2EE modules packaged in an Enterprise Application (EAR), that is registered with the J2EE application container as a shared library at deployment time. After a J2EE library has been registered, you can deploy Enterprise Applications that reference the library in their `weblogic-application.xml` deployment descriptors. Each referencing application receives a copy of the shared J2EE library module(s) on deployment, and can use those modules as if they were packaged as part of the application itself. J2EE library support provides an easy way to share one or more J2EE modules among multiple Enterprise Applications without physically adding the shared modules to each dependent application.

The deployment files of a shared library resemble either a standard Enterprise Application or J2EE module, as discussed in this section. Shared libraries differ from standard EARs and modules only by the contents of their `MANIFEST.MF` files. Creating Shared J2EE Libraries and

Optional Packages in *Developing Applications with WebLogic Server* describes how to assemble and configure J2EE libraries, and how to configure Enterprise Applications that utilize J2EE libraries.

"Deploying JDBC and JMS Application Modules" on page 5-8 provides instructions for how to deploy J2EE libraries and Enterprise Applications that reference J2EE libraries.

Optional Package

Optional packages provide similar functionality to J2EE libraries, allowing you to easily share a single JAR file among multiple applications. However, optional packages function at the level of an individual J2EE module (standalone or within an Enterprise Application), rather than at the Enterprise Application level. For example, third-party Web Application Framework classes needed by multiple Web Applications can be packaged and deployed in a single JAR file, and referenced by multiple Web Application modules in the domain.

Optional packages are delivered as basic JAR files that have no deployment descriptors. You simply designate the JAR as an optional package at deployment time, and WebLogic Server registers the file with the target servers you select. After the optional package has been registered, you can then deploy J2EE modules and applications that reference the optional package in their `MANIFEST.MF` files. Creating Shared J2EE Libraries and Optional Packages in *Developing Applications with WebLogic Server* describes how to assemble and configure optional packages, and how to configure J2EE modules that utilize optional packages.

"Deploying Shared J2EE Libraries and Dependent Applications" on page 5-18 provides instructions for how to deploy optional packages and J2EE modules that reference optional packages.

JDBC and JMS Modules

JMS and JDBC configurations in WebLogic Server 9.0 are now stored in XML documents that conform to the appropriate WebLogic Server schema for the resource: `weblogic-jmsmd.xsd` or `weblogic-jdbc.xsd`. You create and manage JMS and JDBC resources either as system modules, similar to the way they were managed prior to version 9.0, or as application modules, similar to standard J2EE modules.

A JMS or JDBC application module can be deployed as a stand-alone resource, in which case the resource is available in the domain targeted during deployment, or as part of an Enterprise application. An application module deployed as part of an Enterprise Application is available only to the enclosing application (an *application-scoped resource*). Using application-scoped resources ensures that an application always has access to required resources, and simplifies the process of deploying the application into new environments.

In contrast to system modules, application modules are owned by the developer who created and packaged the module, rather than the Administrator who deploys the module. This means that the Administrator has more limited control over JDBC and JMS application modules. When deploying an application module, an Administrator can change resource properties that were specified in the module, but cannot add or delete resources.

System modules are created by the Administrator via the WebLogic Administration Console, and can be changed or deleted as necessary by the Administrator. Similarly, standalone application modules created by the Administrator can be used to recreate global resources in multiple WebLogic Server environments simply by deploying the modules into new domains.

For more information about creating and using JDBC and JMS modules, see:

- New and Changed Features In This Release in *Configuring and Managing WebLogic JMS*
- Configuring and Managing WebLogic JDBC

See "Deploying Applications and Modules" on page 5-1 for information about deploying JMS and JDBC standalone modules.

Client Application Archive

The J2EE specification provides the ability to include a client application archive file within an Enterprise Application. A J2EE client application module contains the Java classes that execute in the client JVM (Java Virtual Machine) and deployment descriptors that describe EJBs (Enterprise JavaBeans) and other WebLogic Server resources used by the client. This enables both the server-side and client-side components to be distributed as a single unit. Client modules in an EAR are defined using the J2EE standard `application-client.xml` deployment descriptor and WebLogic Server `weblogic-appclient.xml` descriptor.

Deployment Archive Files Versus Exploded Archive Directories

WebLogic Server supports deployments that are packaged either as archive files using the `jar` utility or Ant's `jar` tool, or as exploded archive directories. An archive file is a single file that contains all of an application's or module's classes, static files, directories, and deployment descriptor files. In most production environments, the applications you receive for deployment are stored as archive files. BEA also recommends deploying applications as archive files in a production environment, to minimize the possibility of changing, deleting, or moving individual deployment files while the application is running.

Deployment units that are packaged using the `jar` utility have a specific file extension depending on the type:

- EJBs are and client archives are packaged as `.jar` files.
- Web Applications are packaged as `.war` files.
- Resource Adapters are packaged as `.rar` files.
- Enterprise Applications are packaged as `.ear` files, and can contain other J2EE modules such as EJBs, Web Applications, and Resource Adapters.
- Web Services can be packaged either as `.war` files or as `.jar` files, depending on whether they are implemented using Java classes or EJBs.
- J2EE libraries are packaged either as an Enterprise Application (`.ear` file) or as a standard J2EE module.
- Client applications and optional packages are packaged as `.jar` files.

In addition to an archive file, you may also receive a deployment plan, which is a separate file that configures the application for a specific environment. "Configuring Applications for Production Deployment" on page 4-1 describes deployment plans in more detail.

Deploying from archive files is recommended in most production environments, as it ensures that all of an application's supporting files are maintained within a single file, and cannot become accidentally renamed or deleted.

Exploded Archive Directories

An exploded archive directory contains the same files and directories as a JAR archive. However, the files and directories reside directly in your file system and are not packaged into a single archive file with the `jar` utility.

Exploded archive directories are generally not used in production environments, where applications changes must be carefully managed. However, in non-production environments you may choose to deploy from an exploded archive directory under the following circumstances:

- You want to perform partial updates of an Enterprise Application after deployment. Deploying Enterprise Applications as an exploded archive directory makes it easier to update individual modules of the application without having to re-create the archive file.
- You are deploying a Web Application or Enterprise Application that contains static files that you will periodically update. In this case, it is more convenient to deploy the

application as an exploded directory, because you can update and refresh the static files without re-creating the archive.

- You are deploying a Web Application that performs direct file system I/O through the application context (for example, a Web Application that tries to dynamically edit or update parts of the Web Application itself). In this case, the modules that perform the I/O operations should have a physical filesystem directory in which to work; you cannot obtain a file when the application is deployed as an archive, as per the specification.

Creating an Exploded Archive Directory from an Archive File

If you have an archive file that you want to deploy as an exploded archive directory, use the `jar` utility to unpack the archive file in a dedicated directory. For example:

```
mkdir /myapp

cd /myapp

jar xvf /dist/myapp.ear
```

If you are unpacking an archive file that contains other module archive files (for example, an Enterprise Application or Web Service that includes JAR or WAR files) and you want to perform partial updates of those modules, you must expand the embedded archive files as well. Make sure that you unpack each module into a subdirectory having the same name as the archive file. For example, unpack a module named `myejb.jar` into a `/myejb.jar` subdirectory of the exploded Enterprise Application directory.

Note: If you want to use different subdirectory names for the archived modules in an exploded EAR file, you must modify any references to those modules in the application itself. For example, you must update the URI values specified in `application.xml` and `CLASSPATH` entries in the `manifest.mf` file.

Choosing a Deployment Name

When you first deploy an application or standalone module to one or more WebLogic Server instances, you specify a deployment name to describe collectively the deployment files, target servers, and other configuration options you selected. You can later redeploy or stop the deployment unit on all target servers by simply using the deployment name. The deployment name saves you the trouble of re-identifying the deployment files and target servers when you want to work with the deployment unit across servers in a domain.

If you do not specify a deployment name at deployment time, the deployment tool selects a default name based on the deployment source file(s). For archive files, `weblogic.Deployer` uses the name of the archive file without the file extension. For example, the file `myear.ear` has a default deployment name of `myear`. For an exploded archive directory, `weblogic.Deployer` uses the name of the top-level directory you deploy.

For J2EE libraries and optional packages, `weblogic.Deployer` uses the name specified in the library's manifest file, unless you override the name using the `-name` option.

See "Deploying Applications and Modules" on page 5-1 for instructions for specifying a non-default deployment name when you deploy an application or module.

Understanding Deployment Version Strings

In addition to a deployment name, an application or module can also have an associated version string. The version string distinguishes the initial deployment of the application from subsequent redeployed versions. For example, you may want to later update the application to fix problems or add new features. In production systems, it is critical to maintain a version string for both the initial and subsequent deployments of an application. Doing so allows you to update and redeploy an application version without interrupting service to existing clients. See "Updating Applications in a Production Environment" on page 6-1 for more information.

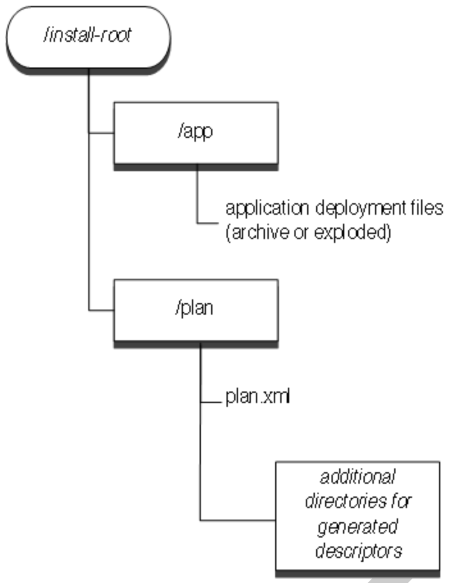
The version string is specified in the manifest file for the application, and should be provided by your development team along with the other deployment files. Specifying Application Versions in *Developing WebLogic Server Applications* describes the conventions for specifying the version string.

Creating an Application Installation Directory

The application installation directory separates generated configuration files from the core application files, so that configuration files can be easily changed or replaced without disturbing the application itself. The directory structure also helps you to organize and maintain multiple versions of the same application deployment files.

The following figure shows the directory hierarchy for storing a single version of a deployable application or module.

Figure 3-1 Application Installation Directory


/install-root
/app
application deployment files
(archive or exploded)
/plan
plan.xml
additional directories for generated descriptors

BEA recommends copying all new production deployments into an application installation directory before deploying to a WebLogic Server domain. Deploying from this directory structure helps you easily identify all of the files associated with a deployment unit—you simply deploy the installation root using the Administration Console, and the Console automatically locates associated files such as deployment plans and WebLogic Server deployment descriptors that were generated during configuration.

Steps for Creating an Application Installation Directory

To deploy an application or module from an installation directory:

1. Decide where you want to store the deployment files for applications and modules on your system. Keep the following best practices in mind:
   – Do not store deployment files alongside configuration files for a WebLogic Server domain.

   - Use source control if available to maintain deployment source files.
   - If possible, store deployment files on a directory that is accessible by the Administration Server and Managed Servers in your domain.

   The instructions that follow use the sample deployment directory, `c:\deployments\production`.

2. Create a dedicated subdirectory for the application or module you want to deploy. For example:

   ```
   mkdir c:\deployments\production\myApplication
   ```

3. Create a subdirectory beneath the application directory to designate the version of the application you are deploying. Name the subdirectory using the exact version string of the application. For example:

   ```
   mkdir c:\deployments\production\myApplication\91Beta
   ```

4. The version subdirectory will become the installation root directory from which you deploy the directory. Create the standard application installation subdirectories under the version subdirectory:

   ```
   mkdir c:\deployments\production\myApplication\91Beta\app
   ```

   ```
   mkdir c:\deployments\production\myApplication\91Beta\plan
   ```

5. Copy your application source deployment files into the `\app` subdirectory. If you are deploying from an archive file, simply copy the archive file, as in:

   ```
   cp c:\downloads\myApplication.ear
   c:\deployments\production\myApplication\91Beta\app
   ```

   If you are deploying from an exploded archive directory, copy the complete exploded archive directory into `\app`. For example:

   ```
   cp -r c:\downloads\myApplication
   c:\deployments\production\myApplication\91Beta\app
   ```

   This results in the new directory, `c:\deployments\production\myApplication\91Beta\app\myApplication`.

6. If you have one or more deployment plans for the application, copy them into the `\plan` subdirectory:

   ```
   cp c:\downloads\myApplicationPlans\plan.xml
   c:\deployments\production\myApplication\91Beta\plan
   ```

7. To deploy the application using Administration Console, select the application installation directory. By default, the Administration Console will use a plan named `plan.xml`, if one is available in the `plan` subdirectory. If multiple plans are available, the Administration Console prompts you to select the plan to use for configuring the application. See "Configuring Applications for Production Deployment" on page 4-1.

   Note: You cannot specify an application installation directory when using the `weblogic.Deployer` tool, and the tool does not use an available `plan.xml` file by default. You must specify the actual deployment file(s) and plan to use for deployment. See "Deploying Applications and Modules" on page 5-1.

Best Practices for Preparing Deployment Files

BEA recommends the following best practices when preparing applications and modules for deployment:

- For production environments, deploy applications and modules in an archive format (`.ear`, `.jar`, `.war`, and so forth). Using an archive file helps to ensure that individual application support files are not accidentally deleted, modified, or moved.
- For non-production environments, it is more convenient to deploy applications in exploded format rather than archive format. Check the scenarios described under "Exploded Archive Directories" on page 3-7 for more information.
- Regardless of whether you deploy an archive file or exploded archive directory, store the deployment files in an installation directory for the application, as described in "Creating an Application Installation Directory" on page 3-9. Using an installation directory simplifies the deployment process, because the Administration Console understand where to locate deployment and configuration files.
- Manage the entire application installation directory in a source control system, so you can easily revert to previous application versions if necessary.

Configuring Applications for Production Deployment

The following sections describe how to configure applications for deployment to a production WebLogic Server environment:

Overview of Deployment Configuration

When you receive a new application, or a new version of an application, from your development or quality assurance teams, the application has generally been configured for a development or testing environment. This means that the application may use specific resource names and performance tuning settings that match the available resources on the target servers used in the development or QA environments where the application was last deployed.

Because development and testing environments can be significantly different from the production environment in which the application is ultimately deployed, an Administrator must configure the application to use resource names and performance tuning parameters that are valid and appropriate for the production environment.

Understanding Application Deployment Descriptors

The basic deployment configuration for an application is defined in multiple XML documents, known as deployment descriptors, that are included as part of the application archive file that you receive for deployment. Deployment descriptor files fall into two separate categories:

- *J2EE deployment descriptors* define the fundamental organization and behavior of a J2EE application or module, regardless of where the application is deployed. Each J2EE application and module requires a specific J2EE deployment descriptor as defined in the J2EE 1.4 specification.
- *WebLogic Server deployment descriptors* define the resource dependencies and tuning parameters that an application uses in a specific WebLogic Server environment.

For the purposes of a production deployment, you should treat both the J2EE and WebLogic Server deployment descriptors as part of the application's source code, which is owned by your development team. Do not edit application deployment descriptors in order to configure an application for deployment to a production environment. Instead, use the Administration Console to persist configuration changes into a WebLogic Server *deployment plan*, which is described in the next section.

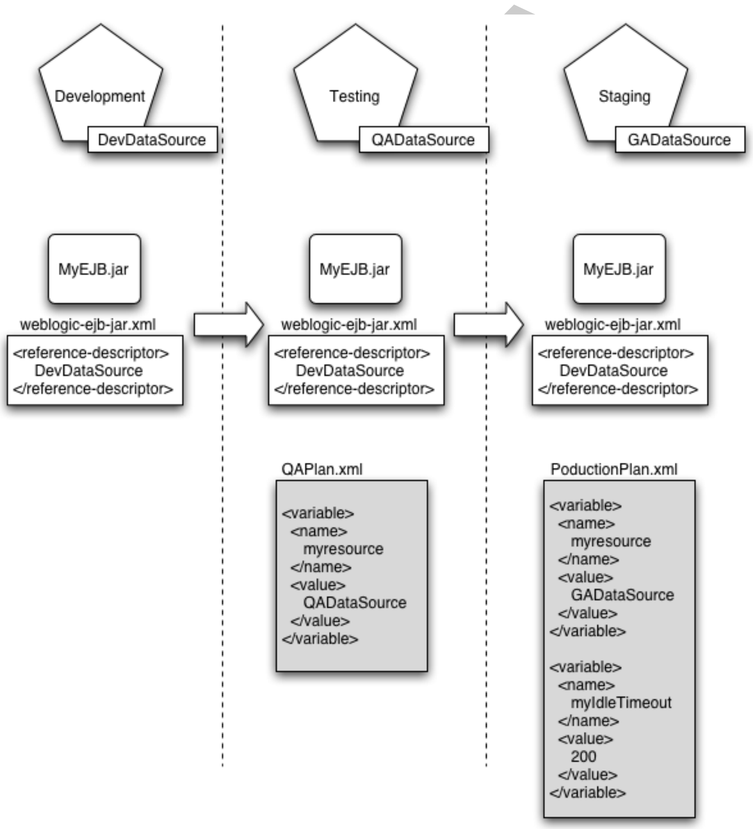
Understanding WebLogic Server Deployment Plans

A WebLogic Server deployment plan is an optional XML document that resides outside of an application archive and configures an application for deployment to a specific WebLogic Server environment. A deployment plan works by setting deployment property values that would normally be defined in an application's WebLogic Server deployment descriptors, or by overriding property values that are already defined in a WebLogic Server deployment descriptor.

Deployment plans are created and owned by the Administrator or deployer for a particular environment, and are stored *outside* of an application archive or exploded archive directory. As a best practice, BEA recommends storing all deployment plans for a single application in the `plan` subdirectory of the application's root directory (See "Creating an Application Installation Directory" on page 3-9).

Deployment plans help you easily modify an application's WebLogic Server configuration for deployment into to multiple, differing WebLogic Server environments *without* modifying the deployment descriptor files included in the application archive. To deploy the application to a new environment, an Administrator simply creates or uses a new deployment plan as necessary.

Figure 4-1 Configuring an Application for Multiple Deployment Environments


Development
DevDataSource
Testing
QADataSource
Staging
GADataSource
MyEJB.jar
MyEJB.jar
MyEJB.jar
weblogic-ejb-jar.xml
weblogic-ejb-jar.xml
weblogic-ejb-jar.xml
<reference-descriptor> DevDataSource </reference-descriptor>
<reference-descriptor> DevDataSource </reference-descriptor>
<reference-descriptor> DevDataSource </reference-descriptor>
QAPlan.xml
PoductionPlan.xml

```
<variable>
  <name>
    myresource
  </name>
  <value>
    QADataSource
  </value>
</variable>
```

```
<variable>
  <name>
    myresource
  </name>
  <value>
    GADataSource
  </value>
</variable>

<variable>
  <name>
    myIdleTimeout
  </name>
  <value>
    200
  </value>
</variable>
```

Figure 4-1 shows how deployment plans are typically used when releasing a new version of an application. During development, a programmer creates both J2EE and WebLogic Server deployment descriptors to configure the application for repeated deployments to their development environment. At this point, the application is generally deployed from an split development directory, rather than an archive file, to facilitate easy editing and redeployment of the application. A deployment plan is not necessary during deployment, because the developer has full access to the application's deployment descriptors, and the development environment is typically a simple, single-server domain. Figure 4-1 shows that the development server uses a simple PointBase database for development, named "DevDataSource," and the `weblogic-ejb-jar.xml` descriptor identifies the resource. Application tuning parameters are not defined in the development configuration, because the developer is focused on the basic semantics and behavior of the application.

To release a new version of the application, the developer packages the application into an archive file and delivers it to an Administrator or deployer in the quality assurance team. At this point, the embedded deployment descriptors provide a configuration that is valid for the development environment used by the developer, but are not valid for the testing environment where the application must be deployed. Figure 4-1 shows that the testing environment uses a different datasource name than the one used during development. To deploy the application, the Administrator of the testing environment generates a deployment plan to override the datasource name configured in the application's embedded deployment descriptors. Figure 4-1 shows that the deployment plan configures the application to use the resource named "QADataSource."

Similarly, when the application is released into production, the Administrator of the staging or production environment creates or uses another deployment plan to configure the application. Figure 4-1 shows that the production deployment plan once again overrides the application deployment descriptor to identify a new JDBC datasource name. For this environment, the deployment plan also defines tuning parameters to make better use of the additional resources available in the production domain.

Goals for Production Deployment Configuration

For the Administrator, the primary goal of configuring an application for production deployment is to generate a new deployment plan that is valid and appropriate for the target WebLogic Server environment. Specifically, the deployment plan must resolve all external resources references for the application to refer to valid resources available in the target environment. If the Application's configuration does not define to valid external resources for the target servers, the application cannot be deployed.

A deployment plan can optionally define or override WebLogic Server tuning parameters, to make ideal use of resources in the target environment. Defining tuning parameters is not required in order to successfully deploy an application. If an application's deployment descriptors and deployment plan do not define tuning parameters, WebLogic Server uses default values.

Creating a New Deployment Plan to Configure an Application

Notes: The Administration Console in this Beta release of WebLogic Server does not support configuration of EJB modules, and does not create deployment plans that can resolve external resource references for an application. If you need to create a deployment plan for EJB module, or create a plan that resolves external resource dependencies, use the `weblogic.Configure` utility described in Exporting an Application for Deployment to New Environments in *Programming Applications with WebLogic Server*. The Administration Console in this release can generate deployment plans that add or override tuning properties for an application.

In this Beta release, deployment plans can be used only with application deployment descriptors that use the new WebLogic Server schemas. Older, DTD-based deployment descriptors are not compatible with deployment plans.

The Administration Console automatically generates (or updates) a valid XML deployment plan for an application when you interactively change deployment properties for an application that you have installed to the domain. You can use the generated deployment plan to configure the application in subsequent deployments, or you can generate new versions of the deployment plan by repeatedly editing and saving deployment properties.

Generating a deployment plan using the Administration Console involves these steps:

1. "Preparing the Deployment Files" on page 4-5
2. "Installing the Application Archive" on page 4-6
3. "Saving Configuration Changes to a Deployment Plan" on page 4-7

Each step is described in the sections that follow.

Note: These sections describe using the sample application, `jspExpressionEar`. Because the sample applications installed with this Beta release use the older, DTD-based WebLogic Server deployment descriptors, please download the updated version of this sample application from http://e-docs.bea.com/wls/docs90/jspExpressionEar.ear, as described in the steps that follow.

Preparing the Deployment Files

Follow these steps to place the deployment files into an application root directory for installation:

1. Create a new root directory for deploying the sample application. For example:

   ```
   mkdir c:\sample_root
   mkdir c:\sample_root\app
   mkdir c:\sample_root\plan
   ```

2. Download the sample application from http://e-docs.bea.com/wls/docs90/jspExpressionEar.ear. Save the file to the `c:\sample_root\app` directory you created in Step1.

   Note: This archive file contains the JSP expression example installed with WebLogic Server, but converted to use the newer, schema-based deployment descriptors.

Installing the Application Archive

The Administration Console uses an application installation assistant to help you install a new application for configuration and deployment to a WebLogic Server environment. The installation assistant copies deployment files to the Administration Server and selects target WebLogic Server instances for deploying the application. After installing an application or module, the deployment files are available in the WebLogic Server domain and can be configured, distributed, and deployed as necessary.

Follow these steps to install the sample application to the examples server domain:

1. Start the examples server by using the Windows start menu or by running the `*WL_HOME*\samples\domains\wl_server\startWebLogic.cmd` script.

2. Access the Administration Console by pointing your browser to http://localhost:7001/console.

3. Log into the Administration Console.

4. Click the Lock & Edit button.

5. Select the Deployments node in the left pane.

6. On the right pane, click the Install button to launch the Install Application Assistant.

7. Use the links on the first page of the assistant to select the `c:\sample_root` directory you created in "Preparing the Deployment Files" on page 4-5. Click the Next button to continue.

8. On the Deployment Identity page, select Finish to accept the defaults. The Administration Console returns to the Summary of Deployments page and lists the `sample_root` name as a deployment.

9. Click the Activate Changes button.

Saving Configuration Changes to a Deployment Plan

Follow these steps to edit deployment configuration properties and save the configuration to a deployment plan:

1. From the Summary of Deployments page in the Administration Console, click the plus (+) sign next to the `sample_root` entry in the table to expand the deployment name. This displays the `jspExpressionWar` module within the Enterprise Application.
2. Click the plus (+) sign next to the `sample_root` entry in the table to expand the deployment name. This displays the `jsp_expr` Web Application.
3. Select the `jsp_expr` Web Application to view an overview of the deployment.
4. Click the Configuration tab. This page displays selected configuration properties that you can edit for the Web Application. (Other module types provide a Configuration tab with editable properties specific to the module).
5. Click the Lock & Edit button.
6. Edit one or more configuration properties in the Configuration tab. For example, change the Session Invalidation Interval to 80 seconds, and the Session Timeout to 8000 seconds.
7. Click Save to save your changes. The Administration Console stores your configuration changes to a new deployment plan. Because you deployed the sample application from a root directory, the Console automatically places the new deployment plan in the `\plan` subdirectory of the root directory, `c:\sample_root\plan\Plan.xml`.
8. Click the Activate Changes button.

Understanding Deployment Plan Contents

The deployment plan generated in "Creating a New Deployment Plan to Configure an Application" on page 4-5 contains the entries shown in "Sample Deployment Plan" on page 4-7.

Listing 4-1 Sample Deployment Plan

```
<deployment-plan xmlns="http://www.bea.com/ns/weblogic/90">
  <application-name>sample_root</application-name>
    <variable-definition>
    <variable>
      <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
      <value>80</value>
    </variable>
    <variable>
      <name>SessionDescriptor_TimeoutSecs_11029744772011</name>
      <value>8000</value>
    </variable>
    </variable-definition>
  <module-override>
    <module-name>jspExpressionEar.ear</module-name>
    <module-type>ear</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-application</root-element>
      <uri>META-INF/weblogic-application.xml</uri>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>application</root-element>
      <uri>META-INF/application.xml</uri>
    </module-descriptor>
  </module-override>
  <module-override>
    <module-name>jspExpressionWar</module-name>
    <module-type>war</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-web-app</root-element>
      <uri>WEB-INF/weblogic.xml</uri>
      <variable-assignment>
        <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
        <xpath>/weblogic-web-app/session-descriptor/invalidation-interval-secs
</xpath>
      </variable-assignment>
      <variable-assignment>
        <name>SessionDescriptor_TimeoutSecs_11029744772011</name>
        <xpath>/weblogic-web-app/session-descriptor/timeout-secs</xpath>
      </variable-assignment>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>web-app</root-element>
      <uri>WEB-INF/web.xml</uri>
    </module-descriptor>
  </module-override>
  <module-override>
    <module-name>sample_root</module-name>
    <module-type>ear</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-application</root-element>
      <uri>META-INF/weblogic-application.xml</uri>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>application</root-element>
      <uri>META-INF/application.xml</uri>
    </module-descriptor>
  </module-override>
  <config-root>C:\sample_root\plan</config-root>
</deployment-plan>
```

The basic elements in the deployment plan serve the following functions:

- `deployment-plan` encapsulates all of the deployment plan's contents.
- `application-name` corresponds to the deployment name for the application or module.
- `variable-definition` defines one or more `variable` elements. Each `variable` defines the `name` of a variable used in a plan and a `value` to assign (which can be null). The sample plan shown in "Sample Deployment Plan" on page 4-7 contains variable definitions for the changes you made to the Session Invalidation Interval and Session Timeout properties.
- `module-override` elements define each module name, type, and deployment descriptor that the deployment plan overrides. A `module-descriptor` element can optionally contain a `variable-assignment` which identifies a variable name used to override a property in the descriptor, and the exact location within the descriptor where the property is overridden.

  The sample plan shown in "Sample Deployment Plan" on page 4-7 contains module override elements for the Enterprise Application, the embedded Web Application, and the enclosing root directory. The module-descriptor entry for the `weblogic.xml` descriptor file contains two `variable-assignment` elements that override the property values for the Session Invalidation Interval and Session Timeout properties you changed in "Saving Configuration Changes to a Deployment Plan" on page 4-7.

For more information about the contents of a WebLogic Server deployment plan, see the "Deployment Plan Reference and Schema" on page 9-1.

Using an Existing Deployment Plan to Configure an Application

Applications that you receive for deployment may come with varying levels of configuration information. If you have an existing deployment plan for an application, simply prepare the application as described in "Preparing the Deployment Files" on page 4-5 and place the deployment plan in the `plan` subdirectory of the application root. Then install the application using the instructions in "Installing the Application Archive" on page 4-6. The Administration Console automatically uses a deployment plan named `plan.xml` in the `plan` subdirectory of an application root directory if one is available. If multiple plans are available in the `plan` subdirectory, the Console prompts you to select the plan to use.

After you install a new application and existing deployment plan, the Administration Console validates the deployment plan configuration against the target servers and clusters that were selected during installation. If the deployment plan contains empty (null) variables, or if any values configured in the deployment plan are not valid for the target server instances, you must override the deployment plan before you can deploy the application. You can also configure tuning parameters to better suit the target environment in which you are deploying the application, as described in "Saving Configuration Changes to a Deployment Plan" on page 4-7. Changes you make to the application's configuration are saved to a new deployment plan.

Note: In this Beta release, the Administration Console ignores null variable definitions in an existing deployment plan.

If you have a valid deployment plan that fully configures an application for the environment in which you are deploying, you can use either the Administration Console or the `weblogic.Deployer` utility to identify the application and plan to use for deployment. Note that any deployment plan you use with the `weblogic.Deployer` utility must be complete and valid for your target servers; `weblogic.Configure` does not allow you to set or override individual deployment properties that are defined in the plan. To deploy a new application and existing deployment plan using `weblogic.Deployer`, see "Deploying an Application with a Deployment Plan" on page 5-3.

Additional Configuration Tasks

See the following sections for information about additional deployment configuration tasks:

- "Deploying an Application with a Deployment Plan" on page 5-3 describes how to deploy an application with a valid deployment plan using the `weblogic.Deployer` tool.
- "Updating the Deployment Configuration for an Application" on page 6-19 describes how to update the deployment configuration for a currently-deployed application.
- Exporting an Application for Deployment to New Environments in *Developing Applications with WebLogic Server* explains how developers can create portable deployment plans using the `weblogic.Configure` tool.
- weblogic.Configure Command Line Reference in *Developing Applications with WebLogic Server* provides a complete reference to the `weblogic.Configure` tool.

Best Practices for Managing Application Configuration

- Always manage multiple deployment configurations using deployment plans, rather than multiple versions of the WebLogic Server deployment descriptor files.

- Always store existing deployment plans in the `plan` subdirectory of an application root directory.
- If your organization requires standardized, repeatable deployments to several environments, use the Application with Single Deployment Plan workflow to maintain a single deployment plan in your source control system.
- If you make extensive changes to an application's deployment configuration using the Administration Console, backup or safely store the updated deployment plan for future use. BEA recommends storing the entire application root directory in a source control system, so that you can maintain configuration information for multiple environments and multiple versions of an application.

Deploying Applications and Modules

The following sections describe how to perform basic and advanced deployment tasks using the `weblogic.Deployer` utility:

Overview of Common Deployment Scenarios

The sections that follow organize common deployment tasks into several general categories, starting with the most basic tasks and progressing to more advanced tasks:

- Deploying to a Single-Server Domain—Explains the basics of deploying an application or module to a single-server WebLogic domain.
- Uploading Deployment Files from a Remote Client—Explains how to upload an application or module to the Administration Server from a remote client.
- Targeting Deployments to Servers, Clusters, and Virtual Hosts—Describes all WebLogic Server target types, and explains how to identify targets during deployment.
- Using Module-Level Targeting for Deploying an Enterprise Application—Describes how to target individual modules in an Enterprise Application to different WebLogic Server targets.
- Deploying JDBC and JMS Application Modules—Describes how to deploy both standalone and application-scoped JDBC and JMS modules in a WebLogic Server domain.
- Controlling Deployment File Copying with Staging Modes—Describes how to use non-default staging modes to control WebLogic Server file-copying behavior.
- Distributing Applications to a Production Environment—Describes how to deploy and test a new application directly in a production environment *without* making the application available for processing client requests.
- Deploying Shared J2EE Libraries and Dependent Applications—Describes how to deploy J2EE Libraries and optional packages to a Weblogic Server environment, and how to deploy and manage applications and modules that reference these shared resources.
- Auto-Deploying Applications in Development Domains—Describes a procedure for quickly deploying sample applications in a development domain.
- Best Practices for Deploying Applications—Summarizes key deployment practices.

Deploying to a Single-Server Domain

A single-server WebLogic Server domain, consisting only of an Administration Server, represents the simplest scenario in which to deploy an application or module. If you are deploying files that reside on the same machine as the domain, use the **`-deploy`** command and identify the file location, with connection arguments for the Administration Server. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -deploy c:\localfiles\myapp.ear
```

In the above command, WebLogic Server creates a default deployment name of `myapp`, as described in "Choosing a Deployment Name" on page 3-8. If you want to specify a non-default deployment name, use the `-name` option, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001
   -user weblogic -password weblogic -deploy
   -name myTestApplication c:\localfiles\myapp.ear
```

Deploying an Application with a Deployment Plan

When you use `weblogic.Deployer` to deploy an application, the deployment plan and WebLogic Server deployment descriptors must define a valid configuration for the target environment, or the deployment fails. This means you cannot use `weblogic.Deployer` with a deployment plan that defines null variables for an application's required resource bindings.

To deploy an application and deployment plan using `weblogic.Deployer`, include the `-plan` option with the **`-deploy`** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -deploy -name myTestDeployment
     -source /myDeployments/myApplication.ear
     -targets myCluster -stage
     -plan /myDeployments/myAppPlan.xml
```

If you are deploying from an application root directory and the deployment plan is located in the `/plan` subdirectory, you still need to identify both the actual deployment source files the plan to use for deployment, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -deploy -name myTestDeployment
     -source
/myDeployments/installedApps/myApplication/app/myApplication.ear
     -targets myCluster -stage
     -plan /myDeployments/installedApps/myApplication/plan/plan.xml
```

When you deploy or distribute an application with a deployment plan, the deployment plan and any generated deployment descriptors are copied to the staging directories of target servers along with the application source files.

Uploading Deployment Files from a Remote Client

In order to deploy an application or module to a domain, the deployment file(s) must be accessible to the domain's Administration Server. If the files do not reside on the Administration Server machine or are not available to the Administration Server machine via a network mounted directory, use the `-upload option` to upload the files before deploying them:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -deploy -upload c:\localfiles\myapp.ear
```

To upload an exploded archive directory, specify the directory name instead of an archive filename (for example `c:\localfiles\myappEar`).

Note: You cannot upload an exploded archive directory using the Administration Console.

When you upload files to the Administration Server machine, the archive file is automatically placed in the server's upload directory. You can configure the path of this directory using the instructions in "Changing the Default Staging Behavior for a Server" on page 5-16.

Targeting Deployments to Servers, Clusters, and Virtual Hosts

In most production environments, you typically deploy applications to one or more Managed Servers configured in a Weblogic Server domain. In some cases, the servers may be included as part of a WebLogic Server cluster, or a virtual host may be used for directing Web Application requests. The term *deployment target* refers to any server or collection of servers to which you can deploy an application or module.

Understanding Deployment Targets

Deployment targets are the servers or groups of servers on which you deploy an application or standalone module. During the deployment process, you select the list of targets from the available targets configured in your domain. You can also change the target list after you have deployed a module.

The following table describes all valid deployment targets for WebLogic Server, and lists the types of modules that you can deploy to each target.

Table 5-1 WebLogic Server 9.0 Deployment Targets

Target Type	Description	Valid Deployments
WebLogic Server Instance	A WebLogic Server instance, such as an Administration Server in a single-server domain, or a Managed Server.	J2EE Applications J2EE modules JMS or JDBC modules J2EE Libraries
Cluster	A configured cluster of multiple WebLogic Server instances	J2EE Applications J2EE modules JMS or JDBC modules J2EE Libraries
Virtual Host	A configured host name that routes requests for a particular DNS name to a WebLogic Server instance or cluster. See Configuring Virtual Hosting in *Designing and Configuring WebLogic Server Environments* for more information.	Web Applications
JMS Server	A JMS Server configured in a Weblogic Server domain	A JMS queue, topic, or session pool defined within a JMS module*

*When deployed as a standalone application module, a JMS or JDBC resource appears as J2EE deployment in the Administration Console. A standalone JMS application module can be targeted to server, cluster, or virtual host targets; queues defined within a JMS module can be further targeted to a configured JMS server.

Deploying to One or More Targets

To deploy to a single WebLogic Server target, list the configured target name after the `-targets` option to `weblogic.Deployer`. For example, to deploy a Web Application to a configured virtual host named `companyHost`, use the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -deploy -targets companyHost
c:\localfiles\myWebApp.ear
```

Specify multiple targets using a comma-separated list, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -deploy -targets ManagedServer-1,ManagedServer-2
   c:\localfiles\myapp.ear
```

Deploying to a Cluster Target

If you specify a cluster target, WebLogic Server targets all server instances in the cluster by default. This corresponds to homogenous module deployment, which is recommended in most clusters. If you want to deploy a module only to a single server in the cluster (that is, "pin" a module to a servers), specify the individual server instance name, rather than the cluster, as the target. This type of deployment is less common, and should be used only in special circumstances where pinned services are required. See Understanding Cluster Configuration and Application Deployment in *Using WebLogic Server Clusters* for more information.

Note: Pinning a deployment to a subset of server instances in a cluster (rather than to a single server) is not recommended and will generate a warning message.

When you deploy an application to a cluster target, WebLogic Server ensures that the deployment successfully deploys on all available members of the cluster. If even a single, available WebLogic Server instance in the cluster cannot deploy the application, the entire deployment fails and no servers in the cluster start the application. This helps to maintain homogeneous deployments to the cluster, because deployment operations succeed or fail as a logical unit.

If a clustered server is unreachable at the time of deployment (for example, because of a network failure between the Administration Server and a Managed Server, or because a cluster member is shut down) that server does not receive the deployment request until the network connection is restored. This default behavior ensures that most deployment operations succeed, even when servers are taken offline.

Enforcing Consistent Deployment to all Configured Cluster Members

The default cluster deployment behavior ensures homogeneous deployment for all clustered server instances that can be reached at the time of deployment. However, if the Administration Server cannot reach one or more clustered servers due to a network outage, those servers do not receive the deployment request until the network connection is restored. For redeployment operations, this can lead to a situation where unreachable servers use an older version of the deployed application, while reachable servers use the newer version. When the network connection is restored, previously-disconnected servers may abruptly update the application as they receive the delayed redeployment request.

It is possible to change WebLogic Server's default deployment behavior for clusters by setting the `ClusterConstraintsEnabled` option when starting the WebLogic Server domain. The `ClusterConstraintsEnabled` option enforces strict deployment for all servers configured in a cluster. `ClusterConstraintsEnabled` ensures that a deployment to a cluster succeeds only if all members of the cluster are reachable and all can deploy the specified files.

Warning: Do not use the `ClusterConstraintsEnabled` option unless you have an extremely reliable network configuration, and you can guarantee that all cluster members are always available to receive deployment and redeployment requests. With `ClusterConstraintsEnabled`, WebLogic Server will fail all deployment operations to a cluster if any clustered server is unavailable, even if a single server has been shut down for maintenance.

To set the `ClusterConstraintsEnabled` for the domain when you start the Administration Server, include the appropriate startup argument:

- `-DClusterConstraintsEnabled=true` enforces strict cluster deployment for servers in a domain.
- `-DClusterConstraintsEnabled=false` ensures that all available cluster members deploy the application or module. Unavailable servers do not prevent successful deployment to the available clustered instances. This corresponds to the default WebLogic Server deployment behavior.

Using Module-Level Targeting for Deploying an Enterprise Application

An Enterprise Application (EAR file) differs from other deployment units because an EAR can contain other module types (WAR and JAR archives). When you deploy an Enterprise Application using the Administration Console, you can target all of the archive's modules together as a single deployment unit, or target individual modules to different servers, clusters, or virtual hosts.

You can also use module-level targeting to deploy only a subset of the modules available in an EAR. This can simplify packaging and distribution of applications by packaging multiple modules in a single, distributable EAR, but targeting only the modules you need to each domain.

Module-Targeting Syntax

To target individual modules in an Enterprise Application, use the ***`module_name`***`@`***`target_name`*** syntax. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myEnterpriseApp
   -targets module1@myserver1,module2@myserver2,module3@myserver3
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

To target Web Application modules that are part of an `.ear` file, use the Web Application's context root name as the module name. For example, if the `application.xml` file for a file, `myEnterpriseApp.ear`, defines:

```
<module>
  <web>
    <web-uri>myweb.war</web-uri>
    <context-root>/welcome</context-root>
  </web>
</module>
```

you can deploy only the Web Application module by using the syntax:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mywebapplication -targets welcome@myserver1
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

Deploying JDBC and JMS Application Modules

Standalone JDBC and JMS application modules can be deployed similar to standalone J2EE modules. For a standalone JDBC or JMS application module, the target list determines the WebLogic Server domain in which the module is available. JNDI names specified within an application module are bound as global names and available to clients. For example, if you deploy a standalone JDBC application module to a single-server target, then applications that require resources defined in the JDBC module can only be deployed to the same server instance. You can target application modules to multiple servers, or to WebLogic Server clusters to make the resources available on additional servers.

If you require JDBC or JMS resources to be available to all servers in a domain, create system modules, rather than deployable application modules. See Programming WebLogic JMS and Programming WebLogic JDBC for more information.

Targeting Application-Scoped JMS and JDBC Modules

JMS and JDBC application modules can also be included as part of an Enterprise Application, as an application-scoped resource module. Application-scoped resource modules can be targeted independently of other EAR modules during deployment, if necessary, by using module-level targeting syntax. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myEnterpriseApp
   -targets
myWebApp@myCluster,myJDBCModule@myserver1,myEJBModule@myserver1
   -stage -deploy c:\localfiles\myEnterpriseApp.ear
```

Using Sub-Module Targeting with JMS Application Modules

Certain JMS resources defined within a JMS application module can be further targeted to a JMS Server available on the module's target during deployment. These resources are known as submodules. Certain types of submodule require deployment to a JMS Server, such as:

- Quotas
- Queues
- Topics

Other submodules can be targeted to JMS Servers as well as WebLogic Server instances and clusters:

- Connection factories
- Foreign servers
- Uniform or weighted distributed topics
- Uniform or weighted distributed queues

During deployment, WebLogic Server selects default JMS Server targets for submodules in a JMS application module, as described in Programming WebLogic JMS. To specify submodule targets at deployment time, you must use an extended form of the module targeting syntax with the `-submoduletargets` option to `weblogic.Deployer`.

For a standalone JMS module, the submodule targeting syntax is: `-submoduletargets` *`submodule_name@target_name`*. For example, to deploy a standalone JMS module on a single server instance, targeting the submodule `myQueue` to a JMS Server named JMSServer1, enter the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myJMSModule
   -targets ManagedServer1 -submoduletargets myQueue@JMSServer1
   -deploy c:\localfiles\myJMSModule.xml
```

For an application-scoped JMS module in an EAR, use the syntax: `submodule_name@module_name@target_name` to target a submodule. For example, if the queue in the above example were instead packaged as part of an enterprise application, you would use the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myEnterpriseApp
   -targets ManagedServer1 -submoduletargets
myQueue@myJMSModule@JMSServer1
   -deploy c:\localfiles\myEnterpriseApp.ear
```

Controlling Deployment File Copying with Staging Modes

The deployment *staging mode* determines how deployment files are made available to target servers that must deploy an application or standalone module. WebLogic Server provides three different options for staging files: stage mode, nostage mode, and external_stage mode. The following table describes the behavior and best practices for using the different deployment staging modes.

Table 5-2 Application Deployment Staging Modes

Deployment Staging Mode	Behavior	When to Use
stage	The Administration Server first copies the deployment unit source files to the staging directories of target servers. (The staging directory is named `stage` by default, and it resides under the target server's root directory.) The target servers then deploy using their local copy of the deployment files.	• Deploying small or moderate-sized applications to multiple WebLogic Server instances. • Deploying small or moderate-sized applications to a cluster.

Deployment Staging Mode	Behavior	When to Use
nostage	The Administration Server does not copy deployment unit files. Instead, all servers deploy using the same physical copy of the deployment files, which must be directly accessible by the Administration Server and target servers. With nostage deployments of exploded archive directories, WebLogic Server automatically detects changes to a deployment's JSPs or Servlets and refreshes the deployment. (This behavior can be disabled if necessary.)	• Deploying to a single-server domain. • Deploying to a cluster on a multi-homed machine. • Deploying very large applications to multiple targets or to a cluster where deployment files are available on a shared directory. • Deploying exploded archive directories that you want to periodically redeploy after changing content. • Deployments that require dynamic update of selected Deployment Descriptors via the Administration Console.
external_stage	The Administration Server does not copy deployment files. Instead, the Administrator must ensure that deployment files are distributed to the correct staging directory location before deployment (for example, by manually copying files prior to deployment). With external_stage deployments, the Administration Server requires a copy of the deployment files for validation purposes. Copies of the deployment files that reside in target servers' staging directories are not validated before deployment.	• Deployments where you want to manually control the distribution of deployment files to target servers. • Deploying to domains where third-party applications or scripts manage the copying of deployment files to the correct staging directories. • Deployments that do not require dynamic update of selected Deployment Descriptors via the Administration Console (not supported in external_stage mode). • Deployments that do not require partial redeployment of application components.

A server's staging directory is the directory in which the Administration Server copies deployment files for stage mode deployments. It is also the directory in which deployment files must reside before deploying an application using external_stage mode.

Most deployments use either stage or nostage modes, and the WebLogic Server deployment tools use the appropriate default mode when you deploy an application or module. The sections that follow explain how to explicitly set the deployment staging mode when deploying an application or module.

Using Nostage Mode Deployment

In nostage mode, the Administration Server does not copy the archive files from their source location. Instead, each target server must access the archive files from a single source directory for deployment. The staging directory of target servers is ignored for nostage deployments.

For example, if you deploy a J2EE Application to three servers in a cluster, each server must be able to access the same application archive files (from a shared or network-mounted directory) to deploy the application.

Note: The source for the deployment files in nostage mode is the path provided by the user at deployment time (as opposed to stage mode, where the source is the path in each server's staging directory). However, even in nostage mode, WebLogic Server copies out parts of the deployment to temporary directories. This enables users to update entire archived deployments or parts of archived deployments.

In nostage mode, the Web Application container automatically detects changes to JSPs and servlets. Nostage also allows you to later update only parts of an application by updating those parts in one file system location and then redeploying.

The Administration Console uses nostage mode as the default when deploying only to the Administration Server (for example, in a single-server domain). `weblogic.Deployer` uses the target server's staging mode, and Administration Servers use stage mode by default. You can also select nostage mode if you run a cluster of server instances on the same machine, or if you are deploying very large applications to multiple machines that have access to a shared directory. Deploying very large applications in nostage mode saves time during deployment because no files are copied.

Syntax for Using Stage Mode

To use nostage mode, specify `-nostage` as an option to `weblogic.Deployer`, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mydeploymentname
```

```
-targets myserver1,myserver2,myserver3 -nostage
-deploy c:\localfiles\myapp.ear
```

Using Stage Mode Deployment

In stage mode, the Administration Server copies the deployment files from their original location on the Administration Server machine to the staging directories of each target server. For example, if you deploy a J2EE Application to three servers in a cluster using stage mode, the Administration Server copies the deployment files to directories on each of the three server machines. Each server then deploys the J2EE Application using its local copy of the archive files.

When copying files to the staging directory, the Administration Server creates a subdirectory with the same name as the deployment name. So if you deployed using the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mytestear -stage -targets mycluster
   -deploy c:\bea\weblogic81\samples\server\medrecd\dist\physicianEar
```

a new directory, `mytestear`, would be created in the staging directory of each server in `mycluster`. If you do not specify a deployment name, a default deployment name (and staging subdirectory) is used:

- For exploded archive deployments, the deployment name and staging subdirectory are the name of the directory you deployed (*`physicianEar`* in the example above).
- For archived deployments, the default deployment name is the name of the archive file without the extension. For example, if you deploy `physicianEar.ear`, the deployment name and staging subdirectory are `physicianEar`.

The Administration Console uses stage mode as the default mode when deploying to more than one WebLogic Server instance. `weblogic.Deployer` uses the target server's staging mode as the default, and managed servers use stage mode by default.

Stage mode ensures that each server has a local copy of the deployment files on hand, even if a network outage makes the Administration Server unreachable. However, if you are deploying very large applications to multiple servers or to a cluster, the time required to copy files to target servers can be considerable. You may consider nostage mode to avoid the overhead of copying large files to multiple servers.

Syntax for Using Stage Mode

To use stage mode, specify `-stage` as an option to `weblogic.Deployer`, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mydeploymentname
   -targets myserver1,myserver2,myserver3 -stage
   -deploy c:\localfiles\myapp.ear
```

Using External_stage Mode Deployment

External_stage mode is similar to stage mode, in that target servers deploy using local copies of the deployment files. However, the Administration Server does not automatically copy the deployment files to targeted servers in external_stage mode; instead, you must copy the files to the staging directory of each target server before deployment. You can perform the copy manually or use automated scripts.

Within each target server's staging directory, deployment files must be stored in a subdirectory that reflects the deployment name. This can either be the name you type in for the deployment, or the default deployment name (the name of the exploded archive directory, or the name of the archive file without its file extension).

External_stage mode is the least common deployment staging mode. It is generally used only in environments that are managed by third-party tools that automate the required copying of files. You may also choose to use external_stage mode when you are deploying very large applications to multiple machines and you do not have a shared file system (and cannot use nostage mode). Using external stage in this scenario decreases the deployment time because files are not copied during deployment.

Syntax for Using external_stage Mode

To deploy an application using external_stage mode:

1. Make sure that the deployment files are accessible to the Administration Server. (See "Uploading Deployment Files from a Remote Client" on page 5-4.)

2. On each target server for the deployment, create a subdirectory in the staging directory that has the same name as the deployment name. For example, if you will specify the name `myEARExternal` for the deployment name, create a `myEARExternal` subdirectory in the staging directories for each target server.

   Note: If you do not specify a deployment name at deployment time, WebLogic Server selects a default name. See "Choosing a Deployment Name" on page 3-8 for more information.

3. If you are deploying a versioned application for use with production redeployment, create a subdirectory beneath the application directory for the version of the application you are deploying. For example, if you are deploying `myEARExternal` at version level 2.0Beta, the deployment files must reside in a subdirectory of each target server's staging directory named `myEARExternal\2.0Beta`.
4. Copy the deployment files into the staging subdirectories you created in Step 2 or 3 above.
5. Deploy the application or module using the `weblogic.Deployer` utility. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -name weblogic
   -password weblogic -external_stage -name myEARExternal
   -deploy c:\myapps\myear
```

Changing the Default Staging Behavior for a Server

The server staging mode specifies the default deployment mode for a server if none is specified at deployment time. For example, the server staging mode is used if you deploy an application or standalone module using `weblogic.Deployer` and you do not specify a staging mode.

Notes: You can only change the server staging mode by using the Administration Console or by directly changing the `ServerMBean` via JMX.

Changing the server staging mode does not affect existing applications. If you want to change the staging mode for an existing application, you must undeploy the application deployment and then redeploy it with the new staging mode.

To set the server staging mode using the Administration Console:

1. Expand the Servers node in the left pane.
2. Select the name of the server instance that you want to configure.
3. Select the Configuration->Deployment tab in the right pane to display the current staging mode.
4. Select stage, nostage, or external_stage from the Staging Mode menu. These modes correspond to the staging modes described in Table 5-2 on page 11, and apply only to the selected server instance.
5. Enter a path in the Staging Directory Name attribute to store staged deployment files. The path is relative to the root directory of the selected server.

6. If you are configuring the staging mode for the Administration Server, also specify an Upload Directory Name, relative to the server's root directory. This is the directory where the Administration Server stores uploaded files for deployment to servers and clusters in the domain.
7. Click Apply to change the staging mode and directory.

Distributing Applications to a Production Environment

Distributing an application copies the application's deployment files to all target servers, validates the application, and places the application in Administration mode. Administration mode restricts access to an application to a configured Administration channel. You can distribute an application to a production environment (or distribute a new version of an application) without opening the application to external client connections.

While in Administration mode, you can connect to an application only via a configured Administration channel. This allows you to perform final ("sanity") checking of the application directly in the production environment without disrupting clients. After performing final testing, you can either undeploy the application to make further changes, or start the application to make it generally available to clients.

Distributing an Application

To distribute an application, use the `weblogic.Deployer` **`-distribute`** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -distribute -name myTestDeployment
     /myDeployments/myApplication/
```

After WebLogic Server distributes the deployment files, you must use a configured Administration channel to access the application. See Configuring Network Resources in *Designing and Configuring WebLogic Server Environments*.

Starting a Distributed Application

After performing final testing of a distributed application using a configured Administration channel, you can open the application to new client connections by using the `weblogic.Deployer` **`-start`** command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -start -name myTestDeployment
```

Deploying Shared J2EE Libraries and Dependent Applications

J2EE library support in WebLogic Server 9.0 provides an easy way to share one or more J2EE modules or JAR files among multiple Enterprise Applications. A J2EE library is a standalone J2EE module, multiple J2EE modules packaged in an Enterprise Application (EAR), or a plain JAR file that is registered with the J2EE application container by deploying it. After a J2EE library has been registered, you can deploy Enterprise Applications that reference the library. Each referencing application receives a copy of the shared J2EE library module(s) on deployment, and can use those modules as if they were packaged as part of the application itself.

See Overview of J2EE Libraries and Optional Packages in *Developing Applications with WebLogic Server* for more information.

Note: BEA documentation and WebLogic Server utilities use the term *library* to refer to both J2EE libraries and optional packages. Optional packages are called out only when necessary.

Understanding Deployment Behavior for Shared Libraries

WebLogic Server 9.0 supports shared J2EE libraries by merging the shared files with a referencing application when the referencing application is deployed.

A J2EE library or optional package is first registered with one or more WebLogic Server instances or clusters by deploying the library and indicating that the deployment is a library (see "Registering Libraries with WebLogic Server" on page 5-19). Deploying a library or package does not activate the deployment on target servers. Instead, WebLogic Server distributes the deployment files to target servers (if nostage deployment mode is used) and records the location of the deployment files, the deployment name, and any version string information for the library or package.

When an application that references a shared library or package is deployed, WebLogic Server checks the names and version string requirements against the libraries registered with the server. If an exact match for a library or package name is not found, or if the version requirements are not met, the application deployment fails.

If WebLogic Server finds a name and version string match for all of the libraries referenced in the application, the server adds the libraries' classes to the classpath of the referencing application and merges deployment descriptors from both the application and libraries in memory. The resulting deployed application appears as if the referenced libraries were bundled with the application itself.

The contents of a J2EE library are loaded into classloaders in the same manner as any other J2EE modules in an enterprise application. For example, EJB modules are loaded as part of the referencing application's classloader, while Web Application modules are loaded in classloaders beneath the application classloader. If a shared J2EE library consists of an EAR, any classes stored in the EAR's `APP-INF/lib` or `APP-INF/classes` subdirectory are also available to the referencing application.

You cannot undeploy any J2EE libraries or optional packages that are referenced by a currently-deployed application. If you need to undeploy a shared library or package, you must first undeploy all applications that use the shared files. For regular application maintenance, you should deploy a new version of a shared library or package and redeploy referencing applications to use the newer version of the shared files. See Editing Manifest Entries for Shared Libraries in *Developing Applications with WebLogic Server* for more information.

Registering Libraries with WebLogic Server

A shared J2EE library is a standard J2EE module or Enterprise Application that is registered with a WebLogic Server container as a library. To register a J2EE library with a WebLogic Server container, perform the following steps:

1. Ensure that the deployment file(s) you are working with represent a valid J2EE library or optional package. See Creating Shared J2EE Libraries in *Developing Applications with WebLogic Server*.

2. Select the WebLogic Server targets to which you will register the library or package. Shared libraries must be registered to the same WebLogic Server instances on which you plan to deploy referencing applications. (You may consider deploying libraries to all servers in a domain, so that you can later deploy referencing applications as needed.)

3. Register the library or package by deploying the files to your selected target servers, and identifying the deployment as a library or package with the `-library` option. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -deploy -targets myserver1,myserver2
     -library /deployments/myLibraryApplication/
```

Specifying Library Versions at Deployment

As a best practice, your development team should always include version string information for a library or optional package in the manifest file for the deployment. See Editing Manifest Attributes for Shared Libraries in *Developing Applications with WebLogic Server* for more information about version string requirements and best practices.

If you are deploying a library or package that does not include version string information, you can specify it at the command line using one or both of the following options:

- `libspecver`—Specifies a specification version for the library or package.
- `libimplver`—Specifies an implementation version for the library or package.

For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -deploy -targets myserver1,myserver2
     -library -libspecver 700 -libimplversion 7.0.0.1Beta
     /deployments/myLibraryApplication/
```

Notes: If both the manifest file and the `weblogic.Deployer` command line specify version information, and the values do not match, the deployment will fail.

If you initially registered a library without specifying a specification or implementation version, you must undeploy the library before registering a newer version and specifying version string information.

You can register multiple versions of a library or package, but each version must use the same version strings. For example, you cannot register a library having only a specification version, and then register a new version of the library having both a specification and an implementation version.

Deploying Applications that Reference Libraries

After you have deployed the required library and package versions, you can deploy Enterprise Applications and modules that reference the libraries. Successfully deploying a referencing application requires that two conditions are met:

- that all referenced libraries have been registered on the application's target servers
- that the registered libraries meet the version requirements of the referencing application

There is no special syntax for deploying a referencing application. Deploy the application as you would a standard Enterprise Application or J2EE module.

Note that referencing applications can be configured with varying levels of version requirements for shared libraries. An application might be configured to require:

- the latest version of a shared library or package (the latest registered version), or
- a minimum version of a library or package (or a higher version), or

- an exact version of a library or package.

If you cannot deploy a referencing application because of version requirements, try registering the required version of the conflicting library or package. Or, consult with your development team to determine whether the version requirements of the application can be relaxed. See Referencing Libraries in an Enterprise Application in *Developing Applications with WebLogic Server* for more information.

Auto-Deploying Applications in Development Domains

Notes: Auto-deployment is a method for quickly deploying an application to a standalone server (Administration Server) for evaluation or testing. It is recommended that this method be used only in a single-server development environment. Use of auto-deployment in a production environment or for deployment to Managed Servers is not recommended.

BEA recommends that you use the WebLogic Server split development directory and `wldeploy` ant task, rather than auto-deployment, when developing an application. See Creating a Split Development Directory in *Developing Applications with WebLogic Server*.

If auto-deployment is enabled, when an application is copied into the `\applications` directory of the Administration Server, the Administration Server detects the presence of the new application and deploys it automatically (if the Administration Server is running). If WebLogic Server is not running when you copy the application to the `\applications` directory, the application is deployed the next time the WebLogic Server Administration Server is started. Auto-deployment deploys only to the Administration Server.

Notes: Due to the file locking limitations of Windows NT, if an application is exploded, all the modules within the application must also be exploded. In other words, you cannot use auto-deployment with an exploded application or module that contains a JAR file.

Auto-deployment is intended for use with a single server target in a development environment. If you use other tools, such as the Administration Console, to add targets to an auto-deployed, exploded application, redeploying the application does not propagate changes to the new target servers.

Enabling and Disabling Auto-Deployment

You can run a WebLogic Server domain in two different modes: development and production.

Development mode enables a WebLogic Server instance to automatically deploy and update applications that are in the `domain_name/applications` directory (where `domain_name` is the name of a WebLogic Server domain). In other words, development mode lets you use auto-deploy. Production mode disables the auto-deployment feature.

By default, a WebLogic Server domain runs in development mode. To specify the mode for a domain, see *Creating and Configuring Domains Using the Configuration Wizard*.

Auto-Deploying, Redeploying, and Undeploying Archived Applications

To auto-deploy an archived application, copy its archive file to the `/applications` directory. WebLogic Server automatically sets the application's deployment mode to stage mode.

A deployment unit that was auto-deployed can be dynamically redeployed while the server is running. To dynamically redeploy, copy the new version of the archive file over the existing file in the `/applications` directory.

To undeploy an archived deployment unit that was auto-deployed, delete the application from the `/applications` directory. WebLogic Server stops the application and removes it from the configuration.

Auto-Deploying, Redeploying, and Undeploying Exploded Archives

To auto-deploy an application in exploded archive format, copy the entire exploded archive directory to the `/applications` directory. WebLogic Server automatically deploys exploded archive applications using the nostage deployment mode. If you deploy an exploded EAR directory that contains archived modules (JAR files), the JAR files are locked during the deployment.

When an application has been auto-deployed in exploded archive format, the Administration Server periodically looks for a file named `REDEPLOY` in the exploded application directory. If the timestamp on this file changes, the Administration Server redeploys the exploded directory.

To redeploy files in an exploded application directory:

1. When you first deploy the exploded application directory, create an empty file named `REDEPLOY`, and place it in the `WEB-INF` or `META-INF` directory, depending on the application type you are deploying:

   An exploded enterprise application has a `META-INF` top-level directory; this contains the `application.xml` file.

An exploded Web application has a `WEB-INF` top-level directory; this contains the `web.xml` file.

An exploded EJB application has a `META-INF` top-level directory; this contains the `ejb-jar.xml` file.

An exploded connector has a `META-INF` top-level directory; this contains the `ra.xml` file.

Note: The `REDEPLOY` file works only for an entire deployed application or standalone module. If you have deployed an exploded Enterprise Application, the `REDEPLOY` file controls redeployment for the entire application—not for individual modules (for example, a Web Application) within the Enterprise Application. If you deploy a Web Application by itself as an exploded archive directory, the `REDEPLOY` file controls redeployment for the entire Web Application.

2. To redeploy the exploded application, copy the updated files over the existing files in that directory.
3. After copying the new files, modify the `REDEPLOY` file in the exploded directory to alter its timestamp.

When the Administration Server detects the changed timestamp, it redeploys the contents of the exploded directory.

Note: You must `touch` the `REDEPLOY` file (alter its timestamp) any time you wish to trigger redeployment of an auto-deployed application. Even if you modify an application while a server is shut down, you must `touch REDEPLOY` to ensure that changes are applied when the server next starts up.

To undeploy an application that was auto-deployed in exploded format, use the `weblogic.Deployer -undeploy` command, or use the Administration Console to remove the deployment configuration. Then remove the application files from the `/applications` directory.

Best Practices for Deploying Applications

BEA recommends the following best practices when deploying applications:

- The Administration Server in a multiple-server domain should be used only for administration purposes. Although you can deploy to the Administration Server in a multiple-server domain, this practice is not recommended.
- Package deployment files in an archive format (`.ear`, `.jar`, `.war`, and so forth) when distributing files to different users or environments.

- Check the scenarios described under "Deployment Archive Files Versus Exploded Archive Directories" on page 3-6 before deploying. In many cases it is more convenient to deploy applications in exploded format rather than archive format.
- The Administration Console, `weblogic.Deployer` tool, `wldeploy` Ant task, and WLST all provide similar functionality for deploying applications:
  – Use the Administration Console for interactive deployment sessions where you do not know the exact location of deployment files, or the exact names of target servers or deployed applications.
  – Use `weblogic.Deployer` to integrate deployment commands with existing administrative shell scripts or automated batch processes.
  – Use `wldeploy` in conjunction with the split development directory for developing and deploying new applications. `wldeploy` can also be used in place of `weblogic.Deployer` in administration environments that use Ant, rather than shell scripts.
  – Use WLST when you want to create automated scripts that perform deployment tasks.
- Use `wldeploy`, rather than the `applications` directory, to deploy your own applications during development. The `applications` directory is best used for quickly running new applications in a test or temporary. For example, if you download a sample application and want to evaluate its functionality, the `applications` directory provides a quick way to deploy the application into a development server.

Updating Applications in a Production Environment

The following sections describe how to use WebLogic Server redeployment to update applications and parts of applications in a production environment:

Overview of Redeployment Strategies

In a production environment, deployed applications frequently require 24x7 availability in order to provide uninterrupted services to customers and internal clients. WebLogic Server 9.0 provides flexible redeployment strategies to help you update or repair production applications based on their required level of availability.

Production Redeployment

The production redeployment strategy works by deploying a new version of an updated application alongside an older version of the same application. WebLogic Server automatically manages client connections so that only new client requests are directed to the new version. Clients already connected to the application during the redeployment continue to use the older version of the application until they complete their work, at which point WebLogic Server automatically retires the older application.

Production redeployment enables you to update and redeploy an application in a production environment without stopping the application or otherwise interrupting the application's availability to clients. Production redeployment saves you the trouble of scheduling application downtime, setting up redundant servers to host new application versions, manually managing client access to multiple application versions, and manually retiring older versions of an application (see "Redeploying Applications and Modules In-Place" on page 6-15 for information about these manual steps).

Production redeployment can be used in combination with the **`-distribute`** command to place a new version of an application in Administration mode. This allows you to perform final sanity testing of a new application version directly in the production environment before making it available to clients. See "Distributing a New Version of a Production Application" on page 6-11.

Production redeployment is supported only for certain J2EE application types. See "Using Production Redeployment to Update Applications" on page 6-4.

In-Place Redeployment

The in-place redeployment strategy works by immediately replacing a running application's deployment files with updated deployment files. In contrast to production redeployment, in-place redeployment of an application or standalone J2EE module does not guarantee uninterrupted service to the application's clients. This is because WebLogic Server immediately removes the running classloader for the application and replaces it with a new classloader that loads the updated application class files.

Note: In-place redeployment is the redeployment strategy used in previous versions of WebLogic Server.

WebLogic Server uses the in-place redeployment strategy for J2EE applications that do not specify a version identifier, and for J2EE applications and standalone modules that are not supported with the Production Redeployment strategy. You should ensure that in-place redeployment of applications and standalone J2EE modules takes place only during scheduled downtime for an application, or when it is not critical to preserve existing client connections to an application. See "Using In-Place Redeployment for Applications and Standalone Modules" on page 6-14 for more information.

WebLogic Server uses the in-place redeployment strategy when performing partial redeployment of a deployed application or module. Partial redeployment can replace either static files in an application, or entire J2EE modules within an Enterprise Application deployment, as described below.

Partial Redeployment of Static Files

WebLogic Server enables you to redeploy selected files in a running application, rather than the entire application at once. This feature is generally used to update static files in a running Web Application, such as graphics, static HTML pages, and JSPs. Partial redeployment is available only for applications that are deployed using an exploded archive directory.

Partial redeployment of static files does not affect existing clients of the application. WebLogic Server simply replaces the static files for the deployed application, and the updated files are served to clients when requested. See "Updating Static Files in a Deployed Application" on page 6-18.

Partial Redeployment of J2EE Modules

Partial redeployment also enables you to redeploy a single module or subset of modules in a deployed Enterprise Application. Again, partial deployment is supported only for applications that are deployed using an exploded archive directory.

Note that redeployment for modules in an Enterprise Application uses the in-place redeployment strategy, which does not guarantee uninterrupted client access to the module. For this reason, you should ensure that partial redeployment of J2EE modules in an EAR takes place only during scheduled application downtime, or when it is not critical to preserve client access to the application. See "Using Partial Redeployment for J2EE Module Updates" on page 6-17.

Understanding When to Use Different Redeployment Strategies

The following table summarizes each WebLogic Server redeployment strategy and describes the scenarios in which you would use each strategy.

Table 6-1 Summary of Redeployment Strategies

Redeployment Strategy	Summary	Usage:
Production Redeployment	Redeploys a newer version of an application alongside an existing version of the application.	• Upgrading Web Applications and Enterprise Applications that demand uninterrupted client access.
In-Place Redeployment of Applications and Modules	Application classloaders are immediately replaced with newer classloaders to load the updated application class files. WebLogic Server does not guarantee uninterrupted client access during redeployment, and existing clients' state information may be lost.	• Replacing applications that have been taken off-line for scheduled maintenance. • Upgrading applications that do not require uninterrupted client access.
Partial Redeployment of Static Files	HTML, JSPs, Graphics Files, or other static files are immediately replaced with updated files.	• Updating individual Web Application files that do not affect application clients.
Partial Redeployment of J2EE Modules	Module classloaders are immediately replaced with newer classloaders to load the updated class files. WebLogic Server does not guarantee uninterrupted client access to the module during redeployment, and existing clients' state information may be lost.	• Replacing a component of an Enterprise Application that has been taken off-line for scheduled maintenance, or that does not require uninterrupted client access.

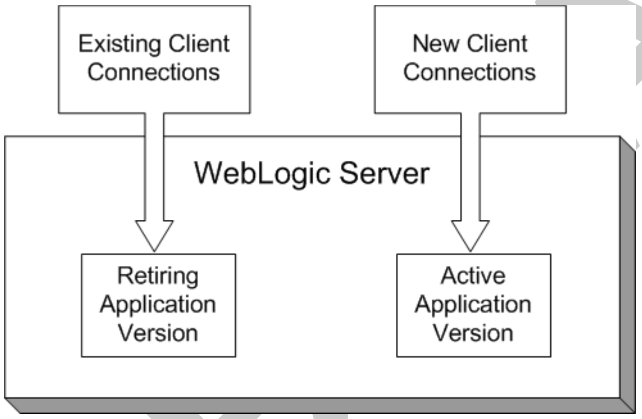
Using Production Redeployment to Update Applications

WebLogic Server 9.0 enables you to redeploy a new, updated version of a production application without affecting existing clients of the application, and without interrupting the availability of the application to new client requests.

How Production Redeployment Works

WebLogic Server performs production redeployment by deploying a new version of an application alongside an older, running version of the application. While redeployment is taking place, one version of the application "active," while the other version is "retiring." The active application version receives all new client connection requests for the application, while the retiring application version processes only those client connections that existed when redeployment took place. WebLogic Server undeploys the retiring application version after all existing clients of the application have finished their work, or when a configured timeout is reached.

Figure 6-1 Production Redeployment


Existing Client Connections
New Client Connections
WebLogic Server
Retiring Application Version
Active Application Version

When you redeploy a new version of an application, WebLogic Server treats the newly-deployed application version as the active version, and begins retiring the older version. During the retirement period, WebLogic Server automatically tracks the application's HTTP sessions and in-progress transactions. WebLogic Server tracks HTTP each session until the session completes or has timed out. In-progress transactions are tracked until the transaction completes, rolls-back, or reaches the transaction timeout period.

You can roll back the production redeployment process by making the older application version active. This may be necessary if, for example, you determine that there is a problem with the newer version of the application, and you want WebLogic Server to begin moving clients back to the older version.

Production Redeployment In Clusters

In a WebLogic Server cluster, each clustered server instance retires its local deployment of the retiring application version based on the current workload. This means that an application version may be retired on some clustered server instances before it is retired on other servers in the cluster. Note, however, that in a cluster failover scenario, client failover requests are always handled by the same application version on the secondary server, if the application version is still available. If the same application version is not available on the secondary server, the failover does not succeed.

Requirements and Restrictions for Using Production Redeployment

In order to use the production redeployment feature, an application must meet certain requirements during the development and deployment phases.

Development Requirements

The production redeployment strategy is provided only for standalone Web Application (WAR) modules and Enterprise Applications (EARs) whose clients access the application via a Web Application (HTTP). Additional support is provided for Enterprise Applications that are accessed by inbound JMS messages from a global JMS destination, or from inbound JCA requests.

Production redeployment *is not supported* for standalone EJB, RAR, or Web Service modules. WebLogic Server redeploys those modules using the in-place redeployment strategy, which interrupts existing clients of the module and makes the module unavailable during the redeployment process.

Production redeployment is not supported for Enterprise Applications that are accessed by Web Services, Java clients, or any non-HTTP access method. Your development and design team must ensure that applications using production redeployment are not accessed by an unsupported client. WebLogic Server does not detect when unsupported clients access the application, and does not preserve unsupported client connections during production redeployment.

During development, applications must be designed to meet specific requirements in order to ensure that multiple versions of the application can safely coexist in a WebLogic Server domain during production redeployment. See Developing Versioned Applications for Production Redeployment in *Developing WebLogic Server Applications* for information about the programming conventions required for applications to use production redeployment.

Warning: Because the production redeployment strategy requires an application to observe certain programming conventions, use production redeployment only with applications that are approved by your development and design staff. Using production redeployment with an application that does not follow BEA's programming conventions can lead to corruption of global resources or other undesirable application behavior.

Deployment Requirements

A deployed application must specify a version number before you can perform subsequent production redeployment operations on the application. In other words, you cannot deploy a non-versioned application, and later perform production redeployment with a new version of the application.

Restrictions on Production Redeployment Updates

WebLogic Server can host a maximum of two different versions of an application at one time.

When you redeploy a new version of an application, you cannot change the application's:

- Deployment targets
- Security model
- Persistent store settings

To change any of the above features, you must first undeploy the active version of the application.

Specifying an Application Version Identifier

WebLogic Server attempts to use the production redeployment strategy when the currently-deployed application and the redeployed application specify different versions.

To assign a version identifier to an application, BEA recommends that you store a unique version string directly in the `MANIFEST.MF` file of the EAR or WAR being deployed. Your development process should automatically increment the version identifier with each new application release before packaging the application for deployment.

Maintaining version information in this manner ensures that the production redeployment strategy is used with each redeployment of the application in a production domain. See Developing Versioned Applications for Production Redeployment in *Developing WebLogic Server Applications*.

Assigning a Version Identifier During Deployment and Redeployment

If you are testing the production redeployment feature, or you want to use production redeployment with an application that does not include a version string in the manifest file, the `weblogic.Deployer` tool allows you to manually specify a unique version string using the `-appversion` option when deploying or redeploying an application:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -deploy -name myTestDeployment
     -source /myDeployments/myApplication/91Beta
     -targets myCluster -stage -appversion .91Beta
```

The version string specified with `-appversion` is applied only when the deployment source files do not specify a version string in `MANIFEST.MF`. See Application Version Conventions in *Developing WebLogic Server Applications* for information about valid characters and character formats for application version strings.

Warning: Do not use `-appversion` to deploy or redeploy in a production environment unless you are certain the application follows BEA's programming conventions for production redeployment. Using production redeployment with an application that does not follow the programming conventions can cause corruption of global resources or other undesirable application behavior.

Displaying Version Information for Deployed Applications

The WebLogic Server Administration Console displays both version string information and state information for all deployed applications and modules. To view this information, select the Deployments node in the left-hand pane of the Administration Console.

You can also display version information for deployed applications from the command line using the `weblogic.Deployer` **`-listapps`** command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -listapps
```

Redeploying a New Version of an Application

To redeploy a new version of an application using the production redeployment strategy:

1. Verify that only one version of the application is currently deployed in the domain. See "Displaying Version Information for Deployed Applications" on page 6-8.

2. Verify that the deployed application and the newer application have different version strings:

   a. Use the instructions in "Displaying Version Information for Deployed Applications" on page 6-8 to determine currently-deployed version of the application.

   b. Examine the version string in the `MANIFEST.MF` file of the new application source you want to deploy:

   ```
   jar xvf myApp.ear MANIFEST.MF
   cat MANIFEST.MF
   ```

3. Place the new application deployment files in a suitable location for deployment. BEA recommends that you store each version of an application's deployment files in a unique subdirectory.

   For example, if the currently-deployed application's files are stored in:

   ```
   /myDeployments/myApplication/91Beta
   ```

   You would store the updated application files in a new subdirectory, such as:

   ```
   /myDeployments/myApplication/1.0GA
   ```

   Warning: For nostage or external_stage mode deployments, do not overwrite or delete the deployment files for the older version of the application. The original deployment files are required if you later choose to roll back the retirement process and revert to the original application version.

4. Redeploy the new application version and specify the updated deployment files. If the updated deployment files contain a unique version identifier in the manifest file, use a command similar to:

   ```
   java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
        -password weblogic -redeploy -name myTestDeployment
        -source /myDeployments/myApplication/1.0GA
   ```

   If the new deployment files do not contain a version identifier in the manifest, see "Assigning a Version Identifier During Deployment and Redeployment" on page 6-8.

   By default WebLogic Server makes the newly-redeployed version of the application active for processing new client requests. Existing clients continue to use the older application until their work is complete and the older application can be safely undeployed.

   If you want to specify a fixed time period after which the older version of the application is undeployed (regardless of whether clients finish their work), use the `-retiretimeout` option with the **`-redeploy`** command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -redeploy -name myTestDeployment
     -source /myDeployments/myApplication/1.0GA
     -retiretimeout 300
```

`-retiretimeout` specifies the number of seconds after which the older version of the application is retired. You can also retire the older application immediately by using the **`-undeploy`** command and specifying the older application version, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -undeploy -name myTestDeployment
     -appversion .91Beta
```

5. Verify that both the old and new versions of the application are deployed, and that the correct version of the application is active. See "Displaying Version Information for Deployed Applications" on page 6-8.

Undeploying a Retiring Application

If WebLogic Server has not yet retired an application version, you can immediately undeploy the application version without waiting for retirement to complete. This may be necessary if, for example, an application remains in the retiring state with only one or two long-running client sessions that you do not want to preserve. To force the undeployment of a retiring version of an application, use the **`-undeploy`** command and specify the application version:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -undeploy -name myTestDeployment
     -appversion .91Beta
```

Note: You cannot specify the `-graceful` option to the **`-undeploy`** command when undeploying an application version that is being retired, or waiting for a retirement timeout to occur.

Rolling Back the Production Redeployment Process

Reversing the production redeployment process switches the state of the active and retiring applications and redirects new client connection requests accordingly. Reverting the production redeployment process might be necessary if you detect a problem with a newly-deployed version of an application, and you want to stop clients from accessing it.

To roll back the production redeployment process, issue a second **`-redeploy`** command and specify the deployment source files for the older version, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -redeploy -name myTestDeployment
     -source /myDeployments/myApplication/91Beta
     -retiretimeout 300
```

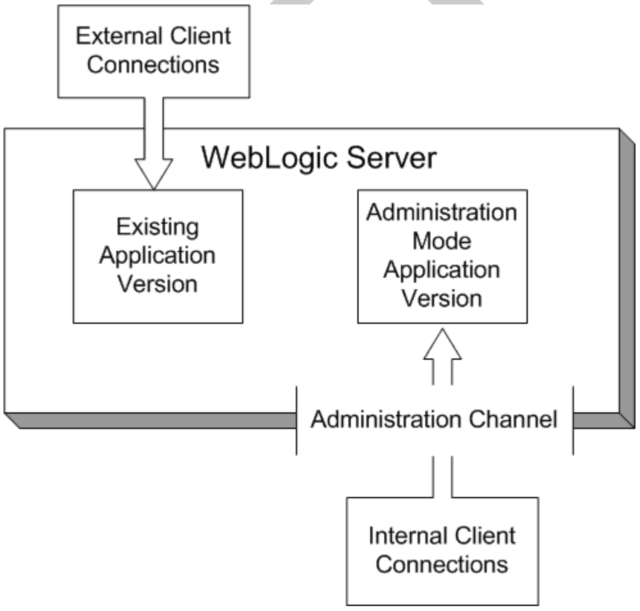
If the deployment files do not contain a version identifier in the manifest, see "Assigning a Version Identifier During Deployment and Redeployment" on page 6-8.

Distributing a New Version of a Production Application

When you distribute a new version of an application, WebLogic Server places the new application version in Administration mode, which makes it available only via a configured Administration channel. External clients cannot access an application that has been distributed.

The older version of the application remains active to process both new and existing client requests. WebLogic Server does not automatically retire the older version of the application when you distribute a newer version.

Figure 6-2 Distributing a New Version of an Application


External Client Connections
WebLogic Server
Existing Application Version
Administration Mode Application Version
Administration Channel
Internal Client Connections

After you complete testing of the new application via an Administration channel, you either undeploy the new application version or start it. Starting the application causes WebLogic Server to route new client connections to the updated application, and begin retiring the older application version.

Steps for Distributing a New Version of an Application

The basic steps for distributing an new version of an application in Administration mode are the same as those documented in "Redeploying a New Version of an Application" on page 6-8. You simply use the `weblogic.Deployer` **`-distribute`** command, rather than the **`-redeploy`** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -distribute -name myTestDeployment
     -source /myDeployments/myApplication/1.0GA
     -appversion 1.0GA
```

While the application is in Administration mode, you must use a configured administration channel to access the application. See Configuring Network Resources in *Designing and Configuring WebLogic Server Environments*.

You can optionally specify the retirement policy or timeout period for distributed applications.

Making a Distributed Application Available to Clients

After performing final testing using the configured Administration channel, you can open the distributed version of the application to new clients connections by using the `weblogic.Deployer` **`-start`** command:

1. Use the Administration Console to view the version and state information of both application versions:

   a. Verify that both versions of the application are still deployed.

   b. Note the version identifier of the application version that is running in Administration mode.

2. Use the `-appversion` option to `weblogic.Deployer` to start the application that was distributed:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -start -name myTestDeployment
     -appversion .91Beta
```

By default WebLogic Server routes new client requests to the application version was previously distributed. Existing clients continue using the older application until their work is complete and the older application can be safely undeployed. If you want to specify a fixed time period after which the older version of the application is undeployed (regardless of whether clients finish their work), use the `-retiretimeout` option:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -start -name myTestDeployment
     -appversion .91Beta -retiretimeout 300
```

`-retiretimeout` specifies the number of seconds after which the older version of the application is retired.

3. Use the Administration Console to verify that the previously-distributed application is now active, and that the former application version is now retiring. See "Displaying Version Information for Deployed Applications" on page 6-8.

Best Practices for Using Production Redeployment

When using production redeployment, keep in mind the following best practices:

- Never specify a version string for a production application unless you are certain the application follows BEA's programming conventions for production redeployment. See Developing Versioned Applications for Production Redeployment in *Developing WebLogic Server Applications*.
- It is easiest to use production redeployment when applications are deployed using the stage deployment mode. With stage mode, WebLogic Server automatically creates a separate directory for each different version of the application that is deployed. These directories are stored in the configured staging directory for the server (by default, the `*server_name*/stage` subdirectory of the domain directory) and are removed when an the associated application version is undeployed.
- If you deploy using nostage mode, store each new version of an application in a dedicated subdirectory. This ensures that you do not overwrite older versions of your deployment files, and allows you to revert to an earlier application version if you detect problems after an update.
- If you deploy using external_stage mode, you must store the deployment files for each application version in the correct version subdirectory of each target server's staging directory. For example, the versioned application files used in the previous sample commands would need to be copied into the subdirectories `/myTestDeployment/.91Beta` and `/myTestDeployment/1.0GA` in each server's staging directory before deployment and redeployment.

Using In-Place Redeployment for Applications and Standalone Modules

The in-place redeployment strategy works by immediately replacing a running application's deployment files with updated deployment files. When used to redeploy entire applications or J2EE modules, in-place redeployment makes the application or module unavailable during the deployment process, and can cause existing clients to lose in-process work. This disruption of client services occurs because application class files and libraries are immediately undeployed from their class loaders and then replaced with updated versions.

Because in-place redeployment of applications and modules adversely affects clients of the application, it should not be used with production applications unless:

- No clients are currently using the application, or
- It is acceptable to lose client access to the application and in-process work during redeployment.

WebLogic Server always performs in-place redeployment for applications that do not include a version identifier. WebLogic Server also uses in-place redeployment for many partial redeployment operations (redeployment commands that affect only a portion of an application). In some cases, using partial redeployment is acceptable with production applications, because the redeployed files do not adversely affect active client connections. Table 6-2 describes each type of partial deployment and its affect on deployed applications.

Table 6-2 Partial Redeployment Behavior

Scope of Partial Redeployment	Redeployment Behavior	Recommended Usage
Graphics files, static HTML files, JSPs	Source files are immediately replaced on-disk and served on the next client request.	Safe for production applications.
J2EE Modules in an Enterprise Application	All files are immediately replaced. Java class files and libraries are unloaded from class loaders and replaced with updated files.	Use only during scheduled application downtime, or when it is not critical to preserve client connections and in-process work.

Scope of Partial Redeployment	Redeployment Behavior	Recommended Usage
Deployment plan with changed tuning parameters (but no changes to resource bindings)	The deployment plan is immediately replaced and the application is updated to use the new configuration.	Safe for all production applications.
Deployment plan with changed resource bindings	If the application is compatible with production redeployment, WebLogic Server uses the production redeployment strategy. If the application cannot use production redeployment, the entire application is redeployed using the in-place redeployment strategy.	Safe for versioned applications that are compatible with production redeployment. See "Using Production Redeployment to Update Applications" on page 6-4. If applications cannot use production redeployment, update the deployment plan only during scheduled application downtime or when it is not critical to preserve client connections and in-process work.

Redeploying Applications and Modules In-Place

To redeploy an entire application or standalone module using the in-place redeployment strategy:

1. If you want to preserve client connections to the application, first take the application offline so and verify that no clients are accessing the application.

   The exact method for taking an application offline will depend on the architecture of your WebLogic Server domain. In most cases, a redundant server or cluster is created to host a separate copy of the application, and load balancing hardware or software manages access to both servers or clusters. To take the application offline, the load balancing policies are changed to roll all client connections from one set of servers or clusters to the redundant set.

2. Place the new application deployment files in a suitable location for deployment. BEA recommends that you store each version of an application's deployment files unique subdirectories.

   For example, if the currently-deployed application's files are stored in:

```
/myDeployments/myApplication/91Beta
```

You would store the updated application files in a new directory, such as:

```
/myDeployments/myApplication/1.0GA
```

3. Redeploy the application and specify the updated deployment source files. To redeploy the application on all configured target servers, specify only the deployment name, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -name myApp
```

If an application was previously deployed to multiple, non-clustered server instances, you can specify a target list to redeploy the application on only a subset of the target servers, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -name myApp
   -targets myserver1,myserver2
```

Note: For applications deployed to a cluster, redeployment always occurs on all target server instances in the cluster. If the application was previously deployed to all servers in the cluster, you cannot subsequently redeploy the application on a subset of servers in the cluster.

4. If you took the server or cluster hosting the application offline, bring the host servers back online after the redeployment completes.

5. If necessary, restore the load balancing policies of your load balancing hardware or software to migrate clients from the temporary servers back to the online production servers.

Best Practices for Redeploying Applications and Modules In-Place

When using in-place redeployment to redeploy entire applications or standalone modules, keep in mind the following:

- Redeploying entire, staged applications may have performance implications due to increased network traffic when deployment files are copied to the Managed Servers. If you have very large applications, consider using the external_stage mode and manually distributing deployment files before you redeploy the application.

- Remember that applications deployed to a WebLogic Server cluster must always be redeployed on all members of the cluster. You cannot redeploy a clustered application to a subset of the cluster.

- To successfully redeploy an Enterprise Application, all of the application's modules must redeploy successfully.
- By default, WebLogic Server destroys current user sessions when you redeploy a Web Application. If you want to preserve Web Application user sessions during redeployment, set `save-sessions-enabled` to "true" in the `container-descriptor` stanza of the `weblogic.xml` deployment descriptor file. Note, however, that the application still remains unavailable while in-place redeployment takes place.

Using Partial Redeployment for J2EE Module Updates

The `weblogic.Deployer` utility uses a different command form if you want to redeploy individual modules of a deployed Enterprise Application. To redeploy a subset of the modules of an Enterprise Application, specify *modulename@servername* in the target server list to identify the modules you want to redeploy. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -name myApp
   -targets mymodule1@myserver1,mymodule2@myserver2
```

Note: The use of `-redeploy` *`module-uri`* is deprecated. Instead, use production redeployment or redeploy the module using the `-targets` *`module@target`* syntax.

If the application was previously deployed to a cluster, you must redeploy the module to the entire cluster, rather than a subset of servers. If you specify a subset of servers in the cluster, `weblogic.Deployer` responds with the error:

```
An attempt to add server target target_name to module module_name has been
rejected . This is because its parent cluster, cluster_name, is aso targeted
by the module.
```

Restrictions for Updating J2EE Modules in an EAR

The following restrictions apply to using partial redeployment for modules in an Enterprise Application:

- If redeploying a single J2EE module in an Enterprise Application would affect other J2EE modules loaded in the same class loader, `weblogic.Deployer` requires that you explicitly redeploy all of the affected modules. If you attempt to use partial redeployment with only a subset of the affected J2EE modules, `weblogic.Deployer` displays the error:

  ```
  Exception:weblogic.management.ApplicationException: [J2EE:160076] You
  must include all of [module_list] in your files list to modify [module]
  ```

- Remember that if you change an application's deployment descriptor files, the container redeploys the entire application even if you attempt a partial redeployment.
- JAR files in `WEB-INF/lib` cannot be redeployed independently of the rest of the Web Application. The container automatically redeploys the entire application, but maintains the state, when redeploying JAR files in `WEB-INF/lib`.

Best Practices for Updating J2EE Modules in an EAR

Keep in mind these best practices when using partial redeployment for Enterprise Application modules:

- When you use partial redeployment to redeploy a J2EE module in an Enterprise Application, all classes loaded in the class loader for the updated module are reloaded. You can define custom class loading hierarchies in the WebLogic Server deployment descriptor to minimize the impact of partial redeployment to other modules in the application. See WebLogic Server Application Classloading in *Developing WebLogic Server Applications* for more information on class loading behavior.
- Classes in the `WEB-INF/classes` directory can be redeployed independently of the rest of the Web Application. You can also deploy only the updated classes (rather than the entire `WEB-INF/classes` directory) by setting the Reload Period for the Web Application. (See weblogic.xml Schema in *Developing Web Applications, Servlets, and JSPs for WebLogic Server* for more information.)
- By default, WebLogic Server destroys current user sessions when you redeploy a Web Application module. If you want to preserve Web Application user sessions during redeployment, set `save-sessions-enabled` to "true" in the `container-descriptor` stanza of the `weblogic.xml` deployment descriptor file. Note, however, that the application still remains unavailable while in-place redeployment takes place.

Updating Static Files in a Deployed Application

In a production environment, you may occasionally need to refresh the static content of a Web Application module—HTML files, image files, JSPs, and so forth—without redeploying the entire application. If you deployed a Web Application or an Enterprise Application as an exploded archive directory, you can use the `weblogic.Deployer` utility to update one or more changed static files in-place.

To redeploy a single file associated within a deployment unit, specify the file name at the end of the redeploy command. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myApp -redeploy -source myApp/copyright.html
```

Always specify the pathname to updated files relative to the root directory of the exploded archive directory. In the above example, the Web Application is deployed as part of an Enterprise Application, so the module directory is specified (`myApp/copyright.html`).

If the Web Application module had been deployed as a standalone module, rather than as part of an Enterprise Application, the file would have been specified alone (`copyright.html`).

You can also redeploy an entire directory of files by specifying a directory name instead of a single file. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myApp -redeploy myApp/myjsps
```

In the above example, all files and subdirectories located in the `myjsps` subdirectory of the Enterprise Application are redeployed in-place.

Updating the Deployment Configuration for an Application

After you have deployed an application or standalone module, you can change the WebLogic Server deployment configuration using either the Administration Console or `weblogic.Deployer`.

The Administration Console enables you to interactively modify individual deployment configuration properties, while `weblogic.Deployer` only allows you to specify an updated deployment plan file to use for reconfiguring the application.

Modifying a Configuration using the Administration Console

The Administration Console enables you to reconfigure all deployment configuration properties for an application, including properties that were not included in the application's deployment plan. If an application was deployed with a deployment plan, the Console displays any deployment plan configuration properties in the plan in the Deployment Plan tab for the application, as shown in Figure 6-3.

Figure 6-3 Editing Deployment Plan Properties in the Console

More information to come.

The remaining configuration tabs for an application, enable you to change other WebLogic Server configuration properties. The exact properties that are available for configuration depend on the type of application or J2EE module that is deployed. These tabs are available regardless of whether or not the application was deployed with an deployment plan.

Note that certain configuration changes are safe to apply to running production applications, while other changes require you to shutdown and restart the application. See "Understanding Redeployment Behavior for Deployment Configuration Changes" on page 6-21.

How Configuration Changes Are Stored

When you use the Administration Console to make changes to properties that were defined in a deployment plan, the Console generates a new deployment plan that containing variable definitions for the new properties you modified, as well as any existing variables defined in the plan. You can select where to save the new deployment plan.

Updating an Application to Use a Different Deployment Plan

The `weblogic.Deployer` utility enables you to update an application's deployment configuration by providing a new deployment plan to use with the application. Note that the updated deployment plan must be valid for the application's current target servers, or the configuration update will fail.

To reconfigure an application with a different deployment plan, use the **`-update`** command and specify the new deployment plan file, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
     -password weblogic -update -name myTestDeployment
     -plan /myDeployments/myNewPlan.xml
```

Note that certain configuration changes are safe to apply to running production applications, while other changes require you to shutdown and restart the application. See "Understanding Redeployment Behavior for Deployment Configuration Changes" on page 6-21.

Understanding Redeployment Behavior for Deployment Configuration Changes

When you change the deployment configuration for a deployed application, WebLogic Server applies the changes using a redeployment operation. The type of redeployment strategy used depends on the nature of configuration changes applied:

- If you modified the value of a resource binding property, the configuration update uses either the production redeployment strategy (if the application supports production redeployment), or the in-place redeployment strategy for the entire application. Performing in-place redeployment for an entire application or module is not recommended for production environments, because the application is first undeployed, causing connected clients to lose in-progress work. See "Overview of Redeployment Strategies" on page 6-2 for more information.
- If you modified only tuning properties, the WebLogic Server uses the in-place redeployment strategy to partially redeploy only the revised deployment plan. In this case, in-place redeployment is safe for production applications because only the deployment plan is redeployed; the application itself remains available for processing client connections.

The above redeployment behavior applies both to configuration changes made using the Administration Console and via the `weblogic.Deployer` **`-update`** command.

Managing Deployed Applications

The following sections describe how to perform common maintenance tasks on applications and modules that are currently deployed to a WebLogic Server domain:

Taking a Production Application Off-Line

WebLogic Server 9.0 provides two different ways to take an application off-line for testing or maintenance purposes:

- Stopping an Application to Restrict Client Access—This method makes an application unavailable for processing client requests, but does not remove the deployment from the WebLogic Server domain. Stopping an application places the deployment in Administration mode, which allows you to perform internal testing using a configured Administration channel.
- Undeploying an Application or Module—This method makes an application unavailable for processing client requests and removes WebLogic Server-generated deployment files from the domain.

Stopping an Application to Restrict Client Access

As described in "Distributing Applications to a Production Environment" on page 5-17, distributing an application places the deployment in Administration mode, which restricts access to an application to a configured Administration channel. Stopping a running application also places the application in Administration mode. In a production environment, you may want to stop an application to confirm a reported problem, or to isolate the application from external client processing in order to perform scheduled maintenance.

Use the **-stop** command to place a running application into Administration mode, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop
```

By default, WebLogic Server immediately stops the application, without regard to pending HTTP sessions or in-process work. If you want to wait for pending HTTP sessions to complete their work before stopping the application, add the `-graceful` option:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop -graceful
```

To restart an application that was previously stopped, making it available to external clients, use the **-start** command and specify the deployment name. You do not need to redeploy a stopped application to make it generally available:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -start
```

Undeploying an Application or Module

After you deploy a new application or standalone module to servers in a domain, the deployment name remains associated with the deployment files you selected. Even after you stop the deployment on all servers, the files remain available for redeployment using either the Administration Console or `weblogic.Deployer` utility.

If you want to remove a deployment name and its associated deployment files from the domain, you must explicitly undeploy the application or standalone module. To undeploy a deployment unit from the domain using `weblogic.Deployer`, specify the **`-undeploy`** command, as in:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -undeploy
```

By default, WebLogic Server immediately stops and undeploys the application, interrupting current clients and in-progress work. For a production application, you may want to undeploy "gracefully", allowing current HTTP clients to complete their work before undeploying:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -undeploy -graceful
```

Undeploying a deployment unit does not remove the original source files used for deployment. It only removes the deployment's configuration from the domain, as well as any deployment files that WebLogic Server created during deployment (for example, files copied with stage deployment mode and files uploaded to the Administration Server).

If you need to redeploy a deployment unit after undeploying it, you must again identify the deployment files, targets, staging mode, and deployment name using the instructions in "Deploying Applications and Modules" on page 5-1.

Undeploying Shared Libraries and Packages

A shared J2EE libraries or optional package cannot be undeployed until all applications that reference the library or package are first undeployed. If no applications reference an application or package, use the instructions in "Undeploying an Application or Module" on page 7-3 to undeploy it.

Adding a New Module to a Deployed Enterprise Application

WebLogic Server's module-level targeting support enables you to add and deploy a new Enterprise Application module without having to redeploy other modules that are already deployed. To deploy a new module in an EAR, you simply use module-level targeting syntax described in "Module-Targeting Syntax" on page 5-7.

For example, if you were to add a module, `newmodule.war`, to a deployed Enterprise Application named `myapp.ear` (update `application.xml` file as necessary), you could then deploy `newmodule.war` using the `weblogic.Deployer` command:

```
java weblogic.Deployer -username myname -password mypassword
   -name myapp.ear -deploy -targets newmodule.war@myserver
   -source /myapp/myapp.ear
```

This command deploys the new module without redeploying the other modules in the application. Note that you must specify the correct file extension (`.war` in the above example) for archived modules in an EAR file.

Changing the Order of Deployment at Server Startup

By default, WebLogic Server deploys applications and resources in the following order:

1. JDBC system modules
2. JMS system modules
3. J2EE Libraries and optional packages
4. Applications and standalone modules
5. Startup classes

Note: WebLogic Server security services are always initialized before server resources, applications, and startup classes are deployed. For this reason, you cannot configure custom security providers using startup classes, nor can custom security provider implementations rely on deployed server resources such as JDBC.

Changing the Deployment Order for Applications and Standalone Modules

You can change the deployment order for a deployed application or *standalone* module by setting the `ApplicationMBean LoadOrder` attribute in the Administration Console (or programmatically using the `ApplicationMBean`). The `LoadOrder` attribute controls the load order of deployments relative to one another—modules with lower `LoadOrder` values deploy before those with higher values. By default, each deployment unit is configured with a Load Order value of 100. Deployments with the same Load Order value are deployed in alphabetical order using the deployment name. In all cases, applications and standalone modules are deployed after the WebLogic Server instance has initialized dependent subsystems.

Note: You cannot change the load order of applications and standalone modules using the `weblogic.Deployer` utility.

Follow these steps to view or change the deployment order of deployments using the Administration Console:

1. Select the Deployments node in the left pane. The right pane displays all applications and standalone modules configured for deployment in the domain, listed in their current deployment order.
2. Select a deployment and click the Order Deployments button.
3. Enter a new value in the Load Order field, and click Save.

Changing the Deployment Order for Modules in an Enterprise Application

The modules contained in an Enterprise Application are deployed in the order in which they are declared in the `application.xml` deployment descriptor. See Enterprise Application Deployment Descriptor Elements in *Developing WebLogic Server Applications*.

Ordering Startup Class Execution and Deployment

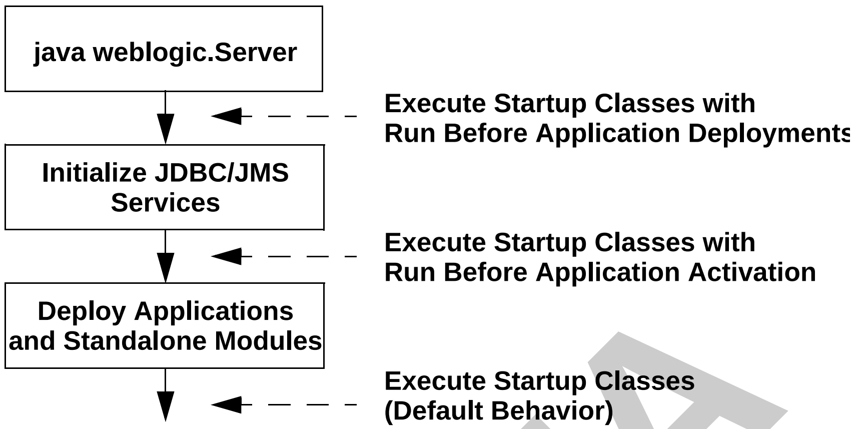
By default WebLogic Server startup classes are run after the server initializes JMS and JDBC services, and after applications and standalone modules have been deployed.

If you want to perform startup tasks after JMS and JDBC services are available, but before applications and modules have been activated, you can select the Run Before Application Deployments option in the Administration Console (or set the `StartupClassMBean`'s `LoadBeforeAppActivation` attribute to "true").

If you want to perform startup tasks before JMS and JDBC services are available, you can select the Run Before Application Activations option in the Administration Console (or set the `StartupClassMBean`'s `LoadBeforeAppDeployments` attribute to "true").

The following figure summarizes the time at which WebLogic Server executes startup classes.

Figure 7-1 Startup Class Execution


java weblogic.Server
Execute Startup Classes with Run Before Application Deployments
Initialize JDBC/JMS Services
Execute Startup Classes with Run Before Application Activation
Deploy Applications and Standalone Modules
Execute Startup Classes (Default Behavior)

See the full Javadocs for `StartupClassMBean` for more information.

Changing the Target List for an Existing Deployment

After you deploy an application or standalone module in a WebLogic Server domain, you can change the target server list to add new WebLogic Server instances or to remove existing server instances. If you remove a target server, only the target list itself is updated—the deployment unit remains deployed to the removed server until you explicitly undeploy it. Similarly, if you add a new target server, you must explicitly deploy the deployment unit on the new server before it is active on that server.

To add a new server to the target list using `weblogic.Deployer`, simply specify the new list of target servers with the **`-deploy`** command. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mydeploymentname -deploy
   -targets server1, newserver
```

Removing Files from a Web Application Deployment

If you deploy a Web Application using an exploded archive directory, you can update static contents of the Web Application either by refreshing the files (see "Updating Static Files in a Deployed Application" on page 6-18), or by deleting files from the deployment. To delete files, you must use the `weblogic.Deployer` utility with the `delete_files` option. For example:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mywebapp -delete_files mywebapp/copyright.html
```

Always specify the pathname to updated files starting with the top level of the exploded archive directory. In the above example, the Web Application resides in an exploded archive directory named `mywebapp`.

Managing Long-Running Deployment Tasks

The WebLogic Server deployment system automatically assigns a unique ID to deployment tasks so that you can track and manage their progress. `weblogic.Deployer` enables you to assign your own task identification numbers for use with deployment commands, as well as monitor and cancel long-running deployment tasks.

For example, the following command assigns a task ID of `redeployPatch2` to a new deployment operation:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -targets myserver -id redeployPatch2
   -nowait -deploy c:\localfiles\myapp.ear
```

You can later check the status of the task using the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -id redeployPatch2 -list
```

You can check the status of all running tasks using the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -listtask
```

If a task takes too long to complete you can cancel it using the command:

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -id redeployPatch2 -cancel
```

weblogic.Deployer Command-Line Reference

`weblogic.Deployer` is a Java-based deployment tool that provides a command-line interface to the WebLogic Server deployment API. `weblogic.Deployer` is intended for administrators and developers who want to perform interactive, command-line based deployment operations.

See the WLST Command and Variable Reference for information about performing deployment operations using the WebLogic Scripting Tool (WLST).

The following sections describe the `weblogic.Deployer` utility:

Required Environment for weblogic.Deployer

To set up your environment to use the `weblogic.Deployer` utility:

1. Install and configure the WebLogic Server software, as described in the WebLogic Server Installation Guide.
2. Add the WebLogic Server classes to the `CLASSPATH` environment variable, and ensure that the correct JDK binaries are available in your `PATH`. You can use the `setWLSEnv.sh` or `setWLSEnv.cmd` script, located in the `server/bin` subdirectory of the WebLogic Server installation directory, to set the environment.
3. If you are connecting to an Administration Server via a configured administration channel, you must also configure SSL on the machine on which you run `weblogic.Deployer`. See See Using the SSL Protocol to Connect to WebLogic Server from weblogic.Admin in *Managing WebLogic Security* for instructions about configuring SSL.

Syntax for Invoking weblogic.Deployer

```
java [SSL Properties] weblogic.Deployer [Connection Arguments]
     [User Credentials Arguments] COMMAND-NAME command-options
     [Common Arguments]
```

Command names and options are not case-sensitive. See "Command Reference" on page 8-8 for detailed syntax and examples of using `weblogic.Deployer` commands.

SSL Properties

If you are deploying an application to an Administration Server via a configured administration channel, you must provide valid SSL properties to the Java runtime. The required SSL properties are the same for both the `weblogic.Deployer` and `weblogic.Admin` utilities. See SSL Arguments in the WebLogic Server Command Reference for more information.

Connection Arguments

```
java [SSL Properties] weblogic.Deployer
     [-adminurl protocol://listen_address:port_number]
     [User Credentials Arguments] COMMAND-NAME command-options [Common
Arguments]
```

Most `weblogic.Deployer` commands require you to specify the `-adminurl` arguments described in Table 8-1 to connect to an Administration Server instance.

Table 8-1 Connection Arguments

Argument	Definition
`-adminurl` `[`*`protocol`*`://]`*`Admin-Server-listen-address`*`:`*`listen-port`*	The `-adminurl` value must specify the listen address and listen port of the Administration Server. To use a port that is not secured by SSL, the format is `-adminurl [`*`protocol`*`]`*`Admin-Server-listen-address`*`:`*`port`* where `t3` and `http` are valid protocols. To use a port that is secured by SSL, the format is `-adminurl` *`secure-protocol`*`://`*`Admin-Server-listen-address`*`:`*`port`* where `t3s` and `https` are valid secure protocols. To connect to the Administration Server via a configured administration channel, you must specify a valid administration port number: `-adminurl` *`secure-protocol`*`://`*`Admin-Server-listen-address`*`:`*`domain-wide-admin-port`* There is no default value for this argument.

User Credentials Arguments

```
java [ SSL Properties ] weblogic.Deployer [Connection Arguments]
     [ { -username username [-password password] } |
     [ -userconfigfile config-file [-userkeyfile admin-key] ] ]
     COMMAND-NAME command-options [Common Arguments]
```

Most `weblogic.Deployer` commands require you to specify the arguments in Table 8-2 to provide the user credentials of a WebLogic Server administrator.

Table 8-2 User Credentials Arguments

Argument	Definition
`-username` *`username`*	The Administrator username. If you supply the `-username` option but you do not supply a corresponding `-password` option, `weblogic.Deployer` prompts you for the password.
`-password` *`password`*	The password of the Administrator user. To avoid having the plain text password appear in scripts or in process utilities such as `ps`, first store the username and encrypted password in a configuration file using the `STOREUSERCONFIG` command with `weblogic.Admin`. Omit both the `-username` and `-password` options to `weblogic.Deployer` to use the values stored in the default configuration file. See STOREUSERCONFIG in the *weblogic.Admin Command-Line Reference* for more information on storing and encrypting passwords. If you want to use a specific configuration file and key file, rather than the default files, use the `-userconfigfile` and `-userkeyfile` options to `weblogic.Deployer`.

Argument	Definition
`-userconfigfile` *`config-file`*	Specifies the location of a user configuration file to use for the administrative username and password. Use this option, instead of the `-user` and `-password` options, in automated scripts or in situations where you do not want to have the password shown on-screen or in process-level utilities such as `ps`. Before specifying the `-userconfigfile` option, you must first generate the file using the `weblogic.Admin` `STOREUSERCONFIG` command as described in STOREUSERCONFIG in the *weblogic.Admin Command-Line Reference*.
`-userkeyfile` *`admin-key`*	Specifies the location of a user key file to use for encrypting and decrypting the username and password information stored in a user configuration file (the `-userconfigfile` option). Before specifying the `-userkeyfile` option, you must first generate the key file using the `weblogic.Admin` `STOREUSERCONFIG` command as described in STOREUSERCONFIG in the *weblogic.Admin Command-Line Reference*.

Common Arguments

The common options described in Table 8-3 can be used with any of the commands described in "Command Reference" on page 8-8.

Table 8-3 Common options for weblogic.Deployer

Option Name	Description
`-advanced`	Prints full command-line help text for all `weblogic.Deployer` actions and options.
`-debug`	Display debug messages in the standard output.
`-examples`	Display example command lines for common tasks.
`-help`	Prints command-line help text for the most commonly-used `weblogic.Deployer` actions and options.
-noexit	By default `weblogic.Deployer` calls `System.exit(1)` if an exception is raised while processing a command. The exit value displayed indicates the number of failures that occurred during the deployment operation. The `-noexit` option overrides this behavior for batch processing.
`-nowait`	`weblogic.Deployer` prints the task ID and exits without waiting for the action to complete. This option is used to initiate multiple tasks and then monitor them later with the `-list` action.
`-output <raw | formatted>`	Specify either `raw` or `formatted` to control the appearance of `weblogic.Deployer` output messages. Both output types contain the same information, but `raw` output does not contain embedded tabs. By default, `weblogic.Deployer` displays `raw` output.
`-remote`	Indicates that `weblogic.Deployer` is not running on the same machine as the Administration Server, and that source paths specified in the command are valid for the Administration Server machine itself. If you do not use the `-remote` option, `weblogic.Deployer` assumes that all source paths are valid paths on the local machine.

Option Name	Description
`-timeout` *`seconds`*	Specifies the maximum time, in seconds, to wait for the deployment task to complete. After the time expires, `weblogic.Deployer` prints out the current status of the deployment and exits.
`-verbose`	Displays additional progress messages, including details about the prepare and activate phases of the deployment.
`-version`	Prints version information for `weblogic.Deployer`.

Command Reference

The following sections describe the `weblogic.Deployer` commands and command options used to perform deployment tasks with WebLogic Server:

- Cancel
- Deploy
- Distribute
- Listapps
- List, Listtask
- Redeploy
- Start
- Stop
- Undeploy
- Update

Note: `weblogic.Deployer` commands are displayed in **bold** type to distinguish them from command options.

Cancel

Attempt to cancel a running deployment task.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -cancel task_id
     [Common Arguments]
```

Argument or Option	Definition
task_id	The identifier of the deployment task to cancel. The identifier can be specified by using the id option with the deploy, distribute, update, undeploy, redeploy, stop, and start commands.

Examples

The following command starts a deployment operation and specifies the task identifier, myDeployment:

```
java weblogic.Deployer -adminurl http://localhost:7001
     -username weblogic -password weblogic
     -deploy ./myapp.ear -id myDeployment
```

If the deployment task has not yet completed, the following command attempts to cancel the deployment operation:

```
java weblogic.Deployer -adminurl http://localhost:7001
     -username weblogic -password weblogic
     -cancel myDeployment
```

Deploy

Deploys or redeploys an application or module.

Note: The `-ACTIVATE` command, an alias for **`-deploy`**, is deprecated.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -deploy [[-name] deployment_name] [-source] file
     [-plan file] [-targets target_list] [-submoduletargets target_list]
     [-upload]
     [-stage | -nostage | -external_stage]
     [-retiretimeout seconds]
     [-library [-libspecver version] [-libimplver version]]
     [-altappdd file] [-altwlsappdd file]
     [-securityModel] [-enableSecurityValidation]
     [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-name` *`deployment_name`*	Specifies the deployment name to assign to a newly-deployed application or standalone module. Both the `-name` option and *`deployment_name`* argument are optional, as described in the Syntax. If a deployment name is not explicitly identified with the **`-deploy`** command, the name is derived from the specified deployment file or directory: • For an archive file, the default deployment name is the name of the archive file without the file extension (`myear` for the file `myear.ear`). • For an exploded archive directory, the default deployment name is the name of the top-level directory. • If you specify an application installation root directory, the default deployment name is derived from the archive filename or exploded archive directory name in the `/app` subdirectory.
`-source` *`file`*	Specifies the archive file or exploded archive directory to deploy. You can omit the `-source` option and supply only the file or directory to deploy.
`-plan` *`file`*	Specifies a deployment plan to use when deploying the application or module. By default, `weblogic.Deployer` does not use an available deployment plan, even if you are deploying from an application root directory that contains a plan.
`-targets` *`target_list`*	Specifies the targets on which to distribute and deploy the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to deploy different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list with the **`-deploy`** command, the target defaults to the Administration Server instance.
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.

Argument or Option	Definition (Continued)
`-upload`	Transfers the specified deployment files, including deployment plans and alternate deployment descriptors, to the Administration Server. Use this option when you are on a remote machine and you cannot copy the deployment files to the Administration Server by other means. The application files are uploaded to the WebLogic Server Administration Server's upload directory prior to distribution and deployment.
`-stage | -nostage | -external_stage`	Specifies a staging mode to use when deploying or distributing an application: • `-stage`—Copies deployment files to target servers' staging directories. `stage` is the default mode used when deploying or distributing to Managed Server targets. • `-nostage`—Does not copy the deployment files to target servers, but leaves them in a fixed location, specified by the `-source` option. Target servers access the same, copy of the deployment files. `nostage` is the default used when deploying or distributing to the Administration Server (for example, in a single-server domain). • `-external_stage`—Does not copy the deployment files to target servers; instead, you must ensure that deployment files have been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the files or use a third-party tool or script. See "Controlling Deployment File Copying with Staging Modes" on page 5-10.
`-retiretimeout` *`seconds`*	Specifies the number of seconds before WebLogic Server undeploys the currently-running version of this application or module. See "Redeploying a New Version of an Application" on page 6-8.
`-library`	Identifies the deployment as a shared J2EE library or optional package. You must include the `-library` option when deploying or distributing any J2EE library or optional package. See "Deploying Shared J2EE Libraries and Dependent Applications" on page 5-18.

Argument or Option	Definition (Continued)
`-libspecver` *`version`*	Specifies the specification version of a J2EE library or optional package. This option can be used only if the library or package does not include a specification version in its manifest file. `-libversion` can be used only in combination with `-library`. See "Registering Libraries with WebLogic Server" on page 5-19.
`-libimplver` *`version`*	Specifies the implementation version of a J2EE library or optional package. This option can be used only if the library or package does not include a implementation version in its manifest file. `-libimplversion` can be used only in combination with `-library`. See "Registering Libraries with WebLogic Server" on page 5-19.
`-altappdd` *`file`*	**(Deprecated.)** Specifies the name of an alternate J2EE deployment descriptor (`application.xml`) to use for deployment.
`-altwlsappdd` *`file`*	**(Deprecated.)** Specifies the name of an alternate WebLogic Server deployment descriptor (`weblogic-application.xml`) to use for deployment.
`-securityModel [ DDOnly | CustomRoles | CustomRolesAndPolicy | Advanced ]`	More information to come.
`-enableSecurityValidation`	More information to come.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-deploy`**, **`-redeploy`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`-deploy`** command:

- "Deploying to a Single-Server Domain" on page 5-2

- "Deploying an Application with a Deployment Plan" on page 5-3
- "Uploading Deployment Files from a Remote Client" on page 5-4
- "Deploying to One or More Targets" on page 5-5
- "Deploying to a Cluster Target" on page 5-6
- "Using Module-Level Targeting for Deploying an Enterprise Application" on page 5-7
- "Targeting Application-Scoped JMS and JDBC Modules" on page 5-8
- "Using Sub-Module Targeting with JMS Application Modules" on page 5-9
- "Using Nostage Mode Deployment" on page 5-13
- "Using Stage Mode Deployment" on page 5-14
- "Using External_stage Mode Deployment" on page 5-15
- "Distributing Applications to a Production Environment" on page 5-17
- "Registering Libraries with WebLogic Server" on page 5-19
- "Deploying Applications that Reference Libraries" on page 5-20

Distribute

Copies and validates deployment files on target servers and places the application in Administration mode. The **`-distribute`** command also registers the deployment name with the Administration Server. While in Administration mode, the application can be accessed by internal clients via a configured Administration port. External clients cannot access the application.

A distributed application can be quickly started by using the Start command.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -distribute [[-name] deployment_name] [-source] file
     [-plan file] [-targets target_list] [-submoduletargets target_list]
     [-upload]
     [-stage | -nostage | -external_stage]
     [-library [-libspecver version] [-libimplver version]]
     [-altappdd file] [-altwlsappdd file]
     [-securityModel] [-enableSecurityValidation]
     [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-name` *`deployment_name`*	Specifies the deployment name to assign to the distributed application or module. Both the `-name` option and *`deployment_name`* argument are optional, as described in the Syntax. If a deployment name is not explicitly identified, a name is derived from the specified deployment file or directory: • For an archive file, the default deployment name is the name of the archive file without the file extension (`myear` for the file `myear.ear`). • For an exploded archive directory, the default deployment name is the name of the top-level directory. • If you specify an application installation root directory, the default deployment name is derived from the archive filename or exploded archive directory name in the `/app` subdirectory.
`-source` *`file`*	Specifies the archive file or exploded archive directory to distribute. You can omit the `-source` option and supply only the file or directory.
`-plan` *`file`*	Specifies a deployment plan to distribute with the application or module, used to configure the application.
`-targets` *`target_list`*	Specifies the targets on which to distribute the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to distribute different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list, the target defaults to the Administration Server instance.
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.

Argument or Option	Definition (Continued)
`-upload`	Transfers the specified deployment files, including any specified deployment plans, to the Administration Server before distribution. Use this option when you are on a remote machine and you cannot copy the deployment files to the Administration Server by other means. The application files are uploaded to the WebLogic Server Administration Server's upload directory prior to distribution.
`-stage | -nostage | -external_stage`	Specifies a staging mode to use when deploying or distributing an application: • `-stage`—Copies deployment files to target servers' staging directories. `stage` is the default mode used when deploying or distributing to Managed Server targets. • `-nostage`—Does not copy the deployment files to target servers, but leaves them in a fixed location, specified by the `-source` option. Target servers access the same, copy of the deployment files. `nostage` is the default used when deploying or distributing to the Administration Server (for example, in a single-server domain). • `-external_stage`—Does not copy the deployment files to target servers; instead, you must ensure that deployment files have been copied to the correct subdirectory in the target servers' staging directories. You can manually copy the files or use a third-party tool or script. See "Controlling Deployment File Copying with Staging Modes" on page 5-10.
`-library`	Identifies the deployment as a shared J2EE library or optional package. You must include the `-library` option when deploying or distributing any J2EE library or optional package. See "Deploying Shared J2EE Libraries and Dependent Applications" on page 5-18.
`-libspecver` *`version`*	Specifies the specification version of a J2EE library or optional package. This option can be used only if the library or package does not include a specification version in its manifest file. `-libversion` can be used only in combination with `-library`. See "Registering Libraries with WebLogic Server" on page 5-19.

Argument or Option	Definition (Continued)
`-libimplver` *`version`*	Specifies the implementation version of a J2EE library or optional package. This option can be used only if the library or package does not include a implementation version in its manifest file. `-libimplversion` can be used only in combination with `-library`. See "Registering Libraries with WebLogic Server" on page 5-19.
`-altappdd` *`file`*	**(Deprecated.)** Specifies the name of an alternate J2EE deployment descriptor (`application.xml`) to use for deployment or distribution.
`-altwlsappdd` *`file`*	**(Deprecated.)** Specifies the name of an alternate WebLogic Server deployment descriptor (`weblogic-application.xml`) to use for deployment or distribution.
`-securityModel [ DDOnly | CustomRoles | CustomRolesAndPolicy | Advanced ]`	More information to come.
`-enableSecurityValidation`	More information to come.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-deploy`**, **`-redeploy`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

The **`-distribute`** command operates similar to **`-deploy`**, but WebLogic Server does not start the application or module on target servers. See the examples links for the `Deploy` command for more information.

Listapps

Lists the deployment names for applications and standalone modules deployed, distributed, or installed to the domain.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -listapps
     [Common Arguments]
```

Examples

See "Displaying Version Information for Deployed Applications" on page 6-8.

List, Listtask

Displays the status of deployment tasks currently running in the domain.

Syntax

```
java [SSL Properties] weblogic.Deployer Connection Arguments
    [User Credentials Arguments] <-list | -listtask> [task_id]
    [Common Arguments]
```

Argument or Option	Definition
task_id	The identifier of a deployment task to display. The identifier can be specified by using the `-id` argument to the DEPLOY, DISTRIBUTE, UPDATE, UNDEPLOY, REDEPLOY, STOP, and START commands.

Examples

See "Managing Long-Running Deployment Tasks" on page 7-7.

Redeploy

Redeploys a running application or part of a running application.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -redeploy [[-name] deployment_name] {-source file | filelist}
     [-plan file] [-targets target_list] [-submoduletargets target_list]
     [-upload]
     [-delete_files]
     [-retiretimeout seconds] [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-name` *deployment_name*	Specifies the deployment name of a deployed application or module. The `-name` option can be omitted, in which case the name is taken from the `-source` *file* argument.
`-source` *file*	Specifies the archive file or exploded archive directory to distribute, deploy, or redeploy. When used with the `REDEPLOY` command, the `-source` option specifies the location of new deployment files to redeploy, for example, when updating an application to a new version. To specify multiple files for a partial redeployment, omit the `-source` option and supply only a *filelist*. **Note:** Use a *file* or *filelist* specification only for redeploying static files *within* a J2EE module. To redeploy an entire J2EE module within an Enterprise Application, use the module-targeting syntax, `-targets` *module@target*, described in "Using Partial Redeployment for J2EE Module Updates" on page 6-17.

Argument or Option	Definition (Continued)
`filelist`	Specifies one or more files to redeploy. If the *`filelist`* specifies multiple files, the redeployment is treated as a partial redeployment of the specified files. **Note:** Use a *`file`* or *`filelist`* specification only for redeploying static files *within* a J2EE module. To redeploy an entire J2EE module within an Enterprise Application, use the module-targeting syntax, `-targets` *`module@target`*, described in "Using Partial Redeployment for J2EE Module Updates" on page 6-17. The use of `-redeploy` *`module-uri`* is deprecated. Instead, use production redeployment or redeploy the module using the `-targets` *`module@target`* syntax.
`-plan` *`file`*	Specifies a deployment plan to use when distributing, deploying, or redeploying. When redeploying an application, the `-plan` option allows you to specify an updated configuration to use during the redeployment. If the revised deployment plan contains changes to resource bindings, WebLogic Server attempts to redeploy a new version of the application alongside an older version. See "Updating the Deployment Configuration for an Application" on page 6-19.
`-targets` *`target_list`*	Specifies the targets on which to distribute, deploy, or redeploy the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to redeploy different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list with the **`-deploy`** command, the target defaults to the Administration Server instance. If you do not specify a target list with the **`-redeploy`** command, the application is redeployed on all of its current target servers.
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.

Argument or Option	Definition (Continued)
`-upload`	Transfers the specified deployment files, including deployment plans and alternate deployment descriptors, to the Administration Server. Use this option when you are on a remote machine and you cannot copy the deployment files to the Administration Server by other means. The application files are uploaded to the WebLogic Server Administration Server's upload directory prior to distribution and deployment. Use the `-upload` option with the `REDEPLOY` command when you are upgrading an application to a new version.
`-delete_files`	Removes static files from a server's staging directory. `delete_files` is valid only for unarchived deployments, and only for applications deployed using `-stage` mode. You must specify target servers when using this option, as shown in the following example: `java weblogic.Deployer -adminurl http://myserver:7001 -username weblogic -password weblogic -name myapp -targets myapp@myserver -redeploy -delete_files myapp/tempindex.html` `delete_files` only removes files that WebLogic Server copied to the staging area during deployment. If you use the `delete_files` option with an application that was deployed using either `-nostage` or `-external_stage` mode, the command does not delete the files. `delete_files` can only be used in combination with the **`-redeploy`** command.
`-retiretimeout` *`seconds`*	Specifies the number of seconds before WebLogic Server undeploys the currently-running version of this application or module. See "Redeploying a New Version of an Application" on page 6-8.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-deploy`**, **`-redeploy`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`-redeploy`** command:

- “Redeploying a New Version of an Application” on page 6-8
- “Rolling Back the Production Redeployment Process” on page 6-10
- “Steps for Distributing a New Version of an Application” on page 6-12
- “Redeploying Applications and Modules In-Place” on page 6-15
- “Using Partial Redeployment for J2EE Module Updates” on page 6-17
- “Updating Static Files in a Deployed Application” on page 6-18

Start

Makes a stopped (inactive) application available to clients on target servers. **`-start`** does not redistribute deployment files to target servers; the files must already be available via an earlier **`-deploy`** or **`-distribute`** command.

Note: The `-activate` command, an alias for **`-start`**, is deprecated.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -start [-name] deployment_name
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
     [-retiretimeout seconds]
     [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-name` *`deployment_name`*	Specifies the deployment name of a deployed application or module. The `-name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-appversion` *`version`*	The version of the application to start.
`-planversion` *`version`*	The version of the deployment plan to use when starting the application.

Argument or Option	Definition (Continued)
`-targets` *`target_list`*	Specifies the targets on which to `DISTRIBUTE`, `DEPLOY`, `REDEPLOY`, or `START` the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to deploy different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list with the **`-deploy`** command, the target defaults to the Administration Server instance. If you do not specify a target list with the **`-redeploy`** or **`-start`** commands, the command is performed on all of the application's current targets.
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.
`-retiretimeout` *`seconds`*	Specifies the number of seconds before WebLogic Server undeploys the currently-running version of this application or module. See "Redeploying a New Version of an Application" on page 6-8.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-distribute`**, `DEPLOY`, **`-redeploy`**, **`-start`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`-start`** command:

- "Starting a Distributed Application" on page 5-17
- "Making a Distributed Application Available to Clients" on page 6-12
- "Stopping an Application to Restrict Client Access" on page 7-2

Stop

Makes an application inactive and unavailable to clients on target servers by placing it in Administration mode. All of the application's staged files remain available on target servers for subsequent **`-start`**, **`-deploy`**, **`-redeploy`**, or **`-undeploy`** actions. While stopped, an application can still be accessed only via a configured Administration channel.

Note: The **`-deactivate`** command, an alias for **`-stop`**, is deprecated.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -stop [-name] deployment_name
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
     [-ignoresessions] [-graceful]
     [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-name` *`deployment_name`*	Specifies the deployment name of a deployed application or module. The `-name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-appversion` *`version`*	The version identifier of the deployed application.
`-planversion` *`version`*	The version identifier of the deployment plan.

Argument or Option	Definition (Continued)
`-targets` *`target_list`*	Specifies the targets on which to **`-distribute`**, **`-deploy`**, **`-redeploy`**, **`-start`**, or **`-stop`** the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to deploy different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list with the **`-deploy`** command, the target defaults to the Administration Server instance. If you do not specify a target list with the **`-redeploy`**, **`-start`**, or **`-stop`** commands, the command is performed on all of the application's current targets.
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.
-graceful	Stops the application after existing HTTP clients have completed their work. If you do not specify the `-graceful` option, WebLogic Server immediately stops the application or module. See "Taking a Production Application Off-Line" on page 7-2.
`-ignoresessions`	This option immediately places the application into Administration mode without waiting for current HTTP sessions to complete.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-distribute`**, `DEPLOY`, **`-redeploy`**, **`-start`**, **`-stop`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`-stop`** command, see "Stopping an Application to Restrict Client Access" on page 7-2.

Undeploy

Stops the deployment unit and removes staged files from target servers.

Note: The -REMOVE command, an alias for **-undeploy**, is deprecated.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -undeploy [-name] deployment_name
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
     [-graceful] [-ignoresessions]
     [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
-name *deployment_name*	Specifies the deployment name of a deployed application or module. The -name option can be omitted, in which case the name is taken directly from the deployment_name. (If the deployment_name specifies a file or directory name, the deployment name is derived from the file specification.)
-appversion *version*	The version identifier of the deployed application.
-planversion *version*	The version identifier of the deployment plan.
-targets *target_list*	Specifies the targets on which to **-distribute**, **-deploy**, **-redeploy**, **-undeploy**, **-start**, or **-stop** the application or module. The *target_list* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to deploy different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list with the **-deploy** command, the target defaults to the Administration Server instance. If you do not specify a target list with the **-redeploy**, **-undeploy**, **-start**, or **-stop** commands, the command is performed on all of the application's current targets.

Argument or Option	Definition (Continued)
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.
`-graceful`	Stops the application after existing HTTP clients have completed their work. If you do not specify the `-graceful` option, WebLogic Server immediately stops the application or module. See "Taking a Production Application Off-Line" on page 7-2. The module is undeployed after it is stopped.
`-ignoresessions`	Immediately stops and undeploys the application without waiting for current HTTP sessions to complete.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-distribute`**, `DEPLOY`, **`-redeploy`**, **`-start`**, **`-stop`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See the following sections for examples of using the **`-undeploy`** command:

- "Redeploying a New Version of an Application" on page 6-8
- "Undeploying a Retiring Application" on page 6-10
- "Undeploying an Application or Module" on page 7-3
- "Undeploying Shared Libraries and Packages" on page 7-3

Update

Updates an application’s deployment plan by redistributing the plan files and reconfiguring the application based on the new plan contents. Note that **`-update`** cannot be used to update an application’s resource bindings. To update the resource bindings for an application, you must use the Redeploy command.

Syntax

```
java [SSL Properties] weblogic.Deployer
     Connection Arguments [User Credentials Arguments]
     -update -plan deployment_plan [-name] deployment_name
     [-appversion version] [-planversion version]
     [-targets target_list] [-submoduletargets target_list]
     [-upload] [-id task_id]
     [Common Arguments]
```

Argument or Option	Definition
`-plan` *`deployment_plan`*	Identifies the deployment plan to use for updating the application’s configuration. The specified deployment plan must be valid for the application’s target servers. For example, the plan cannot contain null variables for required resources.
`-name` *`deployment_name`*	Specifies the deployment name of a deployed application or module. The `-name` option can be omitted, in which case the name is taken directly from the `deployment_name`. (If the `deployment_name` specifies a file or directory name, the deployment name is derived from the file specification.)
`-appversion` *`version`*	The version identifier of the deployed application.
`-planversion` *`version`*	The version identifier of the deployment plan.

Argument or Option	Definition (Continued)
`-targets` *`target_list`*	Specifies the targets on which to **`-distribute`**, **`-deploy`**, **`-redeploy`**, **`-undeploy`**, **`-start`**, or **`-stop`** the application or module. The *`target_list`* argument is a comma-separated list of the target servers, clusters, or virtual hosts. Each target may be qualified with a J2EE module name (<module1>@<server1>). This enables you to deploy different modules of an Enterprise Application to different servers or clusters. If you do not specify a target list with the **`-deploy`** command, the target defaults to the Administration Server instance. If you do not specify a target list with the **`-redeploy`**, **`-undeploy`**, **`-start`**, or **`-stop`** commands, the command is performed on all of the application's current targets.
`-submoduletargets` *`target_list`*	Specifies JMS Server targets for resources defined within a JMS application module. See "Using Sub-Module Targeting with JMS Application Modules" on page 5-9.
`-upload`	Uploads a new deployment plan to the Administration Server before updating the application.
`-id` *`task_id`*	Specifies the task identifier of a running deployment task. You can specify an identifier with the **`-distribute`**, `DEPLOY`, **`-redeploy`**, **`-update`**, **`-start`**, **`-stop`**, or **`-undeploy`** commands, and use it later as an argument to the **`-cancel`** or **`-list`** commands. Make sure that the identifier is unique to all other running deployment tasks. The system automatically generates a unique identifier if you do not specify one.

Examples

See "Updating an Application to Use a Different Deployment Plan" on page 6-20

CHAPTER 9

Deployment Plan Reference and Schema

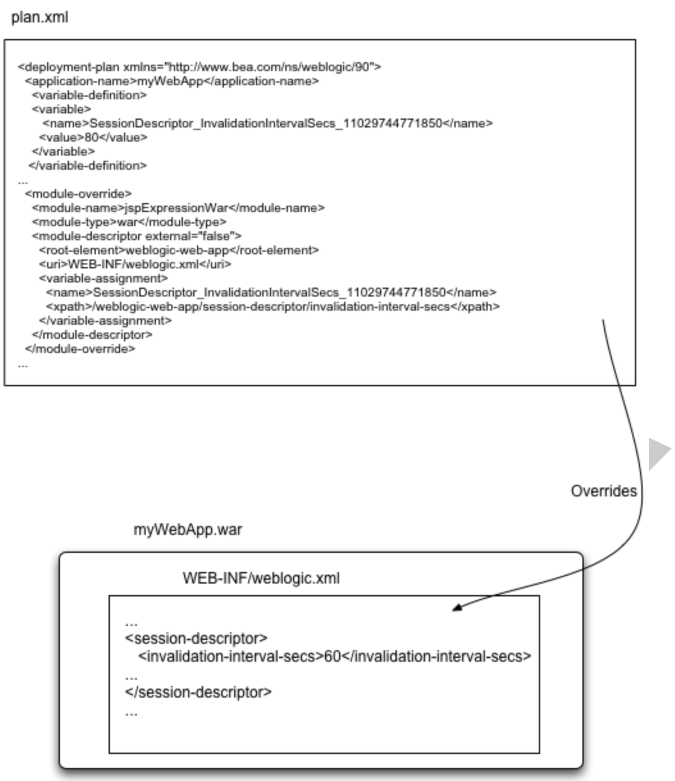
This Appendix provides additional information about WebLogic Server deployment plans. It includes the following sections:

How Deployment Plans Work

A WebLogic Server deployment plan is an optional XML file that configures an application for deployment to WebLogic Server. A deployment plan works by setting property values that would normally be defined in the WebLogic Server deployment descriptors, or by overriding property values already defined in a WebLogic Server deployment descriptor. When exporting an application, the deployment plan typically acts to override selected properties in the WebLogic Server deployment descriptors you created during development.

Deployment plans help an Administrator easily modify an application's WebLogic Server configuration for deployment into to multiple, differing WebLogic Server environments *without* modifying the packaged WebLogic Server deployment descriptor files. Configuration changes are applied by adding or changing variables in the deployment plan, which define both the location of the WebLogic Server descriptor properties to change and the value to assign to those properties. Administrators deploying an application need only change the deployment plan—the original deployment files and deployment descriptors remain unchanged.

Figure 9-1 WebLogic Server Deployment Plan



```
<deployment-plan xmlns="http://www.bea.com/ns/weblogic/90">
  <application-name>myWebApp</application-name>
   <variable-definition>
   <variable>
      <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
     <value>80</value>
   </variable>
  </variable-definition>
...
  <module-override>
   <module-name>jspExpressionWar</module-name>
   <module-type>war</module-type>
   <module-descriptor external="false">
     <root-element>weblogic-web-app</root-element>
     <uri>WEB-INF/weblogic.xml</uri>
     <variable-assignment>
       <name>SessionDescriptor_InvalidationIntervalSecs_11029744771850</name>
       <xpath>/weblogic-web-app/session-descriptor/invalidation-interval-secs</xpath>
     </variable-assignment>
   </module-descriptor>
  </module-override>
...
```

```
...
<session-descriptor>
   <invalidation-interval-secs>60</invalidation-interval-secs>
...
</session-descriptor>
...
```

Deployment Plan Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
     targetNamespace="http://www.bea.com/ns/weblogic/90"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:wls="http://www.bea.com/ns/weblogic/90"
     xmlns:j2ee="http://java.sun.com/xml/ns/j2ee"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified"
     version="1.0">
  <xsd:import namespace="http://java.sun.com/xml/ns/j2ee"
             schemaLocation="http://java.sun.com/xml/ns/j2ee/j2ee_1_4.xsd" />
<xsd:element name="deployment-plan" type="wls:deployment-planType"/>
<xsd:complexType name="deployment-planType">
    <xsd:sequence>
      <xsd:element name="description"
                   type="xsd:string"
                   minOccurs="0"/>
      <xsd:element name="application-name"
                   type="xsd:string"/>
      <xsd:element name="version"
                   type="xsd:string"
                   minOccurs="0"/>
      <xsd:element name="variable-definition"
                   type="wls:variable-definitionType"
                   minOccurs="0"/>
      <xsd:element name="module-override"
```

```
                    type="wls:module-overrideType"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
      <xsd:element name="config-root"
                    type="xsd:string"
                    minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="global-variables"
                    type="xsd:boolean"
                    use="optional"
                    default="false"/>
</xsd:complexType>
<xsd:complexType name="variable-definitionType">
  <xsd:sequence>
     <xsd:element name="variable"
                   type="wls:variableType"
                   minOccurs="0"
                   maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<!--
A single variable definition
-->
<xsd:complexType name="variableType">
  <xsd:sequence>
    <xsd:element name="name"
                  type="xsd:string"/>
    <xsd:element name="value"
                 type="xsd:string"
                 minOccurs="0"
  nillable="true"/>
    <xsd:element name="description"
                 type="xsd:string"
                 minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<!--
Each variable assignment has the following elements:
name: identifies the variable.
xpath: valid xpaths into the descriptor identified by the uri. The xpaths may
resolve to multiple elements.
description: an optional description.
-->
<xsd:complexType name="variable-assignmentType">
  <xsd:sequence>
     <xsd:element name="name"
                  type="xsd:string"/>
     <xsd:element name="xpath"
                  type="j2ee:pathType"/>
     <xsd:element name="description"
                  type="xsd:string"
                  minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="module-overrideType">
```

```
  <xsd:sequence>
    <xsd:element name="module-name"
                 type="xsd:string"/>
    <xsd:element name="module-type"
                 type="xsd:string"/>
    <xsd:element name="module-descriptor"
                 type="wls:module-descriptorType"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="module-descriptorType">
  <xsd:sequence>
    <xsd:element name="root-element"
                 type="xsd:string"/>
    <xsd:element name="uri"
                 type="j2ee:pathType"/>
    <xsd:element name="variable-assignment"
                 type="wls:variable-assignmentType"
                 minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:element name="hash-code"
                 type="xsd:string"
                 minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="external"
```

```
                 type="xsd:boolean"
                 use="optional"
                 default="false"/>
</xsd:complexType>
</xsd:schema>
```

Understanding the Deployment Configuration Process

Deployment Configuration for an application can occur at several points in the lifecycle of an application. Each phase of deployment configuration typically involves creating and working with different deployment files:

- Development Configuration—During development, a programmer creates J2EE deployment descriptors for an application or module. The programmer also creates WebLogic Server deployment descriptors to configure the application for deployment to a Weblogic Server development environment.

  Note: Applications developed outside of the WebLogic Server development environment (for example, a sample or third-party J2EE application such as PetStore) may include only J2EE descriptors.

- Export Configuration—Before releasing an application from development, a programmer or designer may optionally export the application's deployment configuration to a WebLogic Server deployment plan. Exporting a configuration creates deployment plan variables for all or a subset of the deployment properties already defined in the application's WebLogic Server descriptor files.

  Exporting an application helps deployers in other areas of the organization (such as engineers on the QA team or production Administrators) easily deploy the application to environments that differ from the programmer's development environment. The ideal deployment plan provides a complete list of properties that a deployer will likely need to change before deploying the application in a new environment.

- Deployment-time Configuration—An Administrator or deployer configures the application before deploying the application into their target environment. Deployment-time configuration may use the same Weblogic Server deployment configuration files created during development, modified versions of the development configuration files, or custom configuration files that the deployer previously created for their environment, depending on the deployment configuration workflow for your organization.

- Post-Deployment Configuration—After an application has been deployed to a target environment, an Administrator or deployer can reconfigure the application by redeploying a new deployment plan or by using the Administration Console to update an redeploy an existing deployment plan.

Because deployment configuration is performed by different people at different points in the lifecycle of an application, both administrators and developers should work together to create a repeatable configuration workflow for their organization. See "Typical Deployment Configuration Workflows" on page 9-10 for more information.

Administrators and deployers typically perform deployment configuration only at deployment time or after an application has been deployed. At these stages, the additional configuration required for an application depends on the available configuration files. See "Updating the Deployment Configuration for an Application" on page 6-19.

Typical Deployment Configuration Workflows

The introduction of deployment plans in WebLogic Server 9.0 makes it possible to define a convenient, repeatable workflow for configuring an application for deployment to multiple WebLogic Server environments. A configuration workflow for production applications requires cooperation between your development and design team, who creates and packages the deployable application, and the administrator or deployer for each target WebLogic Server environment.

The ideal deployment configuration workflow for your organization will be determined by:

- Number of different environments in which you deploy the same application
- Difference in resources provided by each target environment
- Frequency in which each target environment changes
- Frequency in which the application's J2EE configuration changes
- Ownership requirements for configuration information in different areas of your organization

The sections that follow describe common deployment configuration workflows for managing deployment plans and deploying applications to multiple WebLogic Server domains.

Application with Single Deployment Plan

Organizations that know the exact configuration of different deployment environments can use a single, well-defined deployment plan to deploy an application to multiple WebLogic Server domains. The single deployment plan configuration workflow works in the following way:

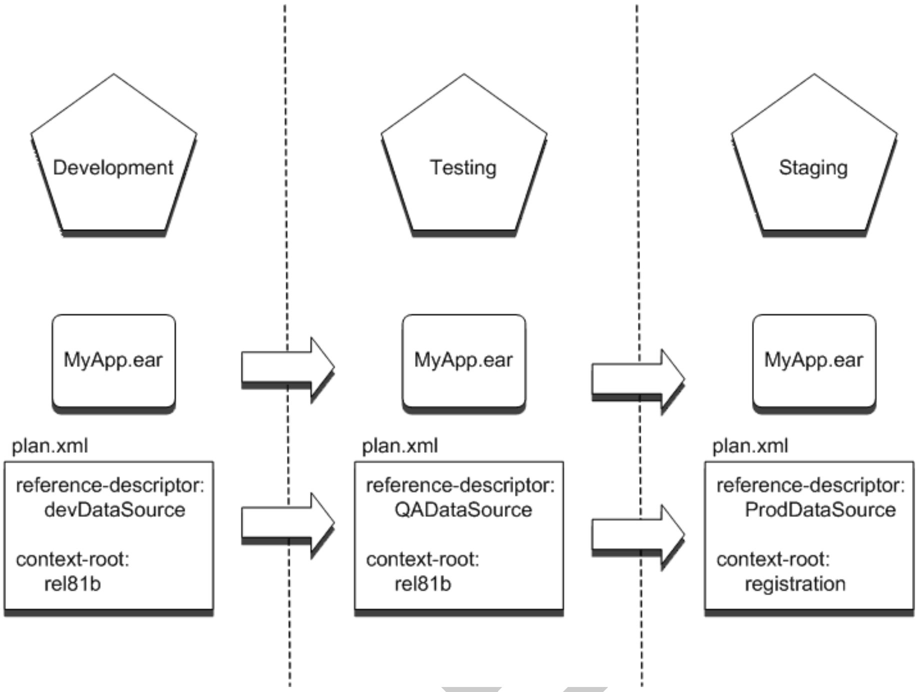
1. The development team, cooperating with administrators and deployers, creates a master deployment plan for use with all target environments. The number of target environments will vary depending on your organizational structure. Common deployment environments include one or more Quality Assurance (QA) or testing domains, staging domains, and production domains.

The deployment plan created at this phase defines variables for all configuration properties that are known to differ between each target environment. For example, the plan might define empty variables for resource names that differ between environments and must be configured before the application can be deployed. The plan may also define default values for common tuning parameters that deployers may want to change in their environments.

For more information about creating a deployment plan during development, see Exporting an Application for Deployment to New Environments in *Developing WebLogic Server Applications*.

2. When a version of the application is released, the development team packages the application deployment files and delivers both the deployment files and a master deployment plan to deployers for each target environment.
3. Each deployer uses the Administration Console to install the application and identify the deployment plan to use for configuration. The Administration Console validates the overall deployment configuration based on the resources available in the target domain. The Console then presents a list of configurable properties defined in the plan (as well as any invalid properties) to the deployer for editing.

Figure 9-2 Single Deployment Plan Workflow


Development
Testing
Staging
MyApp.ear
MyApp.ear
MyApp.ear
plan.xml
reference-descriptor: devDataSource
context-root: rel81b
plan.xml
reference-descriptor: QADataSource
context-root: rel81b
plan.xml
reference-descriptor: ProdDataSource
context-root: registration

4. The deployer uses the Administration Console to interactively configure properties that were defined in the deployment plan. Deployment plan variables that have null values, or invalid values for the target WebLogic Server instances or clusters, must be configured before the application can be deployed. Deployment plan variables that already have valid values need not be changed before deployment.

   Deployers in each environment agree to limit their configuration changes to those properties defined in the deployment plan. If additional configuration changes are required, the deployer must communicate those requirements to the development or design team who modify the master deployment plan.

Using the single deployment plan workflow provides the following benefits:

- It enables the development or design team to control the deployment configuration of the application.
- It reduces the number of configuration decisions that a deployer must make when deploying the application to a target environment.

- It minimizes the number of configuration files associated with the application, making it easy to store deployment configuration information in a source control system.

In general, you would use a single deployment plan workflow if your organization has a few, well-understood target environments, and you want to easily replicate a standardized deployment configuration in each environment.

Deployment Plan Ownership

The single deployment plan workflow assumes that the development or design team maintains ownership of the deployment plan, and that deployers limit their plan changes to those variables defined in the plan. If the deployer modifies only those properties defined in the deployment plan, their changes are written back to the same deployment plan as updates to the variables.

However, WebLogic Server imposes no restrictions on the configuration properties that a deployer can modify using the Administration Console. If a deployer configures deployment properties that were not originally defined in a plan, the Console generates a new deployment plan and with additional variable entries, and uses the new plan for deployment or redeployment operations. This can lead to a situation where the deployer uses a deployment plan that is drastically different from the master deployment plan owned by the development team.

To incorporate new changes into the master deployment plan, the deployer can retrieve their customized deployment plan created by the Console. Ideally, those changes should be applied to the master deployment plan.

Application with Multiple Deployment Plans

Organizations that have numerous deployment environments that frequently change should use a configuration workflow with multiple deployment plans. In a multiple deployment plan workflow, each deployment plan is owned by the deployer of the application rather than the development team. The multiple deployment plan configuration workflow works in the following way:

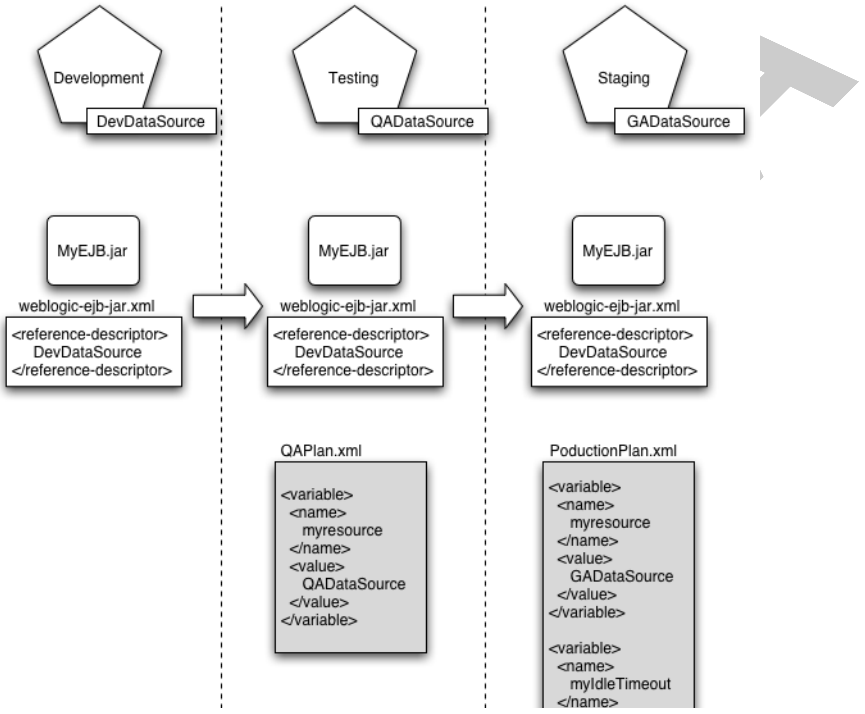
1. The development team releases a version of the packaged application deployment files (containing J2EE and Weblogic Server descriptors). The development team may or may not include a basic deployment plan with exported variables for resource definitions or common tunable parameters.

2. When first deploying the application, each deployer generates a custom deployment plan to configure the application for their target environment.

A custom deployment plan can be created by starting with the basic deployment plan (or no deployment plan) and making interactive changes to the application's deployment configuration using the Administration Console. Changes made using the Console are either written back to the original deployment plan, or are stored in a newly-generated deployment plan for the application.

3. After defining the deployment configuration for their environment, each deployer retrieves their custom deployment plan and maintains it for future deployments of the application. BEA recommends storing custom configuration plans in a source control system so that new versions can be tracked and reverted to if necessary.

Figure 9-3 Multiple Deployment Plan Workflow


Development
DevDataSource
Testing
QADataSource
Staging
GADataSource
MyEJB.jar
weblogic-ejb-jar.xml
<reference-descriptor> DevDataSource </reference-descriptor>
MyEJB.jar
weblogic-ejb-jar.xml
<reference-descriptor> DevDataSource </reference-descriptor>
MyEJB.jar
weblogic-ejb-jar.xml
<reference-descriptor> DevDataSource </reference-descriptor>
QAPlan.xml
<variable> <name> myresource </name> <value> QADataSource </value> </variable>
PoductionPlan.xml
<variable> <name> myresource </name> <value> GADataSource </value> </variable>
<variable> <name> myIdleTimeout </name>

4. For subsequent releases of the application, each deployer uses their customized deployment plan to configure the application for deployment. Using the customized plan allows deployers to perform non-interactive deployments with the `weblogic.Deployer` or fully automated deployments using WLST.

Using the multiple deployment plan workflow provides the following benefits:

- It enables the administrator or deployer to manage both the application configuration and environment configuration in tandem.
- It enables deployers to automate the deployment process by using a custom plan that fully configures the application for their application.

In general, you would use a multiple deployment plan workflow if your organization has many deployment environments that change frequently, making it difficult or impossible to maintain a single master deployment plan.

Deployment Plan Ownership

The multiple deployment plan workflow assumes that the deployer or administrator (rather than the programming or design team) owns and maintains the deployment configuration for an application. It also assumes that the basic J2EE configuration of the application rarely changes, because certain J2EE configuration changes would render a deployer's custom configuration plans invalid. For example, if a module in an Enterprise Application is added, removed, or changed, custom deployment plans referencing the module would become invalid. In this case, each deployer would need to re-create their custom plan by interactively configuring the application using the Administration Console.