Oracle® Fusion Middleware

Upgrade Guide for Oracle WebLogic Server

12*c* Release 1 (12.1.1)

E24497-03

November 2013

This document describes the procedures to upgrade an application environment to Oracle WebLogic Server 12*c* Release 1 (12.1.1).

ORACLE

Oracle Fusion Middleware Upgrade Guide for Oracle WebLogic Server, 12*c* Release 1 (12.1.1)

E24497-03

Copyright © 2007, 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

A WebLogic Server 12.1.1 Compatibility with Previous Releases

Preface

This preface describes the document accessibility features and conventions used in this guide—*Upgrade Guide for Oracle WebLogic Server*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction

This document describes the process of moving an existing application, unchanged, to WebLogic Server 12*c* Release 1 (12.1.1). Although you may decide to change your application when upgrading to WebLogic Server 12.1.1, and although in some cases, you must change your application, this document focuses on issues that you should consider when moving an application to WebLogic Server 12.1.1 without making application changes.

WebLogic Server generally supports very high levels of upgrade capability across WebLogic Server versions. This document is intended to provide WebLogic Server upgrade support and identify issues that may surface during an upgrade so that they can be easily resolved.

> **Note:** For information about upgrading your Java EE environment and your deployed applications from Oracle Application Server 10*g* and Oracle Containers for Java EE (OC4J) to WebLogic Server 12*c* Release 1 (12.1.1), see *Oracle Fusion Middleware Upgrade Guide for Java EE*.
>
> Oracle does not require WebLogic domains to be upgraded from WebLogic Server 10.3 to 12.1.1. WebLogic domains based upon WebLogic Server 10.3 run on WebLogic Server 12.1.1 without modification.

WebLogic Server 12.1.1 includes powerful tools to assist you with upgrading your application environments, including the WebLogic Upgrade Wizard for upgrading domains, custom security providers, and custom node managers.

Most WebLogic Server applications can be run without modifications in the new WebLogic Server 12.1.1 application environment.

The following topics are discussed in this chapter:

1.1 Important Terminology

We recommend that, before proceeding, you familiarize yourself with the following terminology:

- **Upgrade**—In this document, the term upgrade refers to the process of moving an existing application, unchanged, to a new (upgraded) WebLogic Server version.
- **Application Environment**—An application environment includes applications and the WebLogic domains in which they are deployed. It also includes any application data associated with the domain, and may include resources such as database servers, firewalls, load balancers, and LDAP servers.
- **Migrate**—To move an application or domain configuration from a third-party product to an Oracle product.
- **Interoperability**—(1) The ability of an application deployed in one WebLogic Server version to communicate with another application that is deployed in a different WebLogic Server version. (2) The ability of Oracle product components to communicate with third-party software using standard protocols.
- **Compatibility**—The capability of an application built using one WebLogic Server release to run in another another WebLogic Server release, regardless of whether the application was rebuilt.

1.2 Overview of the Upgrade Process

The process required to upgrade an application environment depends on the scope of the application. An *application environment* includes a WebLogic domain and any applications and application data associated with the domain. It may also include external resources, such as firewalls, load balancers, and LDAP servers. Figure 1–1 shows an example of a WebLogic application environment.

Figure 1–1 Example WebLogic Application Environment

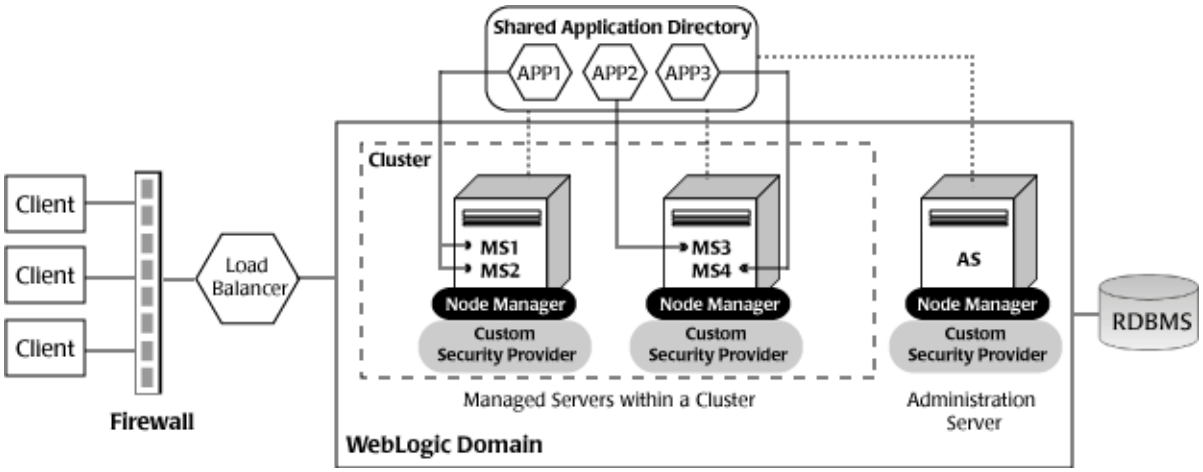


Table 1–1 lists the components of the WebLogic application environment shown in Figure 1–1 and the upgrade requirements for each.

Table 1–1 Upgrade Requirements for Components in Example WebLogic Application Environment

Component	Description	Upgrade Requirements
WebLogic domain	Includes the Administration Server (AS) and optionally one or more Managed Servers (for example, MS1, MS2, MS3, and MS4). The servers in a domain may span multiple machines. Furthermore, you can group Managed Servers into clusters to support load balancing and failover protection for critical applications. For more information about WebLogic domains, see "Understanding Oracle WebLogic domains" in *Understanding Domain Configuration for Oracle WebLogic Server*.	Upgrade the domain directory on each computer in the domain.
Custom security provider	Supports custom security requirements. For information about developing custom security providers, see *Developing Security Providers for Oracle WebLogic Server*.	Upgrade the custom security providers on each computer in the domain.
Node Manager	Provides high availability to Managed Servers. For more information about Node Manager, see "Node Manager Overview" in *Node Manager Administrator's Guide for Oracle WebLogic Server*.	Upgrade custom Node Manager on each computer in the domain.
Applications	Any Java EE applications, including Web applications, EJBs, and so on. Typically, applications are deployed to one or more Managed Servers in a domain. Depending on the deployment strategy, applications may reside locally on a computer or be accessible using a shared directory. In addition, external client applications may access the application environment from outside a firewall.	Most WebLogic Server applications can be run without modifications in the new WebLogic Server 12.1.1 application environment. For more information, see Section 1.5, "Interoperability and Compatibility with Previous Releases."
External resources	Software components, such as databases for storing domain and application data, load balancers, and firewalls.	Verify that all external resources are compatible with WebLogic Server 12.1.1. For more information, see the Oracle Fusion Middleware Supported System Configurations page.

Upgrading business applications that are deployed to WebLogic Server may involve upgrading multiple WebLogic Server applications, and in some cases domains, in a coordinated fashion in order to:

- maintain consistency in the WebLogic Server versions being used
- use the same supported configurations environment across the entire installation
- meet specific interoperability requirements.

For example, you may want to upgrade all applications and domains simultaneously, upgrade them in a well-defined sequence, or upgrade some applications and domains while leaving other applications and domains on older WebLogic Server versions.

1.3 Before You Begin

Before you begin the upgrade process, you should consider the scope of the environments that you are upgrading and which applications will be upgraded in which sequence. Covering all of the permutations of an upgrade is beyond the scope of this document. Therefore, you should consider the following items prior to planning your upgrade. These items focus on upgrades that involve a single application running in a single domain.

- Oracle generally recommends that you upgrade an application in development environments and use a standard QA, testing, and staging process to move upgraded applications to a production environment.
- You will typically upgrade an application either by upgrading an existing domain or by creating a new domain, from which you can run the application on the new WebLogic Server version. The Domain Upgrade Wizard supports upgrading of domains from older WebLogic Server versions to newer WebLogic Server versions.

  The Domain Upgrade Wizard upgrades domains from older WebLogic Server versions to newer WebLogic Server versions. If you upgrade a domain, you do not need to recreate domain configurations, and you can leverage existing domain configurations that are know to support your existing applications.

  There will, however, be cases where you may prefer to create new domains using the Fusion Middleware Configuration Wizard or other configuration tools (such as WLST) in order to test the applications that you are upgrading.
- When planning a WebLogic Server version upgrade, you should review the Fusion Middleware Supported Systems Configurations page to ensure that your upgraded environment is supported by Oracle, in particular:
  - current and planned JVM and JDK versions
  - operating system versions
  - database versions
  - Web services versions
  - versions of other products that interoperate with or run on WebLogic Server, to ensure that the upgraded environment is supported by Oracle or other vendors' products that you are using with WebLogic Server.
- On an ongoing basis, Oracle documents APIs and features that have been deprecated (that is, planned for removal in a future release). This is intended to inform you that you should avoid using these APIs and features to ensure upgradeability. Oracle also documents the APIs and features that have actually been removed in the current release so that if you are upgrading from prior versions, you can determine if your applications will be affected by an upgrade.

  APIs and feature removals are cumulative. For example, if you are upgrading from WebLogic Server 10.0 to WebLogic Server 12.1.1, your applications may be affected by APIs or features that were removed in WebLogic Server 10.3, as well as by APIs or features that were removed in WebLogic Server 12.1.1. When upgrading, you should review all documentation of deprecated and removed features for all applicable WebLogic Server versions.
- You should consider the impact (if any) that the upgrade process may have on any automation (such as WLST scripts) that you are using to configure, deploy, start/stop, or monitor your WebLogic Server applications. You may need to upgrade such automation along with the applications and domains you are upgrading.
- You should consider the potential impact that may result from the use of third-party libraries in your applications, as they may conflict with different versions of those same libraries that are embedded in WebLogic Server. In particular, new versions of WebLogic Server my change the version of open source libraries that are embedded in WebLogic Server. Applications that may run successfully on earlier WebLogic Server versions may encounter new class conflicts after upgrade.

If you are upgrading an application that contains embedded third-party libraries, you should consider using the Classloader Analysis Tool, and filtering classloaders when upgrading WebLogic Server applications to WebLogic Server 12.1.1. This tool enables you to identify, diagnose and resolve such conficts, and may simplify the upgrade process.

- If you are running applications on prior versions of WebLogic Server, and are using WebLogic Server patches or bug fixes, you should investigate whether or not those patches or bug fixes have been incorporated into the version of WebLogic Server to which you are upgrading.

1.4 How the Upgrade Wizard Simplifies the Upgrade Process

The WebLogic Upgrade Wizard guides you through the steps required to upgrade a WebLogic Server 8.1 domain such that it runs in a WebLogic Server 12.1.1 application environment. As part of the upgrade process, you must upgrade any custom security providers and Node Managers used in the domain.

You can also use the WebLogic Upgrade Wizard to upgrade to a WebLogic Server 12.1.1 domain that is compatible with WebLogic Server 9.*x* or 10.x, but this is optional. This type of domain runs under WebLogic Server 12.1.1 without modification.

You can step through the upgrade process interactively, using the graphical user interface (GUI), or "silently", by creating an upgrade script and running it. Silent Mode is supported for upgrading a WebLogic domain, security providers, and Node Manager.

1.5 Interoperability and Compatibility with Previous Releases

Application environments that run with WebLogic Server 12.1.1 can interact with application environments built on WebLogic Server 8.1, 9.*x*, 10.0, or 10.3.

Most existing WebLogic Server applications can be run without modification in the new WebLogic Server 12.1.1 application environment. You should review the compatibility information described in Appendix A, "WebLogic Server 12.1.1 Compatibility with Previous Releases," to determine whether any feature changes affect the applications in your environment. If your application uses APIs that have been deprecated or removed, then you may encounter warnings or exceptions at run time.

2

Roadmap for Upgrading Your Application Environment

This chapter describes how to prepare for and perform an upgrade of your WebLogic application environments.

Topics include:

- Section 2.1, "Plan the Upgrade"
- Section 2.2, "Prepare to Upgrade"
- Section 2.3, "Upgrade Your Application Environment"
- Section 2.4, "Complete Post-Upgrade Procedure"
- Section 2.5, "What to Do If the Upgrade Process Fails"

If you are upgrading from WebLogic 9.x or 10.x, see Chapter 6, "Upgrading WebLogic Server 9.x or 10.x Application Environments to 12.1.1."

> **Note:** Oracle does not require WebLogic domains to be upgraded from WebLogic Server 10.3 to 12.1.1. WebLogic domains based upon WebLogic Server 10.3 run on WebLogic Server 12.1.1 without modification.

2.1 Plan the Upgrade

Planning how you will upgrade an application environment is an important step in the process. To ensure that your plan addresses all of the aspects of upgrading that are necessary for your environment, complete the following steps:

- Step 1: Inventory the Application Environment
- Step 2: Verify Supported Configuration Information
- Step 3: Review the Compatibility Information
- Step 4: Create an Upgrade Plan

2.1.1 Step 1: Inventory the Application Environment

Generate an inventory of the application environment by identifying the following components:

- Administration Server and the computer on which it resides
- Managed Servers and the computer(s) on which they reside

- Custom security providers used in the domain
- Custom Node Managers used in the domain
- Location of the applications (including all external client applications)
- External resources, for example:
  - Databases used to store persisted and application data
  - Firewalls
  - Load balancers
- Tools, scripts, templates, and source code used for automating the tasks required to create the application environment

You can view a sample application environment in Section 1.2, "Overview of the Upgrade Process."

2.1.2 Step 2: Verify Supported Configuration Information

Verify support for all the hardware and software components in the application environment. Table 2–1 lists the key components for which you must verify support.

Table 2–1 Verify Supported Configuration Information

To verify ...	See ...
Operating system and hardware support	"List of Supported Operating System Configurations" in the Oracle Fusion Middleware Supported System Configurations page.
Database support	"Supported Database Configurations" in the Oracle Fusion Middleware Supported System Configurations page. Please note the following: ■ WebLogic Server 12.1.1 supports PointBase 5.7; however, PointBase is no longer included in the WebLogic Server installation program. (Apache Derby replaces PointBase for running WebLogic Server samples.) You must use a full installer to upgrade to WebLogic Server 12.1.1. Therefore, the PointBase installation directory is not preserved. To continue using PointBase, you must: a) complete the instructions described in Section 2.4.2, "Step 2: Re-apply Customizations to Startup Scripts," *after* you upgrade your domain; and b) obtain a PointBase license from `http://www.pointbase.com`. See also Section 5.4.1, "Using PointBase in an Upgraded Domain." **Note:** The pre-5.7 version of PointBase that was distributed with earlier versions of WebLogic Server can be used only for WebLogic domains. ■ As of WebLogic Server 10.3.3, the evaluation database available from the installation program that is provided for use by the sample applications and code examples, and as a demonstration database, is changed from PointBase to Apache Derby. Derby is an open source relational database management system based on Java, JDBC, and SQL standards. For more information about Derby, see `http://db.apache.org/derby/`. If you have a domain based on PointBase from an earlier version of WebLogic Server, and you plan to upgrade that domain to WebLogic Server 12.1.1, you can continue to use PointBase. But you must obtain a license from `http://www.pointbase.com` to use it. For more information, see Section 5.4, "Upgrading a Domain that Uses an Evaluation Database." For information about migrating the domain database from PointBase to Derby, see Section 5.4.2, "Migrating an Upgraded Domain Database to Derby." ■ As of WebLogic Server 10.3, the Oracle Thin Driver is included as part of the WebLogic Server installation.

Table 2–1 (Cont.) Verify Supported Configuration Information

To verify ...	See ...

	■ The WebLogic jDriver for Oracle was removed in the 9.0 release. If the JDBC connection pools in your domain use the WebLogic jDriver to create database connections, you must reconfigure the upgraded data sources to use a different JDBC driver, which may include changing the driver class name, the database URL format, and properties sent to the driver when creating database connections. As a replacement, you can use the Oracle Thin Driver included that is included in your installation or you can use any JDBC driver that implements the specification and is thread safe. The driver you select must be installed (in the CLASSPATH) on all servers on which the data source is deployed. For more information about supported JDBC drivers, see "Supported Database Configurations" in the Oracle Fusion Middleware Supported System Configurations page.
	■ The Oracle 8.1.7 Oracle Thin driver (`oracle.jdbc.driver.OracleDriver`) is not supported in WebLogic Server 10.*x*. If your domain contains `JDBCConnectionPools` that are configured to use this driver, it is recommended that you reconfigure the connection pools to use another driver. Use of this driver causes the domain upgrade to fail during database upgrade.
	■ If you are using an Oracle OCI database driver and want to change to use a Thin database driver, you must remove the `server` property (as illustrated below) from the generated JDBC module. For example:
	`<property>`
	`<name>server</name>`
	`<value>`*`servername`*`</value>`
	`</property>`
	■ The WebLogic Type 4 JDBC Driver for Oracle (deprecated) and the Oracle Thin Driver are installed with WebLogic Server and are ready for use. For more information about using these drivers, see "Using WebLogic Type 4 JDBC Drivers" in *Type 4 JDBC Drivers for Oracle WebLogic Server* and "Third-Party JDBC Drivers Installed with WebLogic Server" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

Web servers, browsers, and firewalls	"Supported Web Servers, Browsers, and Firewalls" in the Oracle Fusion Middleware Supported System Configurations page.

2.1.3 Step 3: Review the Compatibility Information

Most existing WebLogic Server applications can be run without modification in the new WebLogic Server 12.1.1 application environment. However, you should review Appendix A, "WebLogic Server 12.1.1 Compatibility with Previous Releases," to determine whether any feature changes affect the applications in your environment.

2.1.4 Step 4: Create an Upgrade Plan

Using the information gathered in the preceding steps, create a plan for upgrading your application environment. Identify the scope and timing of the upgrade process, based on your business needs. Please note the following:

- Oracle does not recommend upgrading an application environment that is currently deployed in production. Instead, you should upgrade your application environment while it is under development or test and execute standard procedures for quality assurance and performance tuning before promoting the upgraded environment to production.
- If your application is complex, for example, if it includes multiple clustered domains and a large number of deployed applications, you may choose to upgrade the components of the application environment in stages.

- You may consider limiting the number of WebLogic Server versions used in any single application environment to minimize the diversity and cost of systems being administered.
- If you use transactional message-driven beans (MDBs) driven by non-WebLogic XA-enabled JMS providers, you must gracefully complete all pending transactions before shutting down and upgrading a server. Pending transactions cannot be recovered after an upgrade. For more information about message-driven beans, see "Message-Driven EJBs" in *Programming WebLogic Enterprise JavaBeans for Oracle WebLogic Server*.
- If you plan to use the RDBMS security store in a WebLogic domain, Oracle recommends that you create a new domain in which the RDBMS security store is configured. If you have an existing domain in which you want to use the RDBMS security store, you should create the new domain, then migrate your security realm to it. Oracle does not recommend "retrofitting" the RDBMS security store to an existing domain. For more information, see "Managing the RDBMS Security Store" in *Securing Oracle WebLogic Server*.

2.2 Prepare to Upgrade

Before you upgrade the application environment, you must perform the following steps:

- Step 1: Check Your Applications (Undeploy If Necessary)
- Step 2: Shut Down Servers in the Application Environment
- Step 3: Back Up the Application Environment
- Step 4: Install Required Oracle Products
- Step 5: Prepare the Remote Managed Server Domain Directories
- Step 6: Set Up the Environment

2.2.1 Step 1: Check Your Applications (Undeploy If Necessary)

It is not necessary for WebLogic Server applications to be undeployed before upgrading the domain. In most cases, WebLogic Server applications can be run without modifications in the new WebLogic Server 12.1.1 application environment. Review the compatibility information in Appendix A, "WebLogic Server 12.1.1 Compatibility with Previous Releases," to determine whether any features changes affect the applications in your environment. Note that if you use deprecated or removed APIs in the application, you might encounter warnings or exceptions at run time.

2.2.2 Step 2: Shut Down Servers in the Application Environment

Before you upgrade, you must shut down all servers in the application environment.

2.2.3 Step 3: Back Up the Application Environment

You have the option of backing up the domain during the upgrade process, as described in "Backup Domain" in Section 5–1, " Procedure for Upgrading a WebLogic Domain." However, the wizard archives the domain directory only; it does not preserve file permissions.

Oracle recommends that before upgrading your application environment, you manually back up the components defined in Table 2–2. You should back up the relevant information on all machines in the domain.

Table 2–2 Recommendations for Backing Up the Application Environment

Component	Recommendations
Domain directory	Back up the Administration Server and any remote Managed Server domain directories that are defined in the application environment. By default, the Configuration Wizard creates a domain directory in the *MW_HOME*\user_projects directory, where *MW_HOME* represents the Middleware Home directory in which you installed WebLogic Server.
Applications and application-persisted data	Back up any applications and data that reside outside the domain directory. By default, applications are created in the *MW_HOME*\user_projects\applications directory, where *MW_HOME* represents the Middleware Home directory in which you installed WebLogic Server. In releases before 10.3.1, this location was called the BEA Home directory.
Custom security providers	Back up any custom security providers that you are using in your application environment. By default, security providers are located in *WL_HOME*\server\bin\mbeantypes, where *WL_HOME* specifies the root directory of the WebLogic Server installation.
Node Manager directory and scripts	Back up any Node Manager directories and scripts that you are using to manage your servers in a clustered environment. The names of the directory and script depend on your operating system: ■ Windows: *WL_HOME*\common\nodemanager *WL_HOME*\server\bin\startNodeManager.cmd ■ UNIX: *WL_HOME*/common/nodemanager *WL_HOME*/server/bin/startNodeManager.sh In the preceding path names, *WL_HOME* represents the root directory of the WebLogic Server installation, for example, c:\Oracle\Middleware\wlserver_12.1.
Log files	If it is important for you to maintain a record of all messages that are logged, back up the log files. As log files can be large, you may want to delete them to conserve disk space, if it is not important to maintain them.

2.2.4 Step 4: Install Required Oracle Products

Before upgrading your application environment, you must install the Oracle WebLogic products that you require on each computer in the domain. For more information about installing Oracle WebLogic products, see the *Installation Guide for Oracle WebLogic Server*.

Notes: Before you proceed with installation, note the following:

- If you are using Node Manager in your pre-10.0 installation, when installing the 12.1.1 product, you should set the Node Manager listen port to match the port number used in the pre-10.0 installation, if possible. The default value for the listen port for Node Manager is 5556.
- If you have an existing domain that uses PointBase and you plan to continue using PointBase in that domain after upgrading it to WebLogic Server 12.1.1, the PointBase installation will not be preserved. For more information, see Section 5.4, "Upgrading a Domain that Uses an Evaluation Database."

2.2.5 Step 5: Prepare the Remote Managed Server Domain Directories

Some configurations include Managed Servers running on one or more machines that are remote from the Administration Server for the domain. If you have this type of configuration, you must upgrade the domain directories on each of the machines that host the remote Managed Servers.

To prepare the remote domain directories, you must copy the following files from the root directory of the pre-upgraded domain directory on the host computer for the Administration Server to the root directory of the host domain(s) of the remote Managed Servers:

- `config.xml` (configuration file)
- `SerializedSystemIni.dat`

Note: If the database in your configuration is not compatible with WebLogic Server 12.1.1, the data must be upgraded to a database that is supported before it can be used in the new application environment. For more information, see Section 2.1.4, "Step 4: Create an Upgrade Plan."

2.2.6 Step 6: Set Up the Environment

To set up the environment for an upgrade:

1. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX).
2. Add the WebLogic Server classes to the CLASSPATH environment variable and *`WL_HOME`*`\server\bin` to the PATH environment variable, where *`WL_HOME`* refers to the top-level installation directory for WebLogic Server 12.1.1.

   You can perform this step by running the *`WL_HOME`*`\server\bin\setWLSEnv` script.
3. If you use JMS JDBC stores:
   a. Make sure the JDBC driver classes are added to the CLASSPATH environment variable.
   b. Start the corresponding database.

2.3 Upgrade Your Application Environment

Figure 2–1 identifies the steps required to upgrade your application environment.

Figure 2–1 Roadmap for Upgrading Your Application Environment

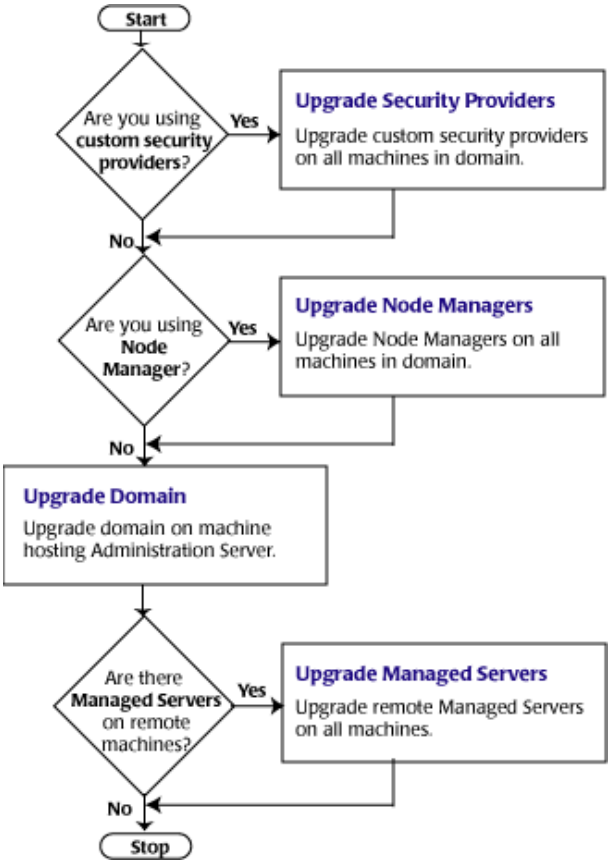


Table 2–3 summarizes the steps for updating an application environment. Some steps are mandatory, others are optional. Each step that is performed must be done on every computer in the domain and in the sequence shown in this table.

Table 2–3 Procedure for Upgrading an Application Environment

Step	Task	Description
1	Upgrade custom security providers	If are upgrading from WebLogic Server 8.1, and you have custom security providers in your current application environment that you want to continue using in the new environment, upgrade them: ■ On all machines in the environment. ■ Before you upgrade the WebLogic domain. **Note:** If you are installing WebLogic Server 12.1.1 into an existing directory that contains a pre-9.0 installation of WebLogic Server, all custom security providers that reside in the default location (*WL_HOME*\server\lib\mbeantypes) are upgraded automatically. If all of your custom security providers reside in the default location, then the security provider upgrade step is complete, and you do not have to perform the steps described in this section.
2	Upgrade Node Managers	If you are currently using a customized version of Node Manager to provide high availability to Managed Servers and you want to continue doing so in the new application environment, upgrade Node Manager: ■ On all machines in the environment ■ Before you start the servers in an upgraded WebLogic domain
3	Upgrade WebLogic domain (Administration Server)	Upgrade the WebLogic domain on the computer that hosts the Administration Server. **Note:** Oracle recommends that you upgrade the Administration Server for a domain before the Managed Servers.
4	Upgrade WebLogic domain (remote Managed Servers)	Upgrade the WebLogic domain on every computer that hosts any Managed Servers. Be sure to copy the appropriate files to the Managed Servers before performing the upgrade, as described in Section 2.2.5, "Step 5: Prepare the Remote Managed Server Domain Directories." **Note:** Managed Servers that reside on the same computer as the Administration Server do not require additional upgrade steps.

2.4 Complete Post-Upgrade Procedure

After you use the WebLogic Upgrade Wizard to upgrade the application environment, it might be necessary to perform the following steps:

- Step 1: Upgrade Your Application Infrastructure (applies only to upgrades from WebLogic Server 9.1 and earlier, or if you are still using the `weblogic.Admin` utility)
- Step 2: Re-apply Customizations to Startup Scripts
- Step 3: Verify File Permissions
- Step 4: Enroll the Computer with Node Manager
- Step 5: Verify Remote Server Startup Options
- Step 6: Promote the Application Environment to Production

Not all of these steps are required for all situations. Review the sections to determine which, if any, of these steps are appropriate for your environment.

2.4.1 Step 1: Upgrade Your Application Infrastructure

This section applies only if you are upgrading from WebLogic Server 9.1 or earlier, or if you are still using the `weblogic.Admin` utility. If you are upgrading from a later

release and you have already migrated to the WebLogic Scripting Tool, you can skip this step.

Due to recent changes in the MBean hierarchy, Oracle does not guarantee that all existing configuration and administration scripts (such as WLST, `wlconfig`, `weblogic.Admin`, Ant, and so on) can be run from all pre-9.2 environments. Oracle recommends that you update your scripts to take advantage of the new features introduced in WebLogic Server 9.2, 10.0, 10.3, and 12.1.1. For more information about new WebLogic Server features and changes in the MBean hierarchy introduced in each release since 9.2, see Section A.18, "WebLogic Administration and Configuration Scripts."

More information is provided in the following sections about scripting tools, custom domain templates, and SNMP:

- Section 2.4.1.1, "weblogic.Admin Utility is Deprecated"
- Section 2.4.1.2, "Using the WebLogic Scripting Tool"
- Section 2.4.1.3, "Upgrading Your Custom Domain Configuration Templates"
- Section 2.4.1.4, "Using SNMP to Monitor WebLogic Server"

2.4.1.1 weblogic.Admin Utility is Deprecated

The `weblogic.Admin` utility is deprecated as of WebLogic Server 9.0. For more information, see "weblogic.Server Command-Line Reference" in the *Command Reference for Oracle WebLogic Server*. If you are currently using `weblogic.Admin` utility, Oracle recommends that you use the Oracle WebLogic Scripting Tool, as described in the following section.

2.4.1.2 Using the WebLogic Scripting Tool

The WebLogic Scripting Tool (WLST) is a command-line scripting interface (built with Jython) that you can use to configure WebLogic domains. Using WLST, WebLogic Server administrators can perform administrative tasks and initiate WebLogic Server configuration changes interactively or by running an executable script.

Online and offline versions of WLST are delivered as a single tool. WLST fully supports the administrative and configuration features offered by 12.1.1. For more information about WLST, see *Oracle WebLogic Scripting Tool*.

As with the other pre-9.2 tools, Oracle does not guarantee that existing pre-9.2 WLST scripts can be run in 9.2 or 10.*x* due to recent changes in the MBean hierarchy. Oracle recommends that you update your scripts to take advantage of the new features provided with WebLogic Server 9.2, 10.0, 10.3, and 12.1.1.

For more information about new WebLogic Server features and changes in the MBean hierarchy introduced in each release since 9.2, see Section A.18, "WebLogic Administration and Configuration Scripts."

2.4.1.3 Upgrading Your Custom Domain Configuration Templates

Table 2–4 summarizes the steps required to upgrade custom domain or extension templates created with the Template Builder.

Table 2–4 Upgrade Procedure for Custom Domain Templates

Step	Task	More Information
1	Using the Upgrade Wizard, upgrade the domain that was created using the custom domain or extension template.	Follow the steps described in Section 2.3, "Upgrade Your Application Environment."
2	Modify the domain to leverage new features, as appropriate.	See *What's New in Oracle WebLogic Server*. For features introduced in each release of WebLogic Server since 9.2, see Table A–4.
3	Use the Template Builder or `pack` command to create a 12.1.1 domain or extension template.	See: ■ *Creating Domain Templates Using the Domain Template Builder* ■ *Creating Templates and Domains Using the Pack and Unpack Commands*

2.4.1.4 Using SNMP to Monitor WebLogic Server

If you use an SNMP manager to monitor WebLogic Server:

1. Load the WebLogic Server 12.1.1 MIB into your SNMP manager.

   The MIB is located at *`WL_HOME`*`/server/lib/BEA-WEBLOGIC-MIB.asn1`. WebLogic Server does not change object identifiers (OIDs) for existing managed objects; it only adds new OIDs for new managed objects.

2. If you are generating traps for any deprecated managed objects, create new traps for the replacement objects.

   For a list of deprecated managed objects, see "Deprecated MBeans" in *Oracle WebLogic Server MBean Reference*. The description of each deprecated MBean includes a pointer to the replacement MBean. (Each SNMP managed object corresponds to an MBean attribute.)

   Note: A number of run-time MBeans that are internal to Oracle have been removed from the MIB as of WebLogic Server 9.0. These MBeans are not included in the deprecated MBeans list. For more information, see "JMX Issues" in *WebLogic Server 9.0 Known and Resolved Issues*.

2.4.2 Step 2: Re-apply Customizations to Startup Scripts

To complete the upgrade of your application environment to 12.1.1, it might be necessary to re-apply any customizations to startup scripts. The following sections describe how to customize the default startup scripts as well as any custom startup scripts.

2.4.2.1 Default Startup Scripts

The Upgrade Wizard does not carry forward any customizations that have been made to the default startup scripts, such as the setting of the `JAVA_OPTIONS` environment variable. After the upgrade process is complete, you must customize the default scripts again.

If you are upgrading your domain to 12.1.1 and you want to continue using PointBase, you must add the PointBase JAR files to the beginning of the CLASSPATH environment variable definition. To do so, update the `set CLASSPATH` statement in your setDomainEnv files.

Note: WebLogic Server 12.1.1 supports PointBase 5.7; however, the use of any version of PointBase with WebLogic Server 10.3.3 or later requires a PointBase license, available at `http://www.pointbase.com`.

2.4.2.2 Custom Startup Scripts

If you have created custom startup scripts, you must update them manually, as follows:

- Set the JDK version to the JDK that you are using with WebLogic Server.
- Update the CLASSPATH variable, as follows:
  - Add WebLogic Server 12.1.1 classes to the beginning of the variable.
  - Remove all *unused* pre-10.3 WebLogic classes.
  - To continue using PointBase, include the PointBase database JARs at the beginning of the CLASSPATH environment variable definition.

    For more information about upgrading a domain that uses an evaluation database, see Section 5.4, "Upgrading a Domain that Uses an Evaluation Database."

2.4.3 Step 3: Verify File Permissions

Verify the file permissions, as follows:

- If you backed up the domain directory as part of the upgrade, you now must make your backup files secure because they might contain confidential information.
- During the upgrade process, file permissions are not preserved. If nondefault file permissions are set on files, they must be verified and reset.
- On a UNIX system, ownership and permissions for any new files created during the upgrade process are assigned to the user performing the upgrade. For example, if the upgrade is performed by root, then root is assigned ownership of any new files. As a result, any user who subsequently wants to update these files in the domain must have root privileges. You may want to review or modify the permissions on files created during the upgrade process.

2.4.4 Step 4: Enroll the Computer with Node Manager

If you upgrade Node Manager during the upgrade process, enroll the computer that is hosting the WebLogic domain with Node Manager. This can be accomplished using the `nmEnroll` command.

Note: If the nodemanager.domains file resides on the computer where the Administration Server and Managed Server are configured to run and they share the same domain directory, you can manually edit the file to include an entry for the domain, in the following form: *<domain-name>=<domain-directory>*.

This file is located in *WL_HOME*/common/nodemanager, by default (where *WL_HOME* refers to the top-level installation directory for WebLogic Server).

For more information, see "Configuring nodemanager.domains File" in "General Node Manager Configuration" in *Node Manager Administrator's Guide for Oracle WebLogic Server*.

The nmEnroll command updates the nodemanager.domains file under the *WL_HOME*/common/nodemanager directory with information about the domain, where *WL_HOME* refers to the top-level installation directory for WebLogic Server. The nodemanager.domains file specifies the domains that a Node Manager instance controls. This file is necessary so that standalone clients are not required to specify the domain directory explicitly.

This command also downloads the following files from the Administration Server:

- nm_password.properties, the Node Manager secret file, which contains the encrypted username and password that is used for server authentication
- SerializedSystemIni.dat file

To enroll the computer with Node Manager using the nmEnroll command:

1. Set up your environment, as described in "Setting Up Your Environment" in *Oracle WebLogic Scripting Tool*.
2. Invoke WLST, as described in "Invoking WLST" in *Oracle WebLogic Scripting Tool*.

   As described in step 2, start a WebLogic Server instance and connect WLST to the server using the connect command.
3. The moment WLST is connected to the Administration Server, enter the "nmEnroll" command to enroll the computer on which WLST is running with Node Manager.

   You have the option of specifying:

   - Path of the domain directory in which you want to save the Node Manager secret file (nm_password.properties) and SerializedSystemIni.dat file. By default, these files are saved in the directory in which WLST was started.
   - Path of the Node Manager home directory. The nodemanager.domains file, containing the information about the domain, is written to this directory. By default, the directory used for this purpose is *WL_HOME*/common/nodemanager, where *WL_HOME* refers to the top-level installation directory for WebLogic Server.

     For example, if the domain directory is specified as c:/bea/mydomain/common/nodemanager, and the default home directory for Node Manager, *WL_HOME*/common/nodemanager, is used, then you enroll the computer on which WLST is running with Node Manager by entering the following command, shown in **bold**:

```
wls:/mydomain/serverConfig> nmEnroll('c:/bea/mydomain/common/nodemanager')
Enrolling this machine with the domain directory at
c:\bea\mydomain\common\nodemanager....
Successfully enrolled this machine with the domain directory at
C:\bea\mydomain\common\nodemanager
wls:/mydomain/serverConfig>
```

For more information, see "nmEnroll" in *WebLogic Scripting Tool Command Reference*.

2.4.5 Step 5: Verify Remote Server Startup Options

When you start the Administration Server, verify the remote server start options, such as `JAVA_HOME`, `MW_HOME`, `BEA_HOME`, and `CLASSPATH`, reference the WebLogic Server 12.1.1 installation on the target Managed Server. This can be accomplished using the Administration Console, as described in "Configure startup arguments for Managed Servers" in *Oracle WebLogic Server Administration Console Help*.

> **Note:** If the remote server startup options are not set correctly, when attempting to start a Managed Server using Node Manager, messages similar to the following may be written to the log file. Because these messages may be sent recursively, they may eventually consume all space available on the drive.

```
No config.xml was found.

Would you like the server to create a default configuration and boot? (y/n):
java.io.IOException: The handle is invalid
at COM.jrockit.io.FileNativeIO.read(III)I(Native Method)
at COM.jrockit.io.NativeIO.read(Ljava.io.FileDescriptor;II)I(Unknown Source)
at COM.jrockit.io.NativeIOInputStream.read(II)I(Unknown Source)
at COM.jrockit.io.NativeIOInputStream.read(I[BI)I(Unknown Source)
at COM.jrockit.io.NativeIOInputStream.read([BII)I(Unknown Source)
at java.io.FileInputStream.read([BII)I(Unknown Source)
```

2.4.6 Step 6: Promote the Application Environment to Production

Execute standard procedures for quality assurance and performance tuning before promoting an application environment to production. You should test the execution of your applications (including external client applications) in your test application environment. If your applications use APIs that have been deprecated or removed, then you may encounter warnings or exceptions at run time. If you do, you can make any required modifications before promoting your applications to production.

When all test criteria have been met, you can promote the application environment to production, as outlined in your upgrade plan (defined previously in Section 2.1.4, "Step 4: Create an Upgrade Plan").

When the new 12.1.1 application environment is deployed into production, you can start redirecting requests to the new environment from the existing environment. Gradually, you can bring the existing environment to a safe state for shutdown. This might be accomplished using a load balancer, for example.

2.5 What to Do If the Upgrade Process Fails

If any step in the upgrade process fails, the WebLogic Upgrade Wizard displays a message indicating the reason for the failure and terminates. To proceed, perform the following steps:

1. Restore the application environment to its original state using the backup files created in Section 2.2.3, "Step 3: Back Up the Application Environment."
2. Correct the failure reported by the WebLogic Upgrade Wizard.
3. Continue with the upgrade process from the step at which the failure occurred, as described in Section 2.3, "Upgrade Your Application Environment."

3 Upgrading a Security Provider

If you are using a custom security provider in a WebLogic Server 8.1 environment, you can use the WebLogic Upgrade Wizard to upgrade your security provider for use in a WebLogic Server 12.1.1 application environment.

> **Notes:** As of 9.1, WebLogic Server includes two new security providers, the XACML Authorization provider and the XACML Role Mapping provider. Existing WebLogic domains that you upgrade to 12.1.1 continue to use the authorization and role mapping providers currently specified, such as third-party partner providers or the original WebLogic Authorization and Role Mapping providers. If you want, you can migrate existing domains from using WebLogic Server proprietary providers to the XACML providers, including performing bulk imports of existing policies. For more information, see "Understanding WebLogic Server Security" in *Understanding Oracle WebLogic Server*.

The following sections describe how to use the WebLogic Upgrade Wizard for upgrading the Security Provider.

- Section 3.1, "What Happens During a Security Provider Upgrade"
- Section 3.2, "Upgrading a Security Provider"

For information about developing custom security providers, see *Developing Security Providers for Oracle WebLogic Server*.

3.1 What Happens During a Security Provider Upgrade

For a security provider upgrade, you specify the source and destination directories for the upgrade, and the WebLogic Upgrade Wizard upgrades the existing JARs so that the security provider can run in a WebLogic Server 12.1.1 application environment.

Note: The security provider JAR must contain the appropriate MBean Definition File (MDF) that defines the MBean. An MDF is used to generate the `.java` files for a particular MBean type. For more information about creating MDFs, see *Developing Security Providers for Oracle WebLogic Server*. If an MDF is not located in the JAR file, the upgrade process fails for that specific security provider.

If an MDF contains undocumented tags, warnings are generated during the upgrade process. These warnings do not affect the upgrade, and can be ignored. To avoid any further such warnings, however, you may want to remove undocumented tags from the MDF.

Security realms defined in pre-9.2 configurations must define a lockout manager (`UserLockoutManagerMBean`), and must conform to the following naming convention for JMX objects: `Security:Name=`*`name`*. Otherwise, the upgrade process fails for the security provider.

During the upgrade, the Upgrade Wizard performs the following tasks:

- Adds required classes to the security provider JAR. These classes include `MBeanImpl` elements, schema files, and so on.
- Reads the MDF and creates the necessary schemas, MBean Implementation, and Binder classes.
- Stores the upgraded JARs for the security provider in the specified location.
- Appends `_Upgraded` to the security provider name to make the upgraded JARs distinct from the existing security provider JARs, which are maintained.
- Ignores security provider JARs that are shipped with Oracle products, or JARs with names that contain `_Upgraded`, indicating that they have been upgraded already.

3.2 Upgrading a Security Provider

You must upgrade each custom security provider to run in the WebLogic Server 12.1.1 environment.

Note: If you are installing WebLogic Server 12.1.1 into an existing home directory[1] that contains an installation of WebLogic Server 8.1, all custom security providers that reside in the default location (that is, *`WL_HOME`*`\server\lib\mbeantypes`, where *`WL_HOME`* represents the root directory of the pre-9.0 installation) are upgraded automatically. If all of your custom security providers reside in the default location, then you do not have to perform the security provider upgrade step described in this section.

[1] In releases of WebLogic Server before 10.3.1, this location was called the BEA Home directory. As of release 10.3.1, this location is called the Middleware home directory and is represented by the *`MW_HOME`* variable.

To verify that a custom security provider has been upgraded, locate the upgraded security provider, *`security_provider_name`*`_Upgraded`, in the *`WL_HOME`*`\server\lib\mbeantypes` directory, where *`WL_HOME`* specifies the root

directory of the 12.1.1 installation, and *security_provider_name* specifies the name of the security provider.

You can upgrade a security provider using the WebLogic Upgrade Wizard in one of the following modes:

- **Graphical**—For upgrading a security provider interactively, using the graphical user interface.
- **Silent**—For upgrading a security provider silently, by specifying upgrade requirements in a file.

You must upgrade a security provider on each computer in the domain.

The following sections describe how to upgrade a security provider:

- Section 3.2.1, "Upgrading a Security Provider in Graphical Mode"
- Section 3.2.2, "Upgrading a Security Provider in Silent Mode"

3.2.1 Upgrading a Security Provider in Graphical Mode

The following sections describe how to upgrade a security provider by using the WebLogic Upgrade Wizard in graphical mode:

- Section 3.2.1.1, "Starting the WebLogic Upgrade Wizard in Graphical Mode to Upgrade a Security Provider"
- Section 3.2.1.2, "Procedure for Upgrading a Security Provider"

> **Note:** The console from which you are running the Upgrade Wizard in graphical mode must support a Java-based GUI. If you attempt to start the Upgrade Wizard in graphical mode on a system that cannot support a graphical display, the invocation fails and an error message is displayed.

3.2.1.1 Starting the WebLogic Upgrade Wizard in Graphical Mode to Upgrade a Security Provider

> **Note:** Before proceeding, make sure you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

To start the WebLogic Upgrade Wizard in graphical mode and upgrade the security provider:

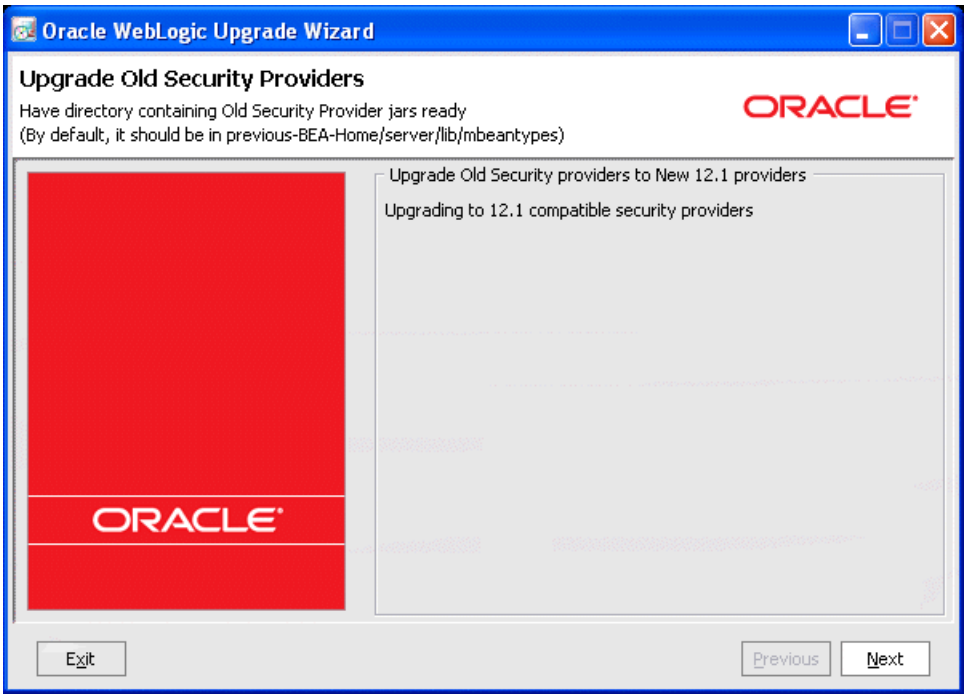
1. Verify that the WebLogic domain is not running.
2. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
3. At a command prompt, enter the following command:

   ```
   java weblogic.Upgrade -type securityproviders [-out file]
   ```

   The `-out` argument is optional. It allows you to designate a file in which you want all standard output (`stdout`) and error messages to be written. By default, these messages are written to the command window and a summary of them is

displayed when the upgrade process is complete. After you run the command, the WebLogic Upgrade Wizard opens, as shown in Figure 3–1.

Figure 3–1 WebLogic Upgrade Wizard for Security Providers



4. Click **Next** to proceed to the Select Source Directory window.

3.2.1.2 Procedure for Upgrading a Security Provider

Table 3–1 summarizes the steps in the procedure to upgrade a security provider using the WebLogic Upgrade Wizard.

Table 3–1 Procedure for Upgrading a Security Provider

In this step ...	You ...
Select Source Directory	Select the directory that contains the security provider JARs that must be upgraded. By default, the selected directory is the current directory. By default, security providers are located in *WL_HOME*\server\lib\mbeantypes, where *WL_HOME* specifies the root directory of the pre-9.0 installation of WebLogic Server. **Note:** The security provider JARs must contain the MBean Definition File (MDF) for the associated MBean. For more information about creating MDFs, see *Developing Security Providers for Oracle WebLogic Server*. If JAR file does not contain an MDF, the upgrade process fails for the associated security provider. Click **Next** to proceed to the Select Destination Directory window.

Table 3–1 (Cont.) Procedure for Upgrading a Security Provider

In this step ...	You ...
Select Destination Directory	Select the directory in which you want to save the new security provider JAR files. The default directory is *WL_HOME*\server\lib\mbeantypes, where *WL_HOME* specifies the root directory of the WebLogic Server 12.1.1 installation. **Note:** To ensure the success of the domain upgrade, you must target the upgraded security providers to the default destination directory, *WL_HOME*\server\lib\mbeantypes. If you prefer to keep the security providers in a different location, you can move them when the domain upgrade process is complete. Click **Next** to proceed to the next window.

Upgrade Security Providers in Progress	Review progress of the wizard as it saves the upgraded JARs and deletes any temporary files that were created during the upgrade process. Progress messages are displayed in the window.
	
	The security provider JAR must contain the MBean Definition File (MDF) for the associated MBean. For more information about creating MDFs, see *Developing Security Providers for Oracle WebLogic Server*. If a JAR file does not contain an MDF, the upgrade process fails for the associated security provider. For example:
	
	`Now processing mySecurityProviderToo.jar ...`
	`No MDFs (.xmls) found in the old security provider jar with name mySecurityProviderToo.jar`
	
	If an MDF contains undocumented tags, warnings are generated during the upgrade process. These warnings do not affect the upgrade; they can be ignored. To avoid further such messages, you may want to remove undocumented tags from the MDF.
	
	If the wizard locates a security provider JAR that was installed with the product, that has been upgraded already, or that is invalid, it does not upgrade that JAR. For example:
	
	`Not upgrading foo.txt because either this is a Out of the Box Oracle Security Provider jar or this Security Provider jar is already upgraded or this is not a valid archive (may be not a .jar)`
	
	Click **Next** to proceed to the next window.

Upgrade Complete	Review the upgrade results, including any important messages that require further consideration. Click **Done** to close the wizard.

3.2.2 Upgrading a Security Provider in Silent Mode

In some circumstances, for example, when the security provider resides on a remote computer, it is not practical to use the WebLogic Upgrade Wizard in graphical mode. In such situations, you can use the wizard in silent mode to upgrade a security provider.

> **Note:** Before proceeding, make sure you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

To start the WebLogic Upgrade Wizard in silent mode and upgrade a security provider:

1. Verify that the WebLogic domain is not running.

2. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
3. (Optional) Create an XML script to define the upgrade requirements. For more information, see Appendix D, "Silent Upgrade XML Script Reference."
4. Navigate to the directory that contains the security provider to upgrade.
5. At a command prompt, enter the following command:

   ```
   java weblogic.Upgrade -mode silent -type securityproviders [-responses xmlfile]
   [-out file]
   ```

   Two arguments are optional: `-responses` and `-out`. Include these arguments to override the default values for the following:

   - The location of an XML file that defines the upgrade requirements. If you do not specify a file with the `-responses` option, the wizard uses the default values during the upgrade process. For more information about the format of the XML file and the default values, see Appendix D, "Silent Upgrade XML Script Reference."
   - The output file in which all standard output (`stdout`) and error messages are written. If you do not specify a file with the `-out` argument, these messages are written to the command window.

4 Upgrading Node Manager

If you are using a customized version of Node Manager in a pre-10.*x* environment, you can use the WebLogic Upgrade Wizard to upgrade Node Manager for use in a WebLogic Server 12.1.1 application environment.

The following sections describe how to use the WebLogic Upgrade Wizard for this purpose:

- Section 4.1, "What Happens During a Node Manager Upgrade"
- Section 4.2, "Upgrading Node Manager"

> **Note:** Before proceeding, make sure you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

For more information about Node Manager, see "Node Manager Overview" in *Node Manager Administrator's Guide for Oracle WebLogic Server*.

4.1 What Happens During a Node Manager Upgrade

During a Node Manager upgrade, you specify the home directory of the Node Manager to upgrade. The WebLogic Upgrade Wizard performs the following tasks:

- Upgrades Node Manager in the directory you specify so that Node Manager runs in a WebLogic Server 12.1.1 application environment. The Upgrade Wizard upgrades the nodemanager.properties file and converts the NodeManagerSerializedSystemIni.dat to nm_data.properties.

  > **Note:** The nodemanager.properties file must be writable.

- Backs up existing log and state management files and stores them in a zip file, weblogic-nodemanager-backup.zip.

Any existing Node Manager files are overwritten during the upgrade process.

You are prompted to specify the username and password to be used for Node Manager authorization when upgrading the domain. For more information, see "Enter Node Manager Credentials" in Table 5–1. If you are upgrading to WebLogic Server 12.1.1 from any pre-9.0 installation, the listen port number used by Node Manager is maintained during the upgrade process.

Note: When installing the 12.1.1 product, you should, if possible, set the Node Manager listen port to the same port number used in any pre-9.0 installation, if applicable. The default listen port for Node Manager is 5556.

When the upgrade process is complete, perform the following:

- Enroll the computer with Node Manager, as described in Section 2.4.4, "Step 4: Enroll the Computer with Node Manager."
- Verify that the username, password, and listen port settings for Node Manager are set as desired.

4.2 Upgrading Node Manager

You must upgrade each instance of Node Manager to run in the WebLogic Server 12.1.1 environment. Specifically, you must upgrade Node Manager on every computer in the domain. You perform an upgrade using the WebLogic Upgrade Wizard in either of the following modes:

- **Graphical**—For upgrading Node Manager interactively, using the graphical user interface
- **Silent**—For upgrading Node Manager silently, by specifying upgrade requirements in a file

You must upgrade Node Manager on each computer in the domain.

The following sections describe how to upgrade Node Manager, including:

- Section 4.2.1, "Upgrading Node Manager in Graphical Mode"
- Section 4.2.2, "Upgrading Node Manager in Silent Mode"

4.2.1 Upgrading Node Manager in Graphical Mode

The following sections describe how to upgrade Node Manager using the WebLogic Upgrade Wizard in graphical mode:

- Section 4.2.1.1, "Starting the WebLogic Upgrade Wizard in Graphical Mode to Upgrade Node Manager"
- Section 4.2.1.2, "Procedure for Upgrading Node Manager"

Note: The console from which you are running the Upgrade Wizard in graphical mode must support a Java-based GUI. If you attempt to start the Upgrade Wizard in graphical mode on a system that cannot support a graphical display, the invocation fails and an error message is displayed.

4.2.1.1 Starting the WebLogic Upgrade Wizard in Graphical Mode to Upgrade Node Manager

Note: Before proceeding, make sure you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

To start the WebLogic Upgrade Wizard in graphical mode and upgrade Node Manager:

1. Verify that the WebLogic domain is not running.
2. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
3. If the Node Manager directory resides in a pre-10.0 installation directory, for example, in the default location, *WL_HOME*/common/nodemanager (where *WL_HOME* specifies the root directory of the WebLogic Server installation), copy the contents of the Node Manager directory to the following 12.1.1 installation directory:

   MW_HOME/wlserver_12.1/common/nodemanager

   In this case, you must upgrade the *copy* Node Manager in the 12.1.1 installation directory.

   Note: Make sure you maintain the current directory structure. It is not necessary for you to copy the log (.log) files to the new location.

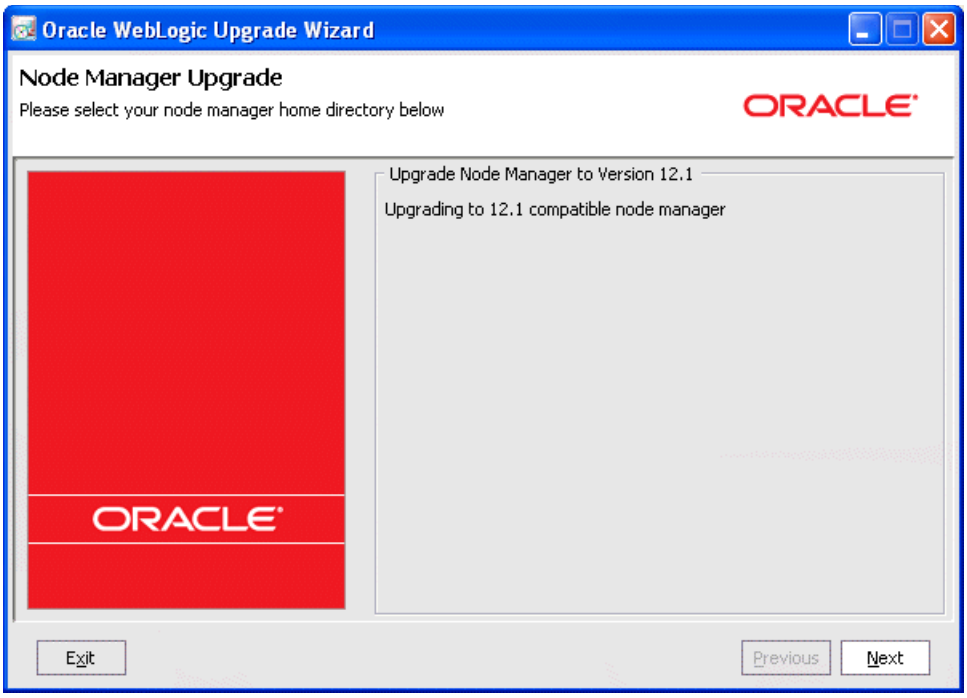
   If the Node Manager directory resides outside of the pre-10.0 installation directory, you can skip this step.
4. At a command prompt, enter the following command:

   ```
   java weblogic.Upgrade -type nodemanager [-out file]
   ```

   The -out argument is optional. It allows you to designate a file in which you want all standard output (stdout) and error messages to be written. By default, these messages are written to the command window and a summary of them is displayed when the upgrade process is complete.

   After you run the command, the WebLogic Upgrade Wizard opens, as shown in Figure 4–1.

Figure 4–1 WebLogic Upgrade Wizard for Node Manager



5. Click **Next** to proceed to the next window.

4.2.1.2 Procedure for Upgrading Node Manager

Table 4–1 summarizes the steps in the procedure to upgrade Node Manager using the WebLogic Upgrade Wizard.

Table 4–1 Procedure for Upgrading Node Manager

In this step ...	You ...
Node Manager Home Directory Selection	Select a home directory for the instance of Node Manager to be upgraded by navigating the local directory hierarchy. The Node Manager home directory contains the `nodemanager.properties` file, logs, and other related files. The exact set of files generated by Node Manager varies between releases and service packs. By default, the Node Manager home directory is *`WL_HOME`*`/common/nodemanager`, where *`WL_HOME`* specifies the root directory of the WebLogic Server installation. Click **Next** to proceed to the next window.
Upgrade Your Node Manager Home	Review progress of the wizard as it saves the upgraded configuration and deletes any temporary files that were created during the upgrade process. Progress messages are displayed in the window. After the process is complete, click **Next** to proceed to the next window.
Upgrade Complete	Review the upgrade results, including any important messages that require further consideration. Click **Done** to close the WebLogic Upgrade Wizard. **Note:** Before using Node Manager, you must enroll the computer, as described in Section 2.4.4, "Step 4: Enroll the Computer with Node Manager." This step should be performed after you complete the WebLogic domain upgrade process.

4.2.2 Upgrading Node Manager in Silent Mode

In some circumstances, for example, when Node Manager resides on a remote computer, it is not practical to use the WebLogic Upgrade Wizard in graphical mode. In such situations, you can use the wizard in silent mode to upgrade Node Manager.

> **Note:** Before proceeding, make sure you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

To start the WebLogic Upgrade Wizard in silent mode and upgrade Node Manager:

1. Verify that Node Manager and all instances of WebLogic Server in the domain are not running.
2. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
3. If the Node Manager directory resides in the pre-10.0 installation directory, for example, in the default location, *WL_HOME*`/common/nodemanager` (where *WL_HOME* specifies the root directory of the WebLogic Server installation), copy the contents of the Node Manager directory to the 12.1.1 installation directory.

   In this case, you must upgrade the *copy* of Node Manager in the 12.1.1 installation directory.

   > **Note:** Make sure you maintain the current directory structure. It is not necessary to copy the log (`.log`) files to the new location.

   If the Node Manager directory resides outside of the pre-10.0 installation directory, you can skip this step.
4. (Optional) Create an XML script to define the upgrade requirements. For more information, see Appendix D, "Silent Upgrade XML Script Reference."
5. Navigate to the Node Manager directory to upgrade.
6. At a command prompt, enter the following command:

   ```
   java weblogic.Upgrade -mode silent -type nodemanager [-responses xmlfile] [-out file]
   ```

   Two arguments are optional: `-responses` and `-out`. Include these arguments to override the default values for the following:

   - The location of an XML file that defines the upgrade requirements. If you do not specify a file with the `-responses` option, the wizard uses the default values during the upgrade process. For more information about the format of the XML file and the default values, see Appendix D, "Silent Upgrade XML Script Reference."
   - The output file in which all standard output (`stdout`) and error messages are written. If you do not specify a file with the `-out` argument, these messages are written to the command window.

Note: Before using Node Manager, you must enroll the computer, as described in Section 2.4.4, "Step 4: Enroll the Computer with Node Manager." This step should be performed after you complete the WebLogic domain upgrade process.

5 Upgrading a WebLogic Domain

You can use the 12.1.1 WebLogic Upgrade Wizard to upgrade domains created in WebLogic Server 8.1. You can also use the WebLogic Upgrade Wizard to upgrade a WebLogic domain created in WebLogic Server 9.x or 10.x to 12.1.1, but this is optional. This type of domain runs under WebLogic Server 12.1.1 without modification.

See Chapter 6, "Upgrading WebLogic Server 9.x or 10.x Application Environments to 12.1.1."

The following sections describe how to use the WebLogic Upgrade Wizard for this purpose:

- Section 5.1, "What Happens During a WebLogic Domain Upgrade?"
- Section 5.2, "Important Notes About the Domain Upgrade Process"
- Section 5.3, "Upgrading a Domain"
- Section 5.4, "Upgrading a Domain that Uses an Evaluation Database"

5.1 What Happens During a WebLogic Domain Upgrade?

During a WebLogic domain upgrade, you specify the domain to upgrade and respond to a set of prompts. The WebLogic Upgrade Wizard performs the following tasks:

1. Optionally, the wizard backs up the original domain directory.

   If a backup is requested, the wizard backs up the domain directory only, and it does not preserve file permissions. Oracle recommends that you back up the domain, any external applications, and application database resources in a separate process, as described in Section 2.2.3, "Step 3: Back Up the Application Environment."

   Note: Backup files created by the wizard must be protected by the user because they might contain confidential information.

2. Recreates scripts, such as startup and shutdown scripts, and renames any original scripts as *orig-scriptname*`.bak`, where *orig-scriptname* specifies the original script name and extension.

   Note: The wizard does not copy any customizations in the original startup scripts to the new scripts. For example, if you specified a nondefault value for the `JAVA_OPTIONS` environment variable in the original script, the specified value is not preserved in the new script.

3. Restructures the original domain, creating a new directory structure and moving domain components to new locations.

   During the restructuring, if a required directory already exists, the wizard simply keeps that directory and maintains the files and subdirectories that reside in it.

   Existing server log files are copied to the `servers/server_name/logs/pre-9.0-logs` directory in the domain, where *server_name* specifies the name of the server.

   To review changes to the domain directory structure, see Appendix B, "WebLogic Domain Directory Structure Enhancements."

4. Upgrades the persisted configuration information stored in the configuration file (`config.xml`) to the config directory.

   If the wizard encounters duplicate resources when upgrading the configuration file (`config.xml`), a message is logged in the progress window. In this case, the last resource definition encountered is used during the conversion.

5. Upgrades persisted data, such as JMS file stores, JMS JDBC stores, and transaction stores.

   Note: If JMS JDBC stores are used in the domain, see Section 2.2.6, "Step 6: Set Up the Environment."

   After the JMS JDBC stores are upgraded, the original JMS JDBC stores are not deleted. You should take this fact into account when performing capacity planning. You can delete the original JMS JDBC store tables after the upgrade is successfully complete. Original JMS JDBC store tables are named *PrefixName*`JMSSTORE` and *PrefixName*`JMSSTATE`, where *PrefixName* is the value of the `Prefix Name` attribute for the JMS JDBC store.

   If you do not want to upgrade persisted JMS messages, you can delete the JMS file store or JMS JDBC store tables before running the upgrade. When you do so, only JMS messages are lost; the configuration is not changed. For information about managing JDBC store tables, see "Managing JDBC Store Tables" in *Configuring Server Environments for Oracle WebLogic Server*.

   The wizard does not upgrade a JMS JDBC or file store if it detects that an upgrade has already been performed. If you must perform multiple upgrades of a domain in which the same persistent stores are use (for example, in a test scenario), revert the data in the JMS store each time you repeat the upgrade process, as follows:

   - For a JMS JDBC store, the upgrade process creates a new table named *PrefixName*`WLSTORE`, where *PrefixName* is the value of the `Prefix Name` attribute for the JMS JDBC store. Before re-running the upgrade process on a domain that uses the JMS JDBC store is used, to drop this table.
   - If you must re-run the upgrade, make sure you first restore the backed up version of the JMS file store.

6. Saves the configuration.

   Note: When upgrading remote Managed Servers, the wizard does not persist the configuration information.

7. Reports any issues with the domain upgrade that require further consideration.

5.2 Important Notes About the Domain Upgrade Process

Please note the following important notes about the upgrade process:

- It is not necessary for WebLogic Server applications to be undeployed. In most cases, WebLogic Server applications can be run without modifications in the new WebLogic Server 12.1.1 application environment. Review the compatibility information in Appendix A, "WebLogic Server 12.1.1 Compatibility with Previous Releases," to determine whether any features changes affect the applications in your environment. Note that if APIs that have been deprecated or removed are used in the application, then you may encounter warnings or exceptions at run time.
- At a minimum, the domain directory must contain the following files:
  - `config.xml`
  - Security-related files, including `SerializedSystemIni.dat`, `DefaultAuthenticatorInit.ldift`, `DefaultAuthorizerInit.ldift`, and `DefaultRoleMapperInit.ldift`

    If the security-related files are not available, the server fails to start and an authentication error message is logged.
  - Any transaction log (`.tlog`) files that reside in the domain. For more information, see "Transaction Log Files" in *Programming JTA for Oracle WebLogic Server*.
- All contents of the domain directory on the target computer are updated during this process.
- You must upgrade the domain on every computer in the application environment.
- The wizard does not upgrade applications during a WebLogic domain upgrade.
- Domains that contain resources for WebLogic Liquid Data, or AquaLogic Data Services Platform cannot be upgraded to WebLogic Server 12.1.1.
- As of 9.0, the `.wlnotdelete` directory is no longer used in the WebLogic Server environment.
- During the upgrade process, file permissions are not preserved. All nondefault file permissions must be verified and reset.
- On a UNIX system, ownership and permissions for any new files created during the upgrade process are assigned to the user performing the upgrade. For example, if the upgrade is performed by root, then root is assigned ownership of any new files. As a result, any user who subsequently wants to update these files in the domain must have root privileges. You may want to review or modify the permissions on files created during the upgrade process.
- When you upgrade a domain on a remote Managed Server, a message similar to the following may be displayed to indicate that the referenced application path does not reside on the system:

  ```
  <Apr 12, 2009 6:42:06 PM EDT> <INFO> <Upgrade> <BEA-800000> <An invalid
  path, 'C:\bea\wlserver_10.3\user_projects\mydomain\medrecEar.ear', was
  specified for application, 'medrecEar'.>
  ```

  This message can be ignored.
- If you upgraded the Avitek Medical Records application from 8.1 to 12.1.1 on a Solaris computer (only), before starting the server, you must edit the

setDomainEnv.sh file to remove `-Xverify:none` from the start command by setting `JAVA_OPTIONS=""` after the following line:

```
. ${WL_HOME}/common/bin/commEnv.sh
```

Otherwise, the server start fails with a JVM error.

5.3 Upgrading a Domain

The wizard supports the following upgrade modes:

- **Graphical**—For upgrading a domain interactively, the Domain Upgrade Wizard using a graphical user interface.
- **Silent**—Y ou can upgrade a WebLogic domain silently by specifying upgrade requirements in a file.

Note: You can also use implicit mode to upgrade a WebLogic domain automatically when the Administration Server is started. For more information, see Appendix E, "Upgrading a Domain at Administration Server Startup (Implicit Mode)."

You must upgrade the domains on every computer in the domain. For information about preparing remote Managed Server domain directories, see Section 2.2.5, "Step 5: Prepare the Remote Managed Server Domain Directories."

Note: Before proceeding, make sure you have:

- Performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."
- Reviewed the important notes described in Section 5.2, "Important Notes About the Domain Upgrade Process."
- Ensured that the WebLogic Domain is not running.
- Backed-up the JMS Store.

The following sections provide instructions for:

- Section 5.3.1, "Upgrading a Domain in Graphical Mode"
- Section 5.3.2, "Upgrading a Domain in Silent Mode"

5.3.1 Upgrading a Domain in Graphical Mode

The following sections describe how to upgrade a WebLogic domain using the WebLogic Upgrade Wizard in graphical mode:

- Section 5.3.1.1, "Starting the WebLogic Upgrade Wizard in Graphical Mode to Upgrade a Domain"
- Section 5.3.1.2, "Procedure for Upgrading a WebLogic Domain"

Note: The console from which you are running the Upgrade Wizard in graphical mode must support a Java-based GUI. If you attempt to start the Upgrade Wizard in graphical mode on a system that cannot support a graphical display, the invocation fails and an error message is displayed.

5.3.1.1 Starting the WebLogic Upgrade Wizard in Graphical Mode to Upgrade a Domain

To start the WebLogic Upgrade Wizard in graphical mode and upgrade a WebLogic domain on a Windows platform, choose the **Domain Upgrade Wizard** option from the Oracle program group in the Windows Start Menu by clicking **Programs** > **Oracle WebLogic** > **WebLogic Server 12c** > **Tools** > **Domain Upgrade Wizard**.

Note: You can only use this option if you do not have to customize the environment to specify JDBC driver classes, as described in step 3 of Section 2.2.6, "Step 6: Set Up the Environment."

To start the WebLogic Upgrade Wizard in graphical mode and upgrade a WebLogic domain from a Windows command prompt or on a UNIX platform:

1. Verify that the WebLogic domain is not running.
2. Backup the JMS Store, if applicable.
3. Open a command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
4. Execute the following script to upgrade your domains.
   - On Windows: *WL_HOME*`\common\bin\upgrade.cmd`
   - On Unix: *WL_HOME*`/common/bin/upgrade.sh`

   The log file is available in the *MW_HOME*`/user_projects/upgrade_logs` directory.

   The following command can also be used to upgrade a WebLogic domain.

   ```
   java weblogic.Upgrade [-type domain] [-out file]
   ```

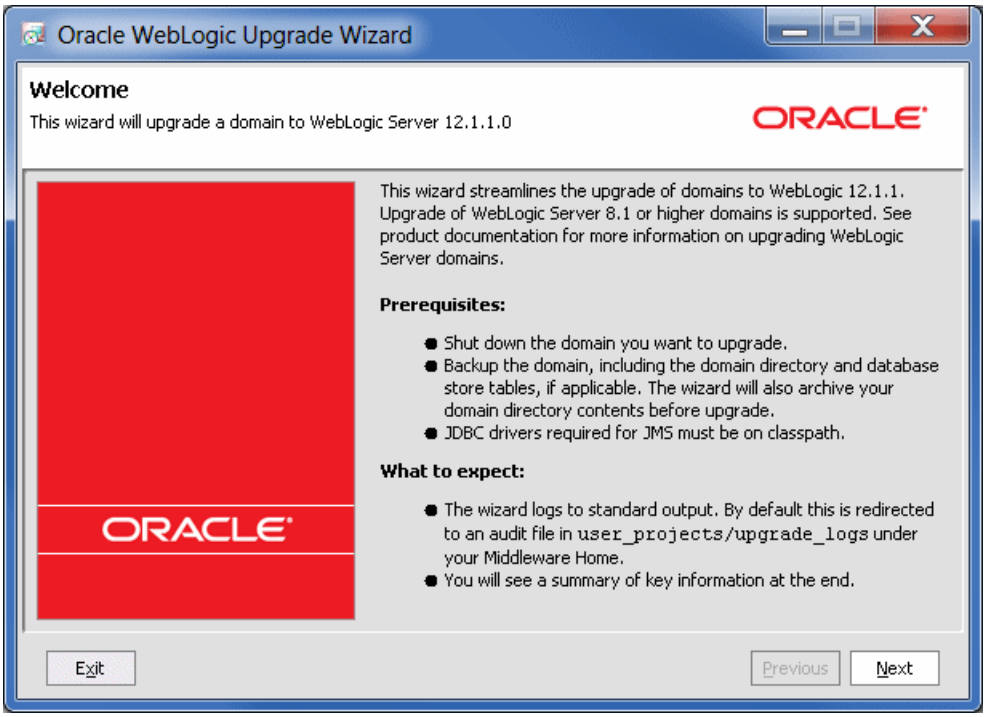
   Two arguments are optional: `-type` and `-out`. Include these arguments to override the default values for the following:

   - The type of upgrade to be performed. If you do not specify a type with the `-type` option, a domain upgrade is performed.
   - The output file in which all standard output (`stdout`) and error messages are written. If you do not specify a file with the `-out` option, such messages are written to the command window, and a summary of messages is displayed when the upgrade process is complete.

   The `-out` argument is optional. It allows you to designate a file in which you want all standard output (`stdout`) and error messages to be written. By default, these messages are written to the command window and a summary of them is displayed when the upgrade process is complete.

   After you run the command, the WebLogic Upgrade Wizard opens, as shown in Figure 5–1.

Figure 5–1 WebLogic Upgrade Wizard



5. If JMS JDBC stores are used, ensure that the corresponding database is running.
6. Click **Next** to proceed to the next window.

5.3.1.2 Procedure for Upgrading a WebLogic Domain

Table 5–1 summarizes the steps in the procedure to upgrade a domain using the WebLogic Upgrade Wizard.

> **Note:** The screens provided in this section are only indicative of what you might see. The actual screens you see depend on the combination of resources used in your domain.

Table 5–1 Procedure for Upgrading a WebLogic Domain

In this step ...	You ...
Select WebLogic Version	Select the WebLogic version of the domain that you are upgrading. Click **Next** to proceed to the next window.
Select a Domain to Upgrade	Select the directory that contains the WebLogic domain to be upgraded by navigating the local directory hierarchy. Click **Next** to proceed to the next window.

Table 5–1 (Cont.) Procedure for Upgrading a WebLogic Domain

In this step ...	You ...
Inspect Domain	Review progress of the wizard as it inspects the domain. Progress messages are displayed in the window. If you attempt to upgrade a domain in which custom security providers are used, without first upgrading those security providers, an error message is displayed and the wizard exits. If you receive this error message, upgrade the custom security providers, as described in Chapter 3.2, "Upgrading a Security Provider," and start the domain upgrade procedure again. After the inspection is complete (and if no error is encountered), the wizard advances to the next window automatically.
Select Administration Server	Select a server to function as the Administration Server in the new domain. **Note:** If there is only one server defined in the domain, this window is skipped. This window is displayed only if the domain you are upgrading has multiple servers. Click **Next** to proceed to the next window.
Enter Node Manager Credentials	Enter the username and password (and password confirmation) for Node Manager authorization. For WebLogic Server 12.1.1, Node Manager requires user and password credentials to be specified for each domain. By default, the username and password are set to `weblogic`. If you do not use Node Manager, leave the default values unchanged. Click **Next** to proceed to the next window.
Select Upgrade Options	▪ **Back up current domain** (recommended) — If selected, the wizard backs up the original domain directory and stores it in a zip file. This option is selected by default. **Note:** The wizard backs up the domain directory only and does not preserve file permissions. Oracle recommends that you back up the domain and any external application and application database resources in a separate process, as described in Section 2.2.3, "Step 3: Back Up the Application Environment." ▪ **Add log files to backup zip** — If selected, log files are included in the backup zip file. The number and size of log files can be large and you may want to disable this option to exclude them from the backup file. By default, log files are included in the backup file. ▪ **Do not set backward compatibility flags** — As of WebLogic Server 9.0, some previously supported behavior has changed to comply with J2EE 1.4. By default, the wizard sets flags to enable the previous behavior in the new domain. If you select this option, these flags are not set for backward compatibility. For more information about the backward compatibility flags, see Section A.2, "Backward Compatibility Flags."
Directory Selection Analysis and Optional Tasks	Review progress as the wizard processes the domain information and options provided. Progress messages are displayed in the window. After processing is complete, the wizard advances automatically to the next window.
Domain Backup	Review progress of the wizard as it prepares to back up the domain. Progress messages are displayed in the window. After processing is complete, the wizard advances automatically to the next window.

Table 5–1 (Cont.) Procedure for Upgrading a WebLogic Domain

In this step ...	You ...
Select Directory for Domain Backup	**Note:** If you did not select **Back up current domain** upgrade option, as described in the "Select Upgrade Options" step, skip to the "Restructure Domain Directory" step. In this window, set values for the following: ■ **Backup directory** — Navigate the local hierarchy and select the directory in which you want to save the backup zip file. By default, the original domain directory is used. ■ **Backup filename** — Enter the name of the backup file in the text box. The default filename is `weblogic-domain-backup-`*`domain`*`.zip`, where *domain* specifies the name of the domain. Click **Next** to proceed to the next window.
Backup Domain	Review progress as the wizard backs-up the domain. A progress bar displays the percentage of the backup process that is complete, and progress messages are displayed in the window. **Note:** Backup files created by the wizard must be protected by the user because they might contain confidential information. When the backup process is complete, the wizard advances automatically to the next window.
Restructure Domain Directory	Review progress as the wizard restructures the domain directory. Progress messages are displayed in the window. When the process is complete, the wizard automatically advances to the next window.
Upgrade Configuration Settings	Review progress as the wizard upgrades the configuration settings. Progress messages are displayed in the window. The configuration information is not persisted until a later step. After the configuration upgrade is complete, the wizard advances automatically to the next window.
Upgrade Persisted Messages and Transaction Log Formats	Review progress as the wizard upgrades the persisted messages (JMS file and JDBC stores) and transaction (JTA) logs that exist in the domain. A progress bar displays the percentage complete and progress messages are displayed in the window. After the persisted message and transaction log upgrade process is complete, the wizard advances to the next window automatically.
Upgrade RDBMS Authenticator Security Provider	Specify whether the deprecated RDBMSAuthenticator should be replaced by the SQLAuthenticator. **Note:** This window appears only when an RDBMSAuthenticator Security Provider exists in the domain you are upgrading. Click **Next** to proceed to the next window.
Persist Upgraded Configuration	Review progress of the wizard as it saves the upgraded configuration and deletes any temporary files that were created during the upgrade process. Progress messages are displayed in the window. **Note:** When upgrading remote Managed Servers, the wizard does not persist the configuration information. After this process is complete, click **Next** to proceed to the next window.
Upgrade Complete	Review the results of the upgrade, including any important messages that require further consideration. Click **Done** to close the WebLogic Upgrade Wizard.

5.3.2 Upgrading a Domain in Silent Mode

Note: Only WebLogic domains can be upgraded in silent mode.

In some circumstances, for example, when the domain resides on a remote computer, it is not practical to use the WebLogic Upgrade Wizard in graphical mode. In such situations, you can use the wizard in silent mode to upgrade the WebLogic domain.

Note: Before proceeding, make sure you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

To start the WebLogic Upgrade Wizard in silent mode and upgrade a WebLogic domain:

1. Verify that the WebLogic domain is not running.
2. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
3. (Optional) Create an XML script to define the upgrade requirements. For more information, see Appendix D, "Silent Upgrade XML Script Reference."
4. Navigate to the directory that contains WebLogic domain to upgrade. For example:

   ```
   cd c:\bea\user_projects\domains\domain
   ```

   where *domain* specifies the name of the domain.
5. At a command prompt, enter the following command:

   ```
   java weblogic.Upgrade -mode silent -type domain [-responses xmlfile] [-out file]
   ```

   The following arguments are optional: `-responses` and `-out`. Include these arguments to override the default values for the following:

   - The location of an XML file that defines the upgrade requirements. If you do not specify a file with the `-responses` option, the wizard uses the default values during the upgrade process. For more information about the format of the XML file and the default values, see Appendix D, "Silent Upgrade XML Script Reference."
   - The output file in which all standard output (`stdout`) and error messages are written. If you do not specify a file with the `-out` argument, these messages are written to the command window.

5.4 Upgrading a Domain that Uses an Evaluation Database

As of WebLogic 10.3.3, the evaluation database that is available from the installation program is changed from PointBase to Apache Derby. The evaluation database is provided for use by the sample applications and code examples and may be used as a demonstration database. If you are upgrading an examples or demonstration domain that was originally based on PointBase, you have two choices:

- Upgrade the domain so that it continues to use PointBase.

- Upgrade the domain and migrate its database to Derby.

The following sections explain how to implement each of these choices.

5.4.1 Using PointBase in an Upgraded Domain

To continue using PointBase as the database for a domain being upgraded to WebLogic 12.1.1, complete the following steps:

1. When installing WebLogic Server 12.1.1, as described in Section 2.2, "Prepare to Upgrade," you must use the full installer. The full installer does not preserve the PointBase installation from the previous WebLogic Server installation.
2. Complete the steps described in Section 2.3, "Upgrade Your Application Environment," and Section 2.4, "Complete Post-Upgrade Procedure."

   The domain's configuration settings for PointBase are preserved.
3. Obtain a PointBase license, available from `http://www.pointbase.com`.
4. Restore your PointBase installation.
5. Add the PointBase database JARs at the beginning of the CLASSPATH environment variable definition.

5.4.2 Migrating an Upgraded Domain Database to Derby

If you are upgrading an examples domain to WebLogic 12.1.1 and you want to migrate the domain database from PointBase to Derby, complete the following steps:

1. Install WebLogic Server 12.1.1 using the full installer, as described in Section 2.2, "Prepare to Upgrade."
2. Complete the steps described in Section 2.3, "Upgrade Your Application Environment," and Section 2.4, "Complete Post-Upgrade Procedure."
3. Import the PointBase database schema and tables to Derby. The Apache Derby Web site has tips and resources that you might find useful, at `http://db.apache.org/derby/integrate/index.html`.
4. Add the following Derby database JARs at the beginning of the CLASSPATH environment variable definition:
   - *WL_HOME*`\common\derby\lib\derbynet.jar`
   - *WL_HOME*`\common\derby\lib\derbyclient.jar`

6

Upgrading WebLogic Server 9.x or 10.x Application Environments to 12.1.1

The following sections describe the procedures for upgrading application environments from WebLogic Server 9.*x* or 10.x to the WebLogic Server 12.1.1 release:

- Section 6.1, "Creating a New Domain"
- Section 6.2, "Updating an Existing Domain"
- Section 6.3, "Upgrading Beehive Applications"

> **Note:** Oracle does not recommend upgrading an application environment that is currently deployed in production. Instead, you should upgrade your application environment while it is under development or test and execute standard procedures for quality assurance and performance tuning before promoting the upgraded environment to production.
>
> For information about upgrading the Web services in your application, see Chapter 7, "Upgrading WebLogic Web Services."

6.1 Creating a New Domain

This option is useful if the process of creating and customizing a domain has been automated. To create a new domain:

1. Install WebLogic Server 12.1.1 software.
2. To create your domain in 12.1.1, use a default domain template provided in 12.1.1. Alternatively, if you used a custom 9.*x* template to create your domain in 9.*x*, use the same to create a new 12.1.1 domain.

   This step can be accomplished using the Configuration Wizard or using automated scripts, built using WebLogic scripting tools such as WLST. The Configuration Wizard is described in *Creating Domains Using the Configuration Wizard*, and WLST is described in *Oracle WebLogic Scripting Tool*.

   It might be necessary for you to update your automated scripts, for example, to reference the 12.1.1 domain template or to implement new features provided in the 12.1.1 release.

   For example, starting in WebLogic 9.1, the default security providers are XACML-based, as described in "Understanding WebLogic Server Security" in *Understanding Oracle WebLogic Server*. You may choose to add support for the

security providers prior to WebLogic 9.1 or make the appropriate changes to use the new XACML-based security providers.

3. Deploy existing WebLogic Server 9.*x* applications to the new 12.1.1 domain.

Note: If you used a custom 9.*x* template in step 2, the 9.*x* applications may already be deployed.

6.2 Updating an Existing Domain

Updating an existing domain is useful for maintaining customizations within your test domain if generation of the domain has not been automated. There are two options for upgrading an existing domain, as follows:

- Option One: Manual Update
  1. Install WebLogic Server 12.1.1 software.
  2. Back up the existing application environment, including the domain directory, application and application data that is external to the domain, and log files (if necessary).
  3. Update the script files in the domain to point to the installation of WebLogic Server 12.1.1. For example, set `MW_HOME`, `MW_HOME`, `BEA_HOME`, `BEA_JAVA_HOME`, `JAVA_HOME`, and `WL_HOME` to the appropriate values.
  4. Update the CLASSPATH to remove path information that is no longer required, such as patch file information that applies to a pre-12.1.1 release.
- Option two: Automated Update

  Use the Domain Upgrade Wizard, as described in Chapter 5, "Upgrading a WebLogic Domain."

6.3 Upgrading Beehive Applications

If you developed Beehive applications in 9.0, 9.1 or 9.2, you must upgrade the applications as described in "Upgrade Paths" in *Beehive Integration in BEA WebLogic Server*, available at `http://download.oracle.com/docs/cd/E13222_01/wls/docs100/beehive/introduction.html#upgrade`.

Note: Use of Apache Beehive as the WebLogic Platform application framework is deprecated in this release. Oracle recommends that you use ADF, which offers a rich application framework and is the Oracle standard.

7

Upgrading WebLogic Web Services

The following sections describe the procedures for upgrading WebLogic Web services from WebLogic Server 9.x or 10.x to the WebLogic Server 12.1.*x* release. It also describes how to upgrade 8.1 WebLogic Web services to 12.1.*x* WebLogic Web services.

Topics include:

- Section 7.1, "Upgrading a 10.3.x RESTful Web Service (JAX-RS) to 12.1.x"
- Section 7.2, "Upgrading a 9.2 or 10.x WebLogic Web Service (JAX-WS or JAX-RPC) to 12.1.x"
- Section 7.3, "Upgrading a 9.0 or 9.1 WebLogic Web Service (JAX-WS or JAX-RPC) to 12.1.x"
- Section 7.4, "Upgrading an 8.1 WebLogic Web Service to 12.1.x"

Note: 9.2 and 10.3.*x* WebLogic Web services (JAX-WS or JAX-RPC) will continue to run, without any changes, on version 12.1.*x* of WebLogic Server because the associated Web services run time is still supported in this release, although they are deprecated and will be removed from the product in future releases. For this reason, Oracle highly recommends that you follow the instructions in this chapter to upgrade your 9.2 or 10.3.*x* Web service to 12.1.*x*.

7.1 Upgrading a 10.3.*x* RESTful Web Service (JAX-RS) to 12.1.*x*

In 10.3.*x*, a set of pre-built shared libraries were delivered with WebLogic Server to support Jersey 1.9 and 1.1.5.1 JAX-RS RI. In order to use the shared libraries, you needed to register them with the WebLogic Server instance, and modify the `web.xml` and `weblogic.xml` deployment descriptors to use the Jersey servlet and reference the shared libraries, respectively.

The following sections described the steps required to upgrade your environment so that the 10.3.*x* RESTful Web services run in the 12.1.*x* environment.

- Section 7.1.1, "Upgrading a 10.3.x RESTful Web Service That Uses the Jersey 1.9 Shared Libraries."
- Section 7.1.2, "Upgrading a 10.3.x RESTful Web Service That Uses the Jersey 1.1.5.1 Shared Libraries."

7.1.1 Upgrading a 10.3.*x* RESTful Web Service That Uses the Jersey 1.9 Shared Libraries

If your 10.3.*x* RESTful WebLogic Web service uses Jersey 1.9 shared libraries, then no additional steps are required to run the application in the 12.1.*x* environment. In this case, the Jersey 1.9 JAR file that is present in the system classpath is loaded by the WebLogic Server system classloader.

Note: If the RESTful Web service application is packaged with a servlet, ensure that it is packaged appropriately based on whether your Web application is using Servlet 3.0 or earlier. For more information, see "Packaging With a Servlet" in *Developing RESTful Web Services*.

7.1.2 Upgrading a 10.3.*x* RESTful Web Service That Uses the Jersey 1.1.5.1 Shared Libraries

---

Note: The Jersey 1.1.5.1 JAX-RS RI shared libraries are provided for backwards compatibility only. It is recommended that you upgrade your RESTful Web service applications to use the Jersey 1.9 JAX-RS RI APIs that are available on the WebLogic Server system classpath.

For backwards compatibility, the Jersey 1.1.5.1 JAX-RS RI shared libraries are delivered with the 12.1.*x* release of WebLogic Server. The shared libraries are located in the following directory: *WL_HOME*`/common/deployable-libraries`.

Table 7–1 summarizes the pre-built shared libraries that support Jersey JAX-RS RI Version 1.1.5.1 Web services, organized by the functionality that they support. The table also indicates whether the shared library is required or optional.

Table 7–1 Shared Libraries for Jersey JAX-RS RI 1.1.5.1

Functionality	Description	Required?
Jersey	■ Shared Library Name: jersey-bundle ■ JAR Filename: jersey-bundle-1.1.5.1.jar ■ WAR Filename: jersey-bundle-1.1.5.1.war ■ Version: 1.1.5.1 ■ License: SUN CDDL+GPL	Required
JAX-RS API	■ Shared Library Name: jsr311 ■ JAR Filename: jsr311-api-1.1.1.jar ■ WAR Filename: jsr311-api-1.1.1.war ■ Version: 1.1.1 ■ License: JSR311 license	Required
JSON processing	■ Shared Library Name: jackson-core-asl ■ JAR Filename: jackson-core-asl-1.1.1.jar ■ WAR Filename: jackson-core-asl-1.1.1.war ■ Version: 1.1.1 ■ License: Apache 2.0	Optional

Table 7–1 (Cont.) Shared Libraries for Jersey JAX-RS RI 1.1.5.1

Functionality	Description	Required?
JSON processing	■ Shared Library Name: jackson-jaxrs ■ JAR Filename: jackson-jaxrs-1.1.1.jar ■ WAR Filename: jackson-jaxrs-1.1.1.war ■ Version: 1.1.1 ■ License: Apache 2.0	Optional
JSON processing	■ Shared Library Name: jackson-mapper-asl ■ JAR Filename: jackson-mapper-asl-1.1.1.jar ■ WAR Filename: jackson-mapper-asl-1.1.1.war ■ Version: 1.1.1 ■ License: Apache 2.0	Optional
JSON streaming	■ Shared Library Name: jettison ■ JAR Filename: jettison-1.1.jar ■ WAR Filename: jettison-1.1.war ■ Version: 1.1 ■ License: Apache 2.0	Optional
ATOM processing	■ Shared Library Name: rome ■ JAR Filename: rome-1.0.jar ■ WAR Filename: rome-1.0.war ■ Version: 1.0 ■ License: Apache 2.0	Optional

To upgrade your environment so that the 10.3.*x* RESTful Web services run in the 12.1.*x* environment and continue to use the Jersey 1.1.5.1 shared libraries, you need to perform the following steps:

1. Register the Jersey 1.1.5.1 shared libraries with one or more WebLogic Server instances.

   Shared Java EE libraries are registered with one or more WebLogic Server instances by deploying them to the target servers and indicating that the deployments are to be shared. Shared Java EE libraries must be targeted to the same WebLogic Server instances you want to deploy applications that reference the libraries.

   The following shows an example of how to deploy the shared libraries that provide support for the basic Jersey JAX-RS RI functionality and JAX-RS API.

   ```
   weblogic.Deployer -verbose -noexit -source C:\myinstall\wlserver_
   10.3\common\deployable-libraries\jersey-bundle-1.1.5.1.war -targets myserver
   -adminurl t3://localhost:7001 -user system -password ******** -deploy -library

   weblogic.Deployer -verbose -noexit -source C:\myinstall\wlserver_
   10.3\common\deployable-libraries\jsr311-api-1.1.1.war -targets myserver
   -adminurl t3://localhost:7001 -user system -password ******** -deploy -library
   ```

   For more information about the weblogic.Deployer, see "weblogic.Deployer Command-Line Reference" in *Deploying Applications to Oracle WebLogic Server*.

2. If the RESTful Web service application is packaged with a servlet, ensure that it is packaged appropriately based on whether your Web application is using Servlet

3.0 or earlier. For more information, see "Packaging With a Servlet" in *Developing RESTful Web Services*.

3. Deploy your RESTful Web service application.

7.2 Upgrading a 9.2 or 10.x WebLogic Web Service (JAX-WS or JAX-RPC) to 12.1.*x*

No steps are required to upgrade a 9.2 or 10.*x* WebLogic Web service to 12.1.*x*; you can redeploy the JAX-WS or JAX-RPC Web service to WebLogic Server 12.1.*x* without making any changes or recompiling.

7.3 Upgrading a 9.0 or 9.1 WebLogic Web Service (JAX-WS or JAX-RPC) to 12.1.*x*

If your 9.0/9.1 Web service used any of the following features, then you must recompile the Web service before you redeploy it to WebLogic Server 12.1.*x*:

- Conversations
- `@weblogic.jws.Context` JWS annotation
- `weblogic.wsee.jws.JwsContext` API

To recompile, simply rerun the `jwsc` Ant task against the JWS file that implements your Web service.

If your 9.0/9.1 Web service did not use these features, then you can redeploy it to WebLogic Server 12.1.*x* without making any changes or recompiling it.

7.4 Upgrading an 8.1 WebLogic Web Service to 12.1.*x*

In the 12.1.1 release, the 8.1 WebLogic Web services run time has been removed. If you are using 8.1 WebLogic Web services, you need to upgrade the 8.1 WebLogic Web service applications to Web service stacks available in this release. The 8.1 WebLogic Web services rely on Apache XMLBeans for the purpose of mapping XML elements in SOAP payloads into Java objects and vice versa. XMLBeans are not supported in 12.1.1. The following upgrade paths are available:

- Upgrade to the WebLogic JAX-RPC stack: This is the simplest upgrade path in terms of level of effort required for upgrade. XMLBeans support in the WebLogic JAX-RPC stack is compatible with 8.1 WebLogic Web services. For more information, see "Upgrading an 8.1 WebLogic Web Service to the WebLogic JAX-RPC Stack" on page 7-5.
- Upgrade to the JAX-WS stack: This is the best upgrade path from the standpoint of taking advantage of latest technologies and standard support in WebLogic Server. This path requires a manual upgrade process, and the level of effort is determined by the nature of the existing 8.1 Web service applications. For example, if the applications have little XMLBeans usage, then the upgrade process is relatively easy. For 8.1 Web Service applications with heavy XMLBeans dependencies, you must modify the business logic in the service implementation in order to use JAXB classes instead of XMLBeans classes. Please note that JAX-WS does not support RPC-encoded style, and 8.1 Web Service applications with RPC-encoded style will need to adopt more interoperable literal style service contracts. For more information, see "Upgrading an 8.1 WebLogic Web Service to the WebLogic JAX-WS Stack" on page 7-20.

7.4.1 Upgrading an 8.1 WebLogic Web Service to the WebLogic JAX-RPC Stack

This section describes how to upgrade an 8.1 WebLogic Web service to use the WebLogic JAX-RPC run-time environment. The WebLogic JAX-RPC run time is based on the *JSR 109: Implementing Enterprise Web Services* specification at `http://www.jcp.org/en/jsr/detail?id=109`. The 12.1.*x* programming model uses standard JDK 1.5 metadata annotations, as specified by the *JSR 181: Web Services Metadata for the Java Platform* specification (JSR-181) at `http://www.jcp.org/en/jsr/detail?id=181`.

Upgrading your 8.1 Web service includes the following high-level tasks; the procedures in later sections go into more detail:

- Update the 8.1 Java source code of the Java class or stateless session EJB that implements the Web service so that the source code uses JWS annotations.

  In 12.1.*x*, WebLogic Web services are implemented using JWS files, which are Java files that contains JWS annotations. The `jwsc` Ant Task always implements the Web service as a plain Java file unless you explicitly implement `javax.ejb.SessionBean` in your JWS file. This latter case is not typical. This programming model differs from that of 8.1, where you were required to specify the type of back-end component (Java class or EJB).

- Update the Ant build script that builds the Web service to call the 12.1.*x* WebLogic Web service Ant task `jwsc` instead of the 8.1 `servicegen` task.

In the sections that follow it is assumed that:

- You previously used `servicegen` to generate your 8.1 Web service and that, more generally, you use Ant scripts in your development environment to iteratively develop Web services and other Java Platform, Enterprise Edition (Java EE) Version 5 artifacts that run on WebLogic Server. The procedures in this section direct you to update existing Ant `build.xml` files.
- You have access to the Java class or EJB source code for your 8.1 Web service.

This section does *not* discuss the following topics:

- Upgrading a JMS-implemented 8.1 Web service, because the WebLogic Web services JAX-RPC run time does not support JMS-implemented services.
- Upgrading Web services from versions previous to 8.1.
- Upgrading a client application that invokes an 8.1 Web service to one that invokes a 12.1.*x* Web service. For details on how to write a client application that invokes a 12.1.*x* Web service, see "Invoking Web Services" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server*.

7.4.1.1 Upgrading an 8.1 Java Class-Implemented WebLogic Web Service to 12.1.*x*: Main Steps

To upgrade an 8.1 Java class-implemented Web service to use the WebLogic Web services JAX-RPC run time:

1. Open a command window and set your WebLogic Server 12.1.*x* environment by executing the `setDomainEnv.cmd` (Windows) or `setDomainEnv.sh` (UNIX) script, located in the `bin` subdirectory of your 12.1.*x* domain directory.

   The default location of WebLogic Server domains is *`MW_HOME`*`/user_projects/domains/`*`domainName`*, where *`MW_HOME`* is the top-level installation directory of the Oracle products and *`domainName`* is the name of your domain.

2. Create a project directory:

```
prompt> mkdir /myExamples/upgrade_pojo
```

3. Create a `src` directory under the project directory, as well as sub-directories that correspond to the package name of the new 12.1.*x* JWS file (shown later in this procedure) that corresponds to the old 8.1 Java class:

```
prompt> cd /myExamples/upgrade_pojo
 prompt> mkdir src/examples/webservices/upgrade_pojo
```

4. Copy the old Java class that implements the 8.1 Web service to the `src/examples/webservices/upgrade_pojo` directory of the working directory. Rename the file, if desired.
5. Edit the Java file, as described in the following steps. See the old and new sample Java files in Section 7.4.1.1.1, "Example of an 8.1 Java File and the Corresponding 12.1.x JWS File" for specific examples.
   a. If needed, change the package name and class name of the Java file to reflect the new 12.1.*x* source environment.
   b. Add `import` statements to import both the standard and WebLogic-specific JWS annotations.
   c. Add, at a minimum, the following JWS annotation:
      – The standard `@WebService` annotation at the Java class level to specify that the JWS file implements a Web service.

        Oracle recommends you also add the following annotations:

      – The standard `@SOAPBinding` annotation at the class-level to specify the type of Web service, such as document-literal-wrapped or RPC-encoded.
      – The WebLogic-specific `@WLHttpTransport` annotation at the class-level to specify the context and service URIs that are used in the URL that invokes the deployed Web service.
      – The standard `@WebMethod` annotation at the method-level for each method that is exposed as a Web service operation.

        See "Programming the JWS File" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server* for general information about using JWS annotations in a Java file.
   d. You might need to add additional annotations to your JWS file, depending on the 8.1 Web service features you want to carry forward to 12.1.*x*. In 8.1, many of these features were configured with attributes of `servicegen`. See Section 7.4.1.3, "Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes" for a table that lists equivalent JWS annotation, if available, for features you enabled in 8.1 using `servicegen` attributes.
6. Copy the old `build.xml` file that built the 8.1 Web service to the 12.1.*x* working directory.
7. Update your Ant `build.xml` file to execute the `jwsc` Ant task, along with other supporting tasks, instead of `servicegen`.

   Oracle recommends that you create a new target, such as `build-service`, in your Ant build file and add the `jwsc` Ant task call to compile the new JWS file you created in the preceding steps. Once this target is working correctly, you can remove the old `servicegen` Ant task.

   The following procedure lists the main steps to update your `build.xml` file; for details on the steps, see the standard iterative development process outlined in

"Developing WebLogic Web Services" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server*.

See Section 7.4.1.1.2, "Example of an 8.1 and Updated 12.1.x Ant Build File for Java Class-Implemented Web Services" for specific examples of the steps in the following procedure.

1. Add the `jwsc` taskdef to the `build.xml` file.
2. Create a `build-service` target and add the tasks needed to build the 12.1.*x* Web service, as described in the following steps.
3. Add the `jwsc` task to the build file. Set the `srdir` attribute to the src directory (`/myExamples/upgrade_pojo/src`, in this example) and the `destdir` attribute to the root Enterprise application directory you created in the preceding step.

   Set the `file` attribute of the `<jws>` child element to the name of the new JWS file, created earlier in this procedure.

   You may need to specify additional attributes to the `jwsc` task, depending on the 8.1 Web service features you want to carry forward to 12.1.*x*. In 8.1, many of these features were configured using attributes of `servicegen`. See Section 7.4.1.3, "Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes" for a table that describes if there is an equivalent `jwsc` attribute for features you enabled using `servicegen` attributes.

8. Execute the `build-service` Ant target. Assuming all the tasks complete successfully, the resulting Enterprise application contains your upgraded 12.1.*x* Web service.

See "Deploying and Undeploying WebLogic Web Services" and "Browsing to the WSDL of the Web Service" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server* for additional information about deploying and testing your Web service.

Based on the sample Java code shown in the following sections, the URL to invoke the WSDL of the upgraded 12.1.*x* Web service is:

```
http://host:port/upgradePOJO/HelloWorld?WSDL
```

7.4.1.1.1 Example of an 8.1 Java File and the Corresponding 12.1.*x* JWS File

Assume that the following sample Java class implemented a 8.1 Web service:

```
package examples.javaclass;
/**
  * Simple Java class that implements the HelloWorld Web service.   It takes
  * as input an integer and a String, and returns a message that includes these
  * two parameters.
  */
public final class HelloWorld81 {
  /**
    * Returns a text message that includes the integer and String input
    * parameters.
    *
    */
   public String sayHello(int num, String s) {
    System.out.println("sayHello operation has been invoked with arguments " + s +
" and " + num);
    String returnValue = "This message brought to you by the letter "+s+" and the
number "+num;
    return returnValue;
   }
```

```
}
```

An equivalent JWS file for a 12.1.*x* Java class-implemented Web service is shown below, with the differences shown in bold. Note that some of the JWS annotation values are taken from attributes of the 8.1 `servicegen` Ant task shown in Section 7.4.1.1.2, "Example of an 8.1 and Updated 12.1.x Ant Build File for Java Class-Implemented Web Services."

```
package examples.webservices.upgrade_pojo;
// Import standard JWS annotations
import javax.jws.WebService;
 import javax.jws.WebMethod;
 import javax.jws.soap.SOAPBinding;
// Import WebLogic JWS anntoation
import weblogic.jws.WLHttpTransport;
/**
  * Simple Java class that implements the HelloWorld92 Web service.   It takes
  * as input an integer and a String, and returns a message that includes these
  * two parameters.
  */
@WebService(name="HelloWorld92PortType", serviceName="HelloWorld",
             targetNamespace="http://example.org")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
              use=SOAPBinding.Use.LITERAL,
              parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
@WLHttpTransport(contextPath="upgradePOJO", serviceUri="HelloWorld",
                  portName="HelloWorld92Port")
public class HelloWorld92Impl {
  /**
    * Returns a text message that includes the integer and String input
    * parameters.
    *
    */
  @WebMethod()
   public String sayHello(int num, String s) {
    System.out.println("sayHello operation has been invoked with arguments " + s +
" and " + num);
    String returnValue = "This message brought to you by the letter "+s+" and the
number "+num;
    return returnValue;
   }
 }
```

7.4.1.1.2 Example of an 8.1 and Updated 12.1.*x* Ant Build File for Java Class-Implemented Web Services

The following simple `build.xml` file shows the 8.1 way to build a WebLogic Web service using the `servicegen` Ant task; in the example, the Java file that implements the 8.1 Web service has already been compiled into the examples.javaclass.HelloWorld81 class:

```
<project name="javaclass-webservice" default="all" basedir=".">
  <!-- set global properties for this build -->
   <property name="source" value="."/>
   <property name="build" value="${source}/build"/>
   <property name="war_file" value="HelloWorldWS.war" />
   <property name="ear_file" value="HelloWorldApp.ear" />
   <property name="namespace" value="http://examples.org" />
  <target name="all" depends="clean, ear"/>
  <target name="clean">
     <delete dir="${build}"/>
   </target>
```

```
  <!-- example of old 8.1 servicegen call to build Web Service -->
  <target name="ear">
     <servicegen
        destEar="${build}/${ear_file}"
        warName="${war_file}">
        <service
          javaClassComponents="examples.javaclass.HelloWorld81"
          targetNamespace="${namespace}"
          serviceName="HelloWorld"
          serviceURI="/HelloWorld"
          generateTypes="True"
          expandMethods="True">
        </service>
      </servicegen>
   </target>
</project>
```

An equivalent `build.xml` file that calls the `jwsc` Ant task to build a 12.1.*x* Web service is shown below, with the relevant tasks discussed in this section in bold. In the example, the new JWS file that implements the 12.1.*x* Web service is called `HelloWorld92Impl.java`:

```
<project name="webservices-upgrade_pojo" default="all">
  <!-- set global properties for this build -->
  <property name="wls.username" value="weblogic" />
   <property name="wls.password" value="weblogic" />
   <property name="wls.hostname" value="localhost" />
   <property name="wls.port" value="7001" />
   <property name="wls.server.name" value="myserver" />
  <property name="ear.deployed.name" value="upgradePOJOEar" />
   <property name="example-output" value="output" />
   <property name="ear-dir" value="${example-output}/upgradePOJOEar" />
  <taskdef name="jwsc"
     classname="weblogic.wsee.tools.anttasks.JwscTask" />
  <taskdef name="wldeploy"
     classname="weblogic.ant.taskdefs.management.WLDeploy"/>
  <target name="all" depends="clean,build-service,deploy" />
  <target name="clean" depends="undeploy">
     <delete dir="${example-output}"/>
   </target>
  <target name="build-service">
    <jwsc
      srcdir="src"
      destdir="${ear-dir}">
     <jws file="examples/webservices/upgrade_pojo/HelloWorld92Impl.java" />
    </jwsc>
  </target>
  <target name="deploy">
     <wldeploy action="deploy" name="${ear.deployed.name}"
       source="${ear-dir}" user="${wls.username}"
       password="${wls.password}" verbose="true"
       adminurl="t3://${wls.hostname}:${wls.port}"
       targets="${wls.server.name}" />
   </target>
  <target name="undeploy">
     <wldeploy action="undeploy" name="${ear.deployed.name}"
       failonerror="false"
       user="${wls.username}" password="${wls.password}" verbose="true"
       adminurl="t3://${wls.hostname}:${wls.port}"
       targets="${wls.server.name}" />
```

```
   </target>
</project>
```

7.4.1.2 Upgrading an 8.1 EJB-Implemented WebLogic Web Service to 12.1.*x*: Main Steps

The following procedure describes how to upgrade an 8.1 EJB-implemented Web service to use the WebLogic Web services JAX-RPC run time.

The 12.1.*x* Web services programming model is quite different from the 8.1 model in that it hides the underlying implementation of the Web service. Rather than specifying up front that you want the Web service to be implemented by a Java class or an EJB, the `jwsc` Ant task always picks a plain Java class implementation, unless you have explicitly implemented `javax.ejb.SessionBean` in the JWS file, which is not typical. For this reason, the following procedure does not show how to import EJB classes or use EJBGen, even though the 8.1 Web service was explicitly implemented with an EJB. Instead, the procedure shows how to create a standard JWS file that is the 12.1.*x* equivalent to the 8.1 EJB-implemented Web service.

1. Open a command window and set your 12.1.*x* WebLogic Server environment by executing the `setDomainEnv.cmd` (Windows) or `setDomainEnv.sh` (UNIX) script, located in the `bin` subdirectory of your 12.1.*x* domain directory.

   The default location of WebLogic Server domains is *`MW_HOME`*`/user_projects/domains/`*`domainName`*, where *`MW_HOME`* is the top-level installation directory of the Oracle products and *`domainName`* is the name of your domain.

2. Create a project directory:

   ```
   prompt> mkdir /myExamples/upgrade_ejb
   ```

3. Create a `src` directory under the project directory, as well as sub-directories that correspond to the package name of the new 12.1.*x* JWS file (shown later on in this procedure) that corresponds to your 8.1 EJB implementation:

   ```
   prompt> cd /myExamples/upgrade_ejb
    prompt> mkdir src/examples/webservices/upgrade_ejb
   ```

4. Copy the 8.1 EJB Bean file that implemented `javax.ejb.SessionBean` to the `src/examples/webservices/upgrade_ejb` directory of the working directory. Rename the file, if desired.

   Note: You do not need to copy over the 8.1 Home and Remote EJB files.

5. Edit the EJB Bean file, as described in the following steps. See the old and new sample Java files in Section 7.4.1.2.1, "Example of 8.1 EJB Files and the Corresponding 12.1.x JWS File" for specific examples.

   a. If needed, change the package name and class name of the Java file to reflect the new 12.1.*x* source environment.

   b. Optionally remove the `import` statements that import the EJB classes (`javax.ejb.*`). These classes are no longer needed in the upgraded JWS file.

   c. Add `import` statements to import both the standard and WebLogic-specific JWS annotations.

   d. Ensure that the JWS file does *not* implement `javax.ejb.SessionBean` anymore by removing the `implements SessionBean` code from the class declaration.

e. Remove all the EJB-specific methods:

   - `ejbActivate()`
   - `ejbRemove()`
   - `ejbPassivate()`
   - `ejbCreate()`

f. Add, at a minimum, the following JWS annotation:

   - The standard `@WebService` annotation at the Java class level to specify that the JWS file implements a Web service.

     Oracle recommends you also add the following annotations:

   - The standard `@SOAPBinding` annotation at the class-level to specify the type of Web service, such as document-literal-wrapped or RPC-encoded.
   - The WebLogic-specific `@WLHttpTransport` annotation at the class-level to specify the context and service URIs that are used in the URL that invokes the deployed Web service.
   - The standard `@WebMethod` annotation at the method-level for each method that is exposed as a Web service operation.

   See "Programming the JWS File" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server* for general information about using JWS annotations in a Java file.

g. You might need to add additional annotations to your JWS file, depending on the 8.1 Web service features you want to carry forward to 12.1.*x*. In 8.1, many of these features were configured using attributes of `servicegen`. See Section 7.4.1.3, "Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes" for a table that lists equivalent JWS annotation, if available, for features you enabled in 8.1 using `servicegen` attributes.

6. Copy the old `build.xml` file that built the 8.1 Web service to the 12.1.*x* working directory.

7. Update your Ant `build.xml` file to execute the `jwsc` Ant task, along with other supporting tasks, instead of `servicegen`.

   Oracle recommends that you create a new target, such as `build-service`, in your Ant build file and add the `jwsc` Ant task call to compile the new JWS file you created in the preceding steps. Once this target is working correctly, you can remove the old `servicegen` Ant task.

   The following procedure lists the main steps to update your `build.xml` file; for details on the steps, see the standard iterative development process outlined in

   See Section 7.4.1.2.6, "Example of an 8.1 and Updated 12.1.x Ant Build File for an 8.1 EJB-Implemented Web Service" for specific examples of the steps in the following procedure.

   a. Add the `jwsc` taskdef to the `build.xml` file.

   b. Create a `build-service` target and add the tasks needed to build the 12.1.*x* Web service, as described in the following steps.

   c. Add the `jwsc` task to the build file. Set the `srdir` attribute to the src directory (`/myExamples/upgrade_ejb/src`, in this example) and the `destdir` attribute to the root Enterprise application directory you created in the preceding step.

Set the `file` attribute of the `<jws>` child element to the name of the new JWS file, created earlier in this procedure.

You may need to specify additional attributes to the `jwsc` task, depending on the 8.1 Web service features you want to carry forward to 12.1.*x*. In 8.1, many of these features were configured using attributes of `servicegen`. See Section 7.4.1.3, "Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes" for a table that indicates whether there is an equivalent `jwsc` attribute for features you enabled using `servicegen` attributes.

8. Execute the `build-service` Ant target. Assuming all tasks complete successfully, the resulting Enterprise application contains your upgraded 12.1.*x* Web service.

See "Deploying and Undeploying WebLogic Web Services" and "Browsing to the WSDL of the Web Service" in *Getting Started With JAX-RPC Web Services for Oracle WebLogic Server* for additional information about deploying and testing your Web service.

Based on the sample Java code shown in the following sections, the URL to invoke the WSDL of the upgraded 12.1.*x* Web service is:

```
http://host:port/upgradeEJB/HelloWorldService?WSDL
```

7.4.1.2.1 Example of 8.1 EJB Files and the Corresponding 12.1.*x* JWS File

Assume that the Bean, Home, and Remote classes and interfaces, shown in the next three sections, implemented the 8.1 stateless session EJB which in turn implemented an 8.1 Web service.

The equivalent 12.1.*x* JWS file is shown in Section 7.4.1.2.5, "Equivalent 12.1.x JWS File." The differences between the 8.1 and 12.1.*x* classes are shown in bold. Note that some of the JWS annotation values are taken from attributes of the 8.1 `servicegen` Ant task shown in Section 7.4.1.2.6, "Example of an 8.1 and Updated 12.1.x Ant Build File for an 8.1 EJB-Implemented Web Service."

7.4.1.2.2 8.1 SessionBean Class

```
package examples.statelessSession;
import javax.ejb.CreateException;
 import javax.ejb.SessionBean;
 import javax.ejb.SessionContext;
/**
  * HelloWorldBean is a stateless session EJB.  It has a single method,
  * sayHello(), that takes an integer and a String and returns a String.
  * <p>
  * The sayHello() method is the public operation of the Web service based on
  * this EJB.
   */
public class HelloWorldBean81 implements SessionBean {
  private static final boolean VERBOSE = true;
   private SessionContext ctx;
  // You might also consider using WebLogic's log service
   private void log(String s) {
     if (VERBOSE) System.out.println(s);
   }
  /**
    *   Single EJB business method.
    */
   public String sayHello(int num, String s) {
    System.out.println("sayHello in the HelloWorld EJB has "+
       "been invoked with arguments " + s + " and " + num);
    String returnValue = "This message brought to you by the "+
       "letter "+s+" and the number "+num;
```

```
    return returnValue;
   }
  /**
    * This method is required by the EJB Specification,
    * but is not used by this example.
    *
    */
   public void ejbActivate() {
     log("ejbActivate called");
   }
  /**
    * This method is required by the EJB Specification,
    * but is not used by this example.
    *
    */
   public void ejbRemove() {
     log("ejbRemove called");
   }
  /**
    * This method is required by the EJB Specification,
    * but is not used by this example.
    *
    */
   public void ejbPassivate() {
     log("ejbPassivate called");
   }
  /**
    * Sets the session context.
    *
    * @param ctx SessionContext Context for session
    */
   public void setSessionContext(SessionContext ctx) {
     log("setSessionContext called");
     this.ctx = ctx;
   }
  /**
    * This method is required by the EJB Specification,
    * but is not used by this example.
    *
    */
   public void ejbCreate () throws CreateException {
     log("ejbCreate called");
   }
}
```

7.4.1.2.3 8.1 Remote Interface

```
package examples.statelessSession;
import java.rmi.RemoteException;
 import javax.ejb.EJBObject;
/**
  * The methods in this interface are the public face of HelloWorld.
  * The signatures of the methods are identical to those of the EJBean, except
  * that these methods throw a java.rmi.RemoteException.
   */
public interface HelloWorld81 extends EJBObject {
  /**
    * Simply says hello from the EJB
    *
    * @param num              int number to return
    * @param s                String string to return
    * @return                 String returnValue
```

```
    * @exception              RemoteException if there is
    *                         a communications or systems failure
    */
   String sayHello(int num, String s)
     throws RemoteException;
 }
```

7.4.1.2.4 8.1 EJB Home Interface

```
package examples.statelessSession;
import java.rmi.RemoteException;
 import javax.ejb.CreateException;
 import javax.ejb.EJBHome;
/**
  * This interface is the Home interface of the HelloWorld stateless session EJB.
  */
 public interface HelloWorldHome81 extends EJBHome {
  /**
    * This method corresponds to the ejbCreate method in the
    * HelloWorldBean81.java file.
    */
   HelloWorld81 create()
     throws CreateException, RemoteException;
}
```

7.4.1.2.5 Equivalent 12.1.*x* JWS File

The differences between the 8.1 and 12.1.*x* files are shown in bold. The value of some JWS annotations are taken from attributes of the 8.1 `servicegen` Ant task shown in Section 7.4.1.2.6, "Example of an 8.1 and Updated 12.1.x Ant Build File for an 8.1 EJB-Implemented Web Service."

```
package examples.webservices.upgrade_ejb;
// Import the standard JWS annotations
import javax.jws.WebMethod;
 import javax.jws.WebService;
 import javax.jws.soap.SOAPBinding;
// Import the WebLogic specific annotation
import weblogic.jws.WLHttpTransport;
// Class-level annotations
@WebService(name="HelloWorld92PortType", serviceName="HelloWorldService",
            targetNamespace="http://example.org")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,
              use=SOAPBinding.Use.LITERAL,
              parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
@WLHttpTransport(contextPath="upgradeEJB", serviceUri="HelloWorldService",
                 portName="HelloWorld92Port")
/**
  * HelloWorld92Impl is the JWS equivalent of the HelloWorld81 EJB that
  * implemented the 8.1 Web Service.  It has a single method,
  * sayHello(), that takes an integer and a String and returns a String.
  */
public class HelloWorld92Impl {
  /** the sayHello method will become the public operation of the Web
    *  Service.
    */
  @WebMethod()
   public String sayHello(int num, String s) {
    System.out.println("sayHello in the HelloWorld92 Web Service has "+
       "been invoked with arguments " + s + " and " + num);
    String returnValue = "This message brought to you by the "+
       "letter "+s+" and the number "+num;
    return returnValue;
  }
```

```
}
```

7.4.1.2.6 Example of an 8.1 and Updated 12.1.*x* Ant Build File for an 8.1 EJB-Implemented Web Service

The following simple `build.xml` file shows the 8.1 way to build an EJB-implemented WebLogic Web service using the `servicegen` Ant task. Following this example is an equivalent `build.xml` file that calls the `jwsc` Ant task to build a 12.1.*x* Web service.

```
<project name="ejb-webservice" default="all" basedir=".">
  <!-- set global properties for this build -->
   <property name="source" value="."/>
   <property name="build" value="${source}/build"/>
   <property name="ejb_file" value="HelloWorldWS.jar" />
   <property name="war_file" value="HelloWorldWS.war" />
   <property name="ear_file" value="HelloWorldApp.ear" />
   <property name="namespace" value="http://examples.org" />
  <target name="all" depends="clean,ear"/>
  <target name="clean">
     <delete dir="${build}"/>
   </target>
  <!-- example of old 8.1 servicegen call to build Web Service -->
  <target name="ejb">
     <delete dir="${build}" />
     <mkdir dir="${build}"/>
     <mkdir dir="${build}/META-INF"/>
     <copy todir="${build}/META-INF">
       <fileset dir="${source}">
         <include name="ejb-jar.xml"/>
       </fileset>
     </copy>
     <javac srcdir="${source}" includes="HelloWorld*.java"
            destdir="${build}" />
     <jar jarfile="${ejb_file}" basedir="${build}" />
     <wlappc source="${ejb_file}" />
   </target>
 <target name="ear" depends="ejb">
    <servicegen
       destEar="${build}/${ear_file}"
       warName="${war_file}">
       <service
         ejbJar="${ejb_file}"
         targetNamespace="${namespace}"
         serviceName="HelloWorldService"
         serviceURI="/HelloWorldService"
         generateTypes="True"
         expandMethods="True">
       </service>
     </servicegen>
  </target>
</project>
```

An equivalent `build.xml` file that calls the `jwsc` Ant task to build a 12.1.*x* Web service is shown below, with the relevant tasks discussed in this section in bold:

```
<project name="webservices-upgrade_ejb" default="all">
  <!-- set global properties for this build -->
  <property name="wls.username" value="weblogic" />
   <property name="wls.password" value="weblogic" />
   <property name="wls.hostname" value="localhost" />
   <property name="wls.port" value="7001" />
```

```
 <property name="wls.server.name" value="myserver" />
<property name="ear.deployed.name" value="upgradeEJB" />
 <property name="example-output" value="output" />
 <property name="ear-dir" value="${example-output}/upgradeEJB" />
<taskdef name="jwsc"
   classname="weblogic.wsee.tools.anttasks.JwscTask" />
<taskdef name="wldeploy"
   classname="weblogic.ant.taskdefs.management.WLDeploy"/>
<target name="all" depends="clean,build-service,deploy" />
<target name="clean" depends="undeploy">
   <delete dir="${example-output}"/>
 </target>
<target name="build-service">
  <jwsc
     srcdir="src"
     destdir="${ear-dir}">
    <jws file="examples/webservices/upgrade_ejb/HelloWorld92Impl.java" />
  </jwsc>
</target>
<target name="deploy">
   <wldeploy action="deploy" name="${ear.deployed.name}"
     source="${ear-dir}" user="${wls.username}"
     password="${wls.password}" verbose="true"
     adminurl="t3://${wls.hostname}:${wls.port}"
     targets="${wls.server.name}" />
 </target>
<target name="undeploy">
   <wldeploy action="undeploy" name="${ear.deployed.name}"
     failonerror="false"
     user="${wls.username}" password="${wls.password}" verbose="true"
     adminurl="t3://${wls.hostname}:${wls.port}"
     targets="${wls.server.name}" />
  </target>
</project>
```

7.4.1.3 Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes

The following table maps the attributes of the 8.1 `servicegen` Ant task to their equivalent 12.1.*x* JWS annotation or `jwsc` attribute.

The attributes listed in the first column are a mixture of attributes of the main `servicegen` Ant task and attributes of the four child elements of `servicegen` (`<service>`, `<client>`, `<handlerChain>`, and `<security>`)

See "JWS Annotation Reference" and "jwsc" in the *WebLogic Web Services Reference for Oracle WebLogic Server* for more information about the 12.1.*x* JWS annotations and `jwsc` Ant task.

Table 7–2 Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes

servicegen or Child Element of servicegen Attribute	Equivalent JWS Annotation or jwsc Attribute
`contextURI`	`contextPath` attribute of the WebLogic-specific `@WLHttpTransport` annotation. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service.
`destEAR`	`destdir` attribute of the `jwsc` Ant task.
`keepGenerated`	`keepGenerated` attribute of the `jwsc` Ant task.

Table 7–2 (Cont.) Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes

servicegen or Child Element of servicegen Attribute	Equivalent JWS Annotation or jwsc Attribute
mergeWithExistingWS	No equivalent.
overwrite	No equivalent.
warName	name attribute of the <jws> child element of the jwsc Ant task.
ejbJAR (attribute of the service child element)	No direct equivalent, because the jwsc Ant task generates Web service artifacts from a JWS file, rather than a compiled EJB or Java class. Indirect equivalent is the file attribute of the <jws> child element of the jwsc Ant task that specifies the name of the JWS file.
excludeEJBs (attribute of the service child element)	No equivalent.
expandMethods (attribute of the service child element)	No equivalent.
generateTypes (attribute of the service child element)	No equivalent.
ignoreAuthHeader (attribute of the service child element)	No equivalent.
includeEJBs (attribute of the service child element)	No equivalent.
javaClassComponents (attribute of the service child element)	No direct equivalent, because the jwsc Ant task generates Web service artifacts from a JWS file, rather than a compiled EJB or Java class. Indirect equivalent is the file attribute of the <jws> child element of the jwsc Ant task that specifies the name of the JWS file.
JMSAction (attribute of the service child element)	No equivalent because JMS-implemented Web services are not supported in the 12.1.*x* release.
JMSConnectionFactory (attribute of the service child element)	No equivalent because JMS-implemented Web services are not supported in the 12.1.*x* release.
JMSDestination (attribute of the service child element)	No equivalent because JMS-implemented Web services are not supported in the 12.1.*x* release.
JMSDestinationType (attribute of the service child element)	No equivalent because JMS-implemented Web services are not supported in the 12.1.*x* release.
JMSMessageType (attribute of the service child element)	No equivalent because JMS-implemented Web services are not supported in the 12.1.*x* release.
JMSOperationName (attribute of the service child element)	No equivalent because JMS-implemented Web services are not supported in the 12.1.*x* release.

Table 7–2 (Cont.) Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes

servicegen or Child Element of servicegen Attribute	Equivalent JWS Annotation or jwsc Attribute
`protocol` (attribute of the `service` child element)	One of the following WebLogic-specific annotations: ■ `@WLHttpTransport` ■ `@WLJmsTransport` **Note**: Because these are WebLogic-specific annotations, you can use them to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service.
`serviceName` (attribute of the `service` child element)	`serviceName` attribute of the standard `@WebService` annotation.
`serviceURI` (attribute of the `service` child element)	`serviceUri` attribute of the WebLogic-specific `@WLHttpTransport` or `@WLJmsTransport` annotations. **Note**: Because these are WebLogic-specific annotations, you can use them to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service.
`style` (attribute of `service` child element)	`style` attribute of the standard `@SOAPBinding` annotation.
`typeMappingFile` (attribute of the `service` child element)	No equivalent.
`targetNamespace` (attribute of the `service` child element)	`targetNamespace` attribute of the standard `@WebService` annotation.
`userSOAP12` (attribute of the `service` child element)	`value` attribute of the WebLogic-specific `@weblogic.jws.Binding` JWS annotation **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service.
`clientJarName` (attribute of `client` child element)	No equivalent.
`packageName` (attribute of the `client` child element)	No direct equivalent. Use the `packageName` attribute of the `clientgen` Ant task to generate client-side Java code and artifacts.
`saveWSDL` (attribute of the `client` child element)	No equivalent.
`userServerTypes` (attribute of the `client` child element)	No equivalent.
`handlers` (attribute of the `handlerChain` child element)	Standard `@HandlerChain` or `@SOAPMessageHandlers` annotation.
`name` (attribute of the `handlerChain` child element)	Standard `@HandlerChain` or `@SOAPMessageHandlers` annotation.
`duplicateElimination` (attribute of the `reliability` child element)	No direct equivalent. Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains Web service reliable messaging policy assertions. See "Using Web Service Reliable Messaging" in *Programming Advanced Features of JAX-RPC Web Services for Oracle WebLogic Server*.

Table 7–2 (Cont.) Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes

servicegen or Child Element of servicegen Attribute	Equivalent JWS Annotation or jwsc Attribute
`persistDuration` (attribute of the `reliability` child element)	No direct equivalent. Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains Web service reliable messaging policy assertions. In this release, the equivalent is valid for JAX-RPC Web services only. See "Using Web Service Reliable Messaging" in *Programming Advanced Features of JAX-RPC Web Services for Oracle WebLogic Server*.
`enablePasswordAuth` (attribute of the `security` child element)	No direct equivalent. Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
`encryptKeyName` (attribute of the `security` child element)	No direct equivalent. Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
`encryptKeyPass` (attribute of the `security` child element)	No direct equivalent. Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
`password` (attribute of the `security` child element)	No direct equivalent. Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.

Table 7–2 (Cont.) Mapping of servicegen Attributes to JWS Annotations or jwsc Attributes

servicegen or Child Element of servicegen Attribute	Equivalent JWS Annotation or jwsc Attribute
`signKeyName`	No direct equivalent.
(attribute of the `security` child element)	Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
`signKeyPass`	No direct equivalent.
(attribute of the `security` child element)	Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
`username`	No direct equivalent.
(attribute of the `security` child element)	Use WebLogic-specific `@Policy` attribute to specify a WS-Policy file that contains message-level security policy assertions. **Note**: Because this is a WebLogic-specific annotation, you can use it to generate *only* a JAX-RPC Web service, and not a JAX-WS Web service. See "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*.

7.4.2 Upgrading an 8.1 WebLogic Web Service to the WebLogic JAX-WS Stack

This section summarizes how to upgrade an 8.1 WebLogic Web service to use the WebLogic JAX-WS stack.

The WebLogic JAX-WS run time is based on the JAX-WS (The Java API for XML-Based Web Services) 2.2 specification and the Web Services for Java EE v1.3 (JSR 109) specifications. These define annotations that are used in a Java Web Service (JWS) source file to define a Web service. Ant tasks are then used to compile the JWS into a Java class and generate all the associated artifacts. The Java Web Service (JWS) is the core of your JAX-WS web service.

Upgrading your 8.1 Web service includes the following high-level tasks:

- Upgrade any Web service EJBs from 2.*x* to 3.*x*.

  JAX-WS supports EJB 3.0 and 3.*x*. It does not support EJB 2.*x*.

- Rewrite the 8.1 Web service class as a JAX-WS JWS file and map any proprietary 8.x features to similar JAX-WS features.

  Note that there is not a one-to-one correspondence between 8.1 Web service features and JAX-WS 12.1.*x* features.

- Update the Ant build script that builds the Web service to call the 12.1.*x* WebLogic Web service Ant task `jwsc` instead of the 8.1 `servicegen` task.

- Generate new JAX-WS clients using the JAX-WS `clientgen` Ant task.

JAX-WS Upgrade Considerations

Before upgrading to JAX-WS, you should consider the following:

- The JAX-WS specification supports the "document-literal" and "rpc-literal" styles, but not "rpc-encoded". If you require rpc-encoded, you should consider upgrading your 8.1 Web service to the JAX-RPC model, rather than JAX-WS.
- SOAP Arrays are not supported by JAX-WS; they are available in JAX-RPC.

For more information about JAX-WS, refer to the following documents:

- *Getting Started With JAX-WS Web Services for Oracle WebLogic Server*
- *Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*

A

WebLogic Server 12.1.1 Compatibility with Previous Releases

This section describes important compatibility information that you should consider before upgrading to WebLogic Server 12.1.1.

See also "WebLogic Server Compatibility" in *Understanding Oracle WebLogic Server* and *What's New in Oracle WebLogic Server* for this and prior releases.

Compatibility considerations are provided in the following sections. In all cases, you should refer to:

- Section A.1, "Deprecated Functionality"
- Section A.2, "Backward Compatibility Flags"

The remaining sections that apply to your situation depend on the WebLogic Server version from which you are upgrading to WebLogic Server 12.1.1. Refer to Table A–1 for a list of sections to which you should refer based on your current WebLogic Server version.

Table A–1 Sections That Apply to Upgrades From Each WebLogic Server Version

WebLogic Server Version	Refer to These Sections
10.3.5	Section A.3, "JVM Settings" Section A.4, "Node Manager startScriptEnabled Default" Section A.5, "Enterprise Java Beans (EJBs)" Section A.6.1, "WebLogic Server 8.1 Web Services Stack Has Been Removed" Section A.6.2, "Universal Description and Discover (UDDI) Registry Has Been Removed" Section A.7.1.1, "Certicom SSL Implementation" Section A.8.1, "Coherence Version" Section A.8.2, "Deprecated and Obsolete Web Application Features" Section A.9, "Evaluation Database Changed From PointBase to Derby" Section A.10.1, "Data Source Profile Logging" Section A.10.2, "Session Affinity Policy" Section A.10.3, "Connection Labeling" Section A.10.4, "ONS Debugging" Section A.10.5, "Oracle Type 4 JDBC drivers from DataDirect" Section A.11.1, "Default Message Mode Has Changed"
10.3.3 and 10.3.4	All sections in the above row, plus: Section A.7.1.2, "Modifications to SSLMBean"
10.3.2	All sections in the above rows, plus: Section A.6.3, "New Web Services Features" Section A.7.1.3, "Introduction of JSSE" Section A.7.2, "Performance Enhancements for Security Policy Deployment" Section A.8.3, "ActiveCache" Section A.8.4, "Class Caching" Section A.10.6, "Deprecated JDBC Drivers" Section A.12.1, "Changes to weblogic.jms.extension API" Section A.12.2, "Persistent Store Updates"
10.3.1	All sections in the above rows, plus: Section A.7.3, "Oracle Internet Directory and Oracle Virtual Directory Authentication Providers"
10.3	All sections in the above rows, plus: Section A.10.7, "capacityIncrement Attribute" Section A.13, "Middleware Home Directory" Section A.14.1, "Resource Registration Name"

Table A–1 (Cont.) Sections That Apply to Upgrades From Each WebLogic Server

WebLogic Server Version	Refer to These Sections
10.0	All sections in the above rows, plus: Section A.7.4, "SAML 2.0 Providers" Section A.7.5, "RDBMS Security Store" Section A.10.9, "Updated WebLogic-branded Data Direct Drivers" Section A.15.1, "Console Configuration Features" Section A.16, "SNMP MIB Refresh Interval and Server Status Check Interval No Longer Used"
9.2	All sections in the above rows, plus: Section A.7.6, "Windows NT Authentication Provider Deprecated" Section A.8.5, "Backward Compatibility Flags" Section A.17, "JMX 1.2 Implementation"
9.1	All sections in the above rows, plus: Section A.7.7, "SAML V2 Providers" Section A.8.6, "BASIC Authentication with Unsecured Resources" Section A.18, "WebLogic Administration and Configuration Scripts"
9.0	All sections in the above rows, plus: Section A.7.8, "XACML Security Providers" Section A.8.7, "Servlet Path Mapping"

Table A–1 (Cont.) Sections That Apply to Upgrades From Each WebLogic Server

WebLogic Server Version	Refer to These Sections
8.1	All sections in the above rows, plus: Section A.7.9, "Security MBeans" Section A.7.10, "Password Encryption" Section A.7.11, "Security for HTTP Requests" Section A.7.12, "Secure Access to MBeanHome" Section A.7.13, "Message-Level Security in Web Services" Section A.11.2, "Modular Configuration and Deployment of JMS Resources" Section A.11.3, "JMS Message ID Format" Section A.11.4, "Improved Message Paging" Section A.14.2, "JTA Transaction Log Migration" Section A.15.2, "Administration Console Extension Architecture" Section A.19, "Dynamic Configuration Management" Section A.20, "Modular Configuration and Deployment of JDBC Resources" Section A.21, "Thread Management" Section A.22, "XML Implementation" Section A.23, "XMLBeans and XQuery Implementation" Section A.24, "Deployment Descriptor Validation and Conversion" Section A.25, "Deprecated Startup and Shutdown Classes" Section A.26, "Resource Adapters" Section A.27, "WLEC"

A.1 Deprecated Functionality

Information about deprecated functionality for WebLogic Server can be found on My Oracle Support at `https://support.oracle.com/`.

- Information about deprecated functionality for WebLogic Server 11*g* Release 1 can be found on My Oracle Support at `https://support.oracle.com/`.

  In the Search Knowledge Base field, enter document ID `888028.1`.

- Information about functionality that is deprecated in WebLogic Server 12.1.1 can be found on My Oracle Support at `https://support.oracle.com/`. Search for **Deprecated Features**.

A.2 Backward Compatibility Flags

The configuration flags in Table A–2 are available to support backward compatibility when you upgrade a domain. By default, these flags are set to support backward compatibility, unless you disable them by selecting the **Do not set backward compatibility flags** option during an upgrade, as described in Section 5.3.1, "Upgrading a Domain in Graphical Mode."

Table A–2 Backward Compatibility Flags

Category	Backward Compatibility Flag	For more information, see ...
Security	■ `EnforceStrictURLPattern`—Specifies whether the server should enforce strict adherence of URL patterns to the Servlet 2.4 specification. During an upgrade, this flag is set to `false` for backward compatibility. ■ `WebAppFilesCaseInsensitive`—Specifies whether the URL-pattern matching behavior is case-insensitive for security constraints, servlets, filters, virtual hosts, and so on, in the Webapp container and external security policies. During an upgrade, this flag is set to `os`, which sets URL-pattern matching as case-sensitive on all platforms except Windows, for compatibility with pre-9.0 releases. As of WebLogic Server 9.0, URL-pattern matching is strictly enforced across operating systems	"SecurityConfigurationMBean"
Web Application	■ `backward-compatible`—Specifies which JSP version is supported. Depending on the version of the Web application (version 2.4 or 2.5) and the setting of the backward-compatible element, WebLogic Server 12.1.1 supports JSP 2.1 or JSP 2.0. `backward-compatible` is located within the "jsp-descriptor" element, as described in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*. ■ `AllowAllRoles`—Specifies that a wildcard (*) character can be used to set the role name to enable all users/roles in the *realm* to have access to the resource collection. As of WebLogic Server 9.0, if you specify the wildcard (*) character as the role name, all users/roles in the *Web application* have access to the resource collection. ■ `FilterDispatchedRequestsEnabled`—Specifies whether to apply filters to dispatched requests. As of WebLogic Server 9.0, the new Dispatcher element makes this behavior explicit. ■ `JSPCompilerBackwardsCompatible`—Specifies whether to allow JSPs that do not comply with the JSP 2.0 specification. ■ `ReloginEnabled`—Specifies whether to allow users multiple attempts to log in to a Web page if the original credentials were not supported. As of WebLogic Server 9.0, the FORM/BASIC authentication behavior has been modified to return the 403 (FORBIDDEN) Web page. ■ `RtexprvalueJspParamName`—Specifies whether to allow run-time expression values for the JSP `<param name>` tag. As of WebLogic Server 9.0, the JSP Compiler no longer allows run-time expression values.	"WebAppContainerMBean"

A.3 JVM Settings

When upgrading a WebLogic Server 11*g* or earlier domain to a WebLogic Server 12.1.1 domain, you may have to:

- manually set the location of the Java endorsed directory (*`JRE_HOME`*`/lib/endorsed`) or directories.
- manually increase the permgen space and maximum permgen space

A.3.1 Setting the Location of the Java Endorsed Directory

In the following situations, you *do not* need to manually set the location of the Java endorsed directory or directories:

- you are using JDK7.
- you are using one of the JDKs that is installed with WebLogic Server 12.1.1.
- you are using WLS 12*c* domains and start scripts that were generated by domain creation via the WebLogic Server 12*c* Configuration Wizard, or your start scripts reference `commEnv.cmd/sh` as installed by the WebLogic Server installer, or both.

If none of these situations apply, and any one of the following situations apply, you must manually set the location of the Java endorsed directory in the command you use to start your Managed Servers:

- you are using Node Manager to start your Managed Servers, but you are not using a start script, that is `startScriptEnabled=false`. Note that as of WebLogic Server 12.1.1, the default value for `startScriptEnabled` is `true`.
- you are using custom start scripts, that is, start scripts that are not provided by Oracle.
- you are trying to create an empty domain using `java.weblogic.Server`.

In any of these cases, include the `java.endorsed.dir` parameter in the Managed Server startup command.

```
startWeblogic.sh -Djava.endorsed.dir=WL_HOME/endorsed
```

To specify multiple Java endorsed directories, separate each directory path with a colon (:).

Note: In all of the options described in this section, you must replace *WL_HOME* with the full path to your WebLogic Server installation.

You can also specify this value when calling `startServer` by passing the values as `jvmArgs`, or when calling `nmstart` by passing them as properties, such as:

```
wls:/nm/mydomain> prps =
makePropertiesObject("Arguments=-Djava.endorsed.dirs=/WL_
HOME/endorsed")

wls:/nm/mydomain> nmStart("AdminServer",props=prps)
```

If you are using Node Manager to start the Managed Server, you can include the `-Djava.endorsed.dirs=/WL_HOME/endorsed")` parameter in the ServerStartMBean's `arguments` attribute, either using WLST or the Administration Console. If using the Administration Console, enter this parameter in the **Arguments** field on the server's **Configuration > Server Start** tab. This attribute will be applied when you call `start(server_name 'Server')` from a WLST client that is connected to the Administration Server or when you click on the **Start** button for the server in the Administration Console.

A.3.2 Setting permgen space

If you receive an `OutOfMemory: PermGen Space` error when starting a Managed Server, you have to manually set the permgen space to at least 128M and increase the maximum permgen space to at least 256M.

Note: In all of the options described here, you must replace *WL_HOME* with the full path to your WebLogic Server installation.

This can be done by specifying the following in the ServerStartMBean's `arguments` attribute, either using WLST or the Administration Console. If using the Administration Console, enter this parameter in the **Arguments** field on the server's **Configuration > Server Start** tab. This attribute will be applied when you call `start(server_name 'Server')` from a WLST client that is connected to the Administration Server or when you click on the **Start** button for the server in the Administration Console.

```
startWeblogic.sh -XX:PermSize=128m -XX:MaxPermSize=256m
```

You can also specify these values when calling `startServer` by passing the values as `jvmArgs`, or when calling `nmstart` by passing them as properties, such as:

```
wls:/nm/mydomain> prps = makePropertiesObject("Arguments=
-XX:PermSize=128m -XX:MaxPermSize=256m")

wls:/nm/mydomain> nmStart("AdminServer",props=prps)
```

A.4 Node Manager startScriptEnabled Default

As of WebLogic Server 12.1.1, the default value for `startScriptEnabled` has been changed to `true`. In all previous releases, the default was `false`. If you do not want to use a start script with Node Manager, change this value to `false` after upgrading.

A.5 Enterprise Java Beans (EJBs)

Oracle Kodo has been deprecated as of WebLogic Server 10.3.1. As of WebLogic Server 12.1.1, EclipseLink is the default JPA provider, replacing Kodo. Applications that continue to use Kodo as the persistence provider with WebLogic Server 12.1.1 will need to be updated. For more information, see "Updating Applications to Overcome Conflicts" in Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server.

As of WebLogic Server 12.1.1, support for JPA 2.0 is built in. JPA 2.0 includes improvements and enhancements to domain modeling, object/relational mapping, `EntityManager` and `Query` interfaces, and the Java Persistence Query Language (JPQL), and more. For more information, see "Using JPA 2.0 with TopLink in WebLogic Server" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

As of WebLogic Server 10.3.2, you can leverage the features of Oracle TopLink with your EJB 3.0 applications. Oracle TopLink is an advanced, object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts, and increase enterprise application functionality. For more information, see *Oracle Fusion Middleware Developer's Guide for Oracle TopLink*.

A.6 Web Services

This section describes Web services features that have been removed from WebLogic Server 12.1.1, as well as new Web services features that were added in WebLogic Server 10.3.3 or greater.

Note: See also Section A.7.13, "Message-Level Security in Web Services."

A.6.1 WebLogic Server 8.1 Web Services Stack Has Been Removed

The WebLogic Server 8.1 Web services stack has been removed in the WebLogic Server 12.1.1 release. Therefore, WebLogic Server 8.1 Web services applications will no longer work. Oracle recommends that you upgrade such applications to the WebLogic JAX-RPC or JAX-WS stacks, per the instructions in "Upgrading an 8.1 WebLogic Web Service to 12.1.x" on page 7-4.

A.6.2 Universal Description and Discover (UDDI) Registry Has Been Removed

The Universal Description and Discovery (UDDI) registry has been removed as of WebLogic Server 12.1.1. If you are still using UDDI and want to upgrade to WebLogic Server 12.1.1, Oracle recommends that you migrate to the Oracle Service Registry (OSR), which is UDDI 3.0 compliant.

A.6.3 New Web Services Features

The following new features have been added in WebLogic Server as of release 10.3.3:

- Support for Web services atomic transactions—WebLogic Web services enable interoperability with other external transaction processing systems, such as WebSphere, JBoss, Microsoft .NET.
- Enhanced support for Web services in a clustered environment
- Enhanced monitoring of Web services and clients
- Attachment of Oracle WSM policies to WebLogic Web services using Fusion Middleware Control
- EclipseLink DBWS support for declarative Web service solution for accessing relational databases
- Method-Level policy attachment behavior change—Before WebLogic Server 10.3.3, if a policy was attached, through the Administration Console, to a method of one Web service, the policy was also attached to all methods of the same name for all Web services in that module. As of WebLogic Server 10.3.3, the policy is attached only to the method of the appropriate Web service.
- `policy:` prefix now removed from OWSM policy names
- Web services WSDL tab now removed—Before WebLogic Server 10.3.3, you could view the WSDL for the current Web service by selecting the **Configuration** > **WSDL** tab. The WSDL tab has been removed as of WebLogic Server 10.3.3.
- New development tools—Oracle JDeveloper and Oracle Enterprise Pack for Eclipse (OEPE)
- Integration with Oracle Enterprise Manager Fusion Middleware Control
- Support for Oracle WebLogic Services Manager (WSM) security policies
- Support for WS-SecureConversation 1.3 on JAX-WS and MTOM with WS-Security on JAX-WS

For more information, see "Web Services" in *What's New in Oracle WebLogic Server*.

A.7 Security

This section describes security changes that have been made over various WebLogic Server releases.

A.7.1 SSL Support Changes

This section describes SSL changes that have been made over various WebLogic Server versions.

A.7.1.1 Certicom SSL Implementation

As of WebLogic Server 12.1.1, the Certicom SSL Implementation has been removed. This change may require you to update system properties and debug switches as described in "Configuring SSL" in *Securing Oracle WebLogic Server*.

A.7.1.2 Modifications to SSLMBean

The SSLMBean has been modified as of WebLogic Server 10.3.5 to support additional SSL configuration capabilities, including the ability to enable or disable the JSSE adapter.

For more information, see the following documents:

- For a list of the differences in the way the JSSE SSL implementation handles the WebLogic system properties, see "System Property Differences Between the JSSE and Certicom SSL Implementations" in *Securing Oracle WebLogic Server*.
- For more information about SSL support in WebLogic Server, see "Secure Sockets Layer (SSL)" in *Understanding Security for Oracle WebLogic Server*.
- For more information on JSEE, see "Java Secure Socket Extension (JSEE) Reference Guide" at `http://download.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html`.

A.7.1.3 Introduction of JSSE

As of WebLogic Server 10.3.3, Java Secure Socket Extension (JSSE) was introduced as an SSL implementation. JSSE is the Java standard framework for SSL and TLS and includes both blocking-I/O and non-blocking-I/O APIs, and a reference implementation including several commonly-trusted CAs.

A.7.2 Performance Enhancements for Security Policy Deployment

As of release 10.3.3, WebLogic Server includes a deployment performance enhancement for Deployable Authorization providers and Role Mapping providers that are thread safe. WebLogic Server by default supports thread-safe parallel modification to security policy and roles during application and module deployment. For this reason, deployable Authorization and Role Mapping providers configured in the security realm should support parallel calls. The WebLogic deployable XACML Authorization and Role Mapping providers meet this requirement.

However, if your custom deployable Authorization or Role Mapping providers do not support parallel calls, you must disable the parallel security policy and role modification and instead enforce a synchronization mechanism that results in each application and module being placed in a queue and deployed sequentially. You can turn on this synchronization enforcement mechanism from the Administration

Console or by using the `DeployableProviderSynchronizationEnabled` and `DeployableProviderSynchronizationTimeout` attributes of the RealmMBean.

See "Enabling Synchronization in Security Policy and Role Modification at Deployment" in *Securing Oracle WebLogic Server* for additional information.

A.7.3 Oracle Internet Directory and Oracle Virtual Directory Authentication Providers

Two new LDAP authentication providers were added to WebLogic Server 10.3.2: the Oracle Internet Directory Authentication Provider and the Oracle Virtual Directory Authentication Provider. These authentication providers can store users and groups in, and read users and groups from, the Oracle Internet Directory and Oracle Virtual Directory LDAP servers, respectively.

For information about configuring and using these new security providers, see "Configuring LDAP Authentication Providers" in *Securing Oracle WebLogic Server*.

A.7.4 SAML 2.0 Providers

For SAML 2.0 support, the SAML 2.0 Credential Mapping provider and SAML 2.0 Identity Assertion provider were added in WebLogic Server 10.3. These new providers can be used, respectively, to generate and consume SAML 2.0 assertions in the following use cases:

- SAML 2.0 Web Single Sign-On Profile
- WS-Security SAML Token Profile version 1.1

For SAML 2.0 Web Single Sign-On, the assertions generated by the SAML 2.0 Credential Mapping provider may be consumed only by the SAML 2.0 Identity Assertion provider. They are not compatible with SAML 1.1 assertions.

SAML Token Profile 1.1 is supported by WebLogic Server Web Services as of Release 10.3. This feature includes support for SAML 2.0 and SAML 1.1 assertions, and is backward compatible with SAML Token Profile 1.0 SAML tokens are configured for a Web Service through use of the appropriate WS-SecurityPolicy assertions.

Note: SAML Token Profile 1.1 is supported only through WS-SecurityPolicy. The earlier "WebLogic Server 9.2 Security Policy" supports SAML Token Profile 1.0/SAML 1.1 only.

A.7.5 RDBMS Security Store

WebLogic Server 10.3 added the option of using an external RDBMS as a data store that is used by the following security providers:

- XACML Authorization provider
- XACML Role Mapping provider
- The following providers for SAML 1.1:
  - SAML Identity Assertion provider V2
  - SAML Credential Mapping provider V2
- The following providers for SAML 2.0:
  - SAML 2.0 Identity Assertion provider
  - SAML 2.0 Credential Mapping provider

- WebLogic Credential Mapping provider
- PKI Credential Mapping provider
- Certificate Registry

This data store, called the RDBMS security store, is strongly recommended for the use of SAML 2.0 services in two or more WebLogic Server instances in that domain, such as in a cluster. When the RDBMS security store is configured in a domain, an instance of any of the preceding security providers that has been created in the security realm automatically uses only the RDBMS security store as a data store, and not the embedded LDAP server. WebLogic security providers configured in the domain that are not among those in the preceding list continue to use their respective default stores; for example, the WebLogic Authentication provider continues to use the embedded LDAP server.

In order to use the RDBMS security store, the preferred approach is first to create a domain in which the external RDBMS server is configured. Before you boot the domain, create the tables in the data store that are required by the RDBMS security store. The WebLogic Server installation directory contains a set of SQL scripts that create these tables for each supported database.

If you have an existing domain in which you want to use the RDBMS security store, you should create the new domain, then migrate your security realm to it. Oracle does not recommend "retrofitting" the RDBMS security store to an existing domain. For more information, see "Managing the RDBMS Security Store" in *Securing Oracle WebLogic Server*.

A.7.6 Windows NT Authentication Provider Deprecated

The Windows NT Authentication provider is deprecated as of WebLogic Server 10.0. Use one or more of the other supported Authentication providers instead.

A.7.7 SAML V2 Providers

For SAML 1.1 support, new versions of the SAML Credential Mapping provider and SAML Identity Assertion provider were added in WebLogic Server 9.2. The SAML Credential Mapping V1 provider and SAML Identity Assertion V1 provider are deprecated; you should use the V2 versions of the SAML Credential Mapping and SAML Identity Assertion providers.

Although the version number of the providers has been incremented to V2, the new SAML security providers implement the SAML 1.1 standard, as did the V1 providers.

Note: For web single sign-on, the SAML 1.1 providers described in this section are not compatible with a WebLogic Server instance that has been configured with SAML 2.0 services.

A.7.8 XACML Security Providers

As of 9.1, WebLogic Server includes two new security providers, the XACML Authorization provider and the XACML Role Mapping provider. Previous releases of WebLogic Server used an authorization provider and a role mapping provider based on a proprietary security policy language. These XACML security providers support the eXtensible Access Control Markup Language (XACML) 2.0 standard from OASIS. These providers can import, export, persist, and execute policy expressed using all standard XACML 2.0 functions, attributes, and schema elements.

WebLogic domains created using WebLogic Server 9.1 and later include the XACML providers by default. The new XACML providers are fully compatible with policies and roles created using the WebLogic Authorization provider (DefaultAuthorizer) and WebLogic Role Mapping provider (DefaultRoleMapper). Existing WebLogic domains that you upgrade to 12.1.1 continue to use the authorization and role mapping providers currently specified, such as third-party partner providers or the original WebLogic Authorization and Role Mapping providers. If you want, you can migrate existing domains from using WebLogic Server proprietary providers to the XACML providers, including performing bulk imports of existing policies. For more information, see "Understanding WebLogic Server Security" in *Understanding Oracle WebLogic Server*.

A.7.9 Security MBeans

Table A–3 lists the changes to security MBeans as of WebLogic Server 9.0.

Table A–3 Changes to Security MBeans as of WebLogic Server 9.0

Type of Security MBean	Description
All security MBeans	In WebLogic Server 8.1, when you updated a security MBean attribute, the values were available to the security configuration and management hierarchy immediately, and to the security run-time hierarchy following a server reboot. As of WebLogic Server 9.0, whether a security MBean attribute change is effective and available to the configuration, management, and run-time hierarchies immediately or upon server reboot is controlled by setting that attribute as dynamic or non-dynamic. For more information, see Section A.19, "Dynamic Configuration Management."

`RealmMBean`, `UserLockoutManagerMBean`, and all security provider MBeans	▪ The `wls_getDisplay` method is deprecated. In its place, you should use the new `getName` method. In addition, the following security methods have been removed:
	
	`wls_getAttributeTag`
	`wls_getConstructorTag`
	`wls_getMBeanTag`
	`wls_getNotificationTag`
	`wls_getOperationTag`
	
	▪ The `weblogic.Admin` tools and pre-9.2 JMX security APIs can no longer be used to configure security MBeans. These utilities and APIs can still be used, however, to view and invoke methods on the security MBeans.
	
	For security provider MBeans (only):
	
	▪ When adding or removing a security provider, you must reboot the server before any changes take effect.
	
	▪ When modifying an existing security provider, if you modify any non-dynamic attributes, the server must be rebooted before *any* (that is, non-dynamic or dynamic) changes take effect. For more information, see Section A.19, "Dynamic Configuration Management."

Table A–3 (Cont.) Changes to Security MBeans as of WebLogic Server 9.0

Type of Security MBean	Description

All *custom* security provider MBeans	▪ By default, all custom security provider MBeans attributes are non-dynamic. For more information, see Section A.19, "Dynamic Configuration Management."
	▪ You can set an MBean attribute to be dynamic by setting `Dynamic="true"` for the attribute within the MDF file. For example:
	`<MBeanAttribute`
	`Name = "Foo"`
	`Type = "java.lang.String"`
	`Dynamic = "true"`
	`Description = "Specifies that this attribute is a dummy."`
	`/>`

A.7.10 Password Encryption

To prevent unauthorized access to sensitive data such as passwords, some attributes in configuration MBeans are encrypted. The attributes persist their values in the domain configuration files as an encrypted string. For further security, the in-memory value is stored in the form of an encrypted byte array to help reduce the risk of the password being snooped from memory.

In pre-9.0 releases, you could edit the config.xml file to specify an encrypted attribute, such as a password, in clear-text or encrypted format. In this case, when booted, the WebLogic Server encrypts the information the next time it writes to the file.

As of WebLogic Server 9.0, when operating in production mode, the password of an encrypted attribute must be encrypted in the configuration files. In development mode, the password of an encrypted attribute can be either encrypted or clear-text.

You can use the `weblogic.security.Encrypt` command-line utility to encrypt the passwords, as follows:

```
java weblogic.security.Encrypt
```

You are prompted to enter a password, and the command returns the encrypted version. Then, copy the encrypted password returned into the appropriate file.

This utility is not just used for passwords in the configuration files. It can also be used to encrypt passwords in descriptor files (for example, a JDBC or JMS descriptor) and in deployment plans. For more information, see "encrypt" in *Command Reference for Oracle WebLogic Server*.

A.7.11 Security for HTTP Requests

By default, when an instance of WebLogic Server responds to an HTTP request, its HTTP response header does not include the WebLogic Server name and version number. This behavior is different from releases before WebLogic Server 9.0.

To have the name and version number included in the HTTP response header when responding to an HTTP request, enable the Send Server Header attribute for the WebLogic Server instance in the Administration Console. The attribute is located on the **Server** > *ServerName* > **Protocols** > **HTTP** tab under the Advanced Options section. Note that enabling this feature may creates a security risk if a possible attacker knows about a vulnerability in the specified version of WebLogic Server.

For more information about ensuring security, see "Securing the WebLogic Security Service" in "Ensuring the Security of Your Production Environment" in *Securing a Production Environment for Oracle WebLogic Server*.

A.7.12 Secure Access to MBeanHome

In pre-9.0 releases of WebLogic Server, anonymous access to `MBeanHome` was enabled by default. With the security enhancements delivered as of WebLogic Server 9.0, anonymous access to `MBeanHome` is no longer allowed.

Although doing so is not recommended, you can re-enable anonymous access by specifying the following flag when starting the server:

```
-Dweblogic.management.anonymousAdminLookupEnabled
```

A.7.13 Message-Level Security in Web Services

As of WebLogic Server 9.0, message-level security in Web Services was enhanced to use the standards-based Web Services Policy Framework (WS-Policy). WS-Policy provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web Services-based system. For more information about WS-Policy, see "Using WS-SecurityPolicy 1.2 Policy Files" in *Security and Administrator's Guide for Web Services*.

In 8.1, the implementation was based on an OASIS implementation of the Web Services Security (WSS) standard. This implementation is supported for backward compatibility, but is deprecated as of 9.0. For more information, see `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

A.8 Web Applications, JSPs, and Servlets

The following sections provide important compatibility information for Web applications, JSPs, and Servlets in WebLogic Server 12.1.1:

- ActiveCache
- Coherence Version
- Class Caching
- Deprecated and Obsolete Web Application Features
- BASIC Authentication with Unsecured Resources
- Backward Compatibility Flags
- Servlet Path Mapping

A.8.1 Coherence Version

The WebLogic Server 12.1.1 installer includes Coherence 3.7.1. All servers in a cluster must use the same version of Coherence. Therefore, all cache servers in the cluster must be upgraded to Coherence 3.7.1.

A.8.2 Deprecated and Obsolete Web Application Features

For a list of Web application features that are deprecated or are not supported as of WebLogic Server 12.1.1, refer to the following:

- Information about deprecated functionality for WebLogic Server 11*g* Release 1 can be found on My Oracle Support at `https://support.oracle.com/`.

In the Search Knowledge Base field, enter document ID `888028.1`.

- Information about functionality that is deprecated in WebLogic Server 12.1.1 can be found on My Oracle Support at `https://support.oracle.com/`. Search for **Deprecated Features**.

A.8.3 ActiveCache

As of WebLogic Server 10.3.3, applications deployed on WebLogic Server can easily use Coherence data caches, and seamlessly incorporate Coherence*Web for session management and TopLink Grid as an object-to-relational persistence framework. Collectively, these features are called ActiveCache.

ActiveCache provides replicated and distributed data management and caching services that you can use to reliably make an application's objects and data available to all servers in a Coherence cluster.

For more information, see *Using ActiveCache*.

A.8.4 Class Caching

As of release 10.3.3, WebLogic Server allows you to enable class caching. The advantages of using class caching are:

- Reduces server startup time.
- The package level index reduces search time for all classes and resources.

Class caching is supported in development mode when starting the server using a startWebLogic script. Class caching is disabled by default and is not supported in production mode. The decrease in startup time varies among different JRE vendors. For more information, see "Configuring Class Caching" in *Developing Applications for Oracle WebLogic Server*.

A.8.5 Backward Compatibility Flags

For WebLogic Server 10.0 and later, backward compatibility for WebLogic Server 9.2 or earlier is supported by using the `backward-compatible` element within the `jsp-descriptor` element, as described in this section and in "jsp-descriptor" in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

A.8.5.1 JSP 2.1 Support and Compatibility With JSP 2.0 Web Applications

JSP 2.1 is supported as of WebLogic Server 10.0. Depending on the version of the Web application (version 2.4 or 2.5) and the setting of the `backward-compatible` element, WebLogic Server 10.0 and later also supports JSP 2.0.

See "Backward Compatibility Flags" in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server* for important information about the buffer suffix setting and implicit servlet 2.5 package imports.

A.8.5.2 Support for JSP 2.0

JSP 2.0 was supported as of WebLogic Server 9.0, and continues to be supported as described in Section A.8.5, "Backward Compatibility Flags." Please note the following changes to the JSP behavior as required in support of JSP 2.0:

- If a JSP does not participate in a session (or if the session in which a JSP participates is invalid), an `IllegalStateException` is thrown when the following command is executed:

```
PageContext.getAttribute(name, PageContext.SESSION_SCOPE)
```

If you are not concerned about this type of error, you can catch and ignore it.

- `JspWriterImpl` now replaces `\n` with `System.getProperty("line.separator")` for each `printline` function. This replacement causes problems with JSPs that:
  - Contain multiple page directives that appear on new lines. For example:

    ```
    <%@ page import="com.foo.bar.*" %>
    <%@ page import="com.foo.xyz.*" %>
    ...
    ```

  - Generate output in XML format.
  - Generate an XML declaration following the page directives.
  - Are served by Windows systems. In this case, `\r\n` is output for each page directive.
  - Are viewed using Internet Explorer.

    When viewed in Internet Explorer, each page directive outputs an empty `\r\n` and the XML declaration (`<?xml version="1.0" encoding="iso-8859-1"?>`) appears after every new line. Internet Explorer displays an error message indicating that it cannot locate the declaration and that the page cannot be viewed, even though it can be compiled.

  To work around any issues caused by changes to `JspWriterImpl`, you can perform one or both of the following tasks:

  - Define the XML declaration at the top of the page.
  - Group the page directives into a single declaration, for example:

    ```
    <%@ page import="com.foo.bar.*, com.foo.baz.*"
    contentType="text/html" pageEncoding="UTF-8" errorPage="Error.jsp" %>
    ```

- The JSP `<param name>` tag no longer allows run-time expression values. For example:

  ```
  <jsp:param name="<%= AdminActions.RETURN_LINK %>" value="<%= returnlink %>" />
  ```

  You can continue to support this feature by disabling the **Do not set backward compatibility flags** upgrade option during the domain upgrade, as described in the "Select Upgrade Options" row of Table 5–1, or enabling the `backwardCompatible` flag in the weblogic.xml file, as follows:

  ```
  <jsp-descriptor>
  <jsp-param>
  <param-name>backwardCompatible</param-name>
  <param-value>true</param-value>
  </jsp-param>
  </jsp-descriptor>
  ```

A.8.6 BASIC Authentication with Unsecured Resources

For WebLogic Server versions 9.2 and later, client requests that use HTTP BASIC authentication must pass WebLogic Server authentication, even if access control is not enabled on the target resource.

The setting of the Security Configuration MBean flag "`enforce-valid-basic-auth-credentials`" determines this behavior. (The

DomainMBean can return the new Security Configuration MBean for the domain.) It specifies whether the system should allow requests with invalid HTTP BASIC authentication credentials to access unsecured resources.

> **Note:** The Security Configuration MBean provides domain-wide security configuration information. The `enforce-valid-basic-auth-credentials` flag effects the entire domain.

The `enforce-valid-basic-auth-credentials` flag is true by default, and WebLogic Server authentication is performed. If authentication fails, the request is rejected. WebLogic Server must therefore have knowledge of the user and password.

See "Understanding BASIC Authentication with Unsecured Resources" in *Programming Security for Oracle WebLogic Server* for complete information.

A.8.7 Servlet Path Mapping

As of version 2.3 of the Java Servlet Specification, the following syntax is used to define mappings:

- A servlet path string that contains only the `/` (slash) character indicates the default servlet of the application. The servlet path resolves to the request URI minus the context path; in this case, the path resolves to `null`.
- A String that begins with an `*` (asterisk) specifies an extension mapping.

These changes introduce a change in behavior with the following `HttpServletRequest` methods:

- `getPathInfo`
- `getServletPath`

To better illustrate the change in behavior, consider the request `/abc/def.html` that resolves to ServletA:

- If `/` maps to ServletA, then `servletPath="abc/def.html"` and `pathInfo=null`.
- If `/*` maps to ServletA, then `servletPath=""` and `pathInfo="abc/def.html"`.

To ensure that the path info returned is non-null, replace all occurrences of the `/` (slash) servlet mapping string with `/*`.

The Java Servlet Specification can be downloaded from the following location:

`http://www.oracle.com/technetwork/java/javaee/servlet/index.html`

A.9 Evaluation Database Changed From PointBase to Derby

As of WebLogic Server 10.3.3, the evaluation database available from the WebLogic Server installation program has been changed from PointBase to Apache Derby. If you select the **Evaluation Database** option on the Choose Products and Components screen, the Derby database is installed in the *`WL_HOME`*`\common\derby` directory. If you select a Typical installation, Derby is installed by default.

If you have a domain based on PointBase and you want to continue using PointBase after upgrading the domain to WebLogic Server 10.3.3 or later, you must obtain a

PointBase license from http://www.pointbase.com. Note that the full WLS installer does not preserve the PointBase installation directory. As an alternative to using PointBase, you can migrate the domain database to Derby.

For more information, see Section 5.4, "Upgrading a Domain that Uses an Evaluation Database."

A.10 JDBC Feature Changes

The following sections describe changes to JDBC support:

- Data Source Profile Logging
- Session Affinity Policy
- Connection Labeling
- ONS Debugging
- Oracle Type 4 JDBC drivers from DataDirect
- Deprecated JDBC Drivers
- capacityIncrement Attribute
- JDBC 4.0 Support
- Updated WebLogic-branded Data Direct Drivers

For details about new JDBC features, see "JDBC" in *What's New in Oracle WebLogic Server*.

A.10.1 Data Source Profile Logging

To provide better usability and performance, WebLogic Server 12.1.1 and higher uses a data source profile log to store events. See "Monitoring WebLogic JDBC Resources" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

A.10.2 Session Affinity Policy

In WebLogic Server 12.1.1, GridLink data sources use the session affinity policy to improve performance by directing the database operations of a servlet session to the same RAC instance in a RAC cluster. See "GridLink Affinity" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

A.10.3 Connection Labeling

In WebLogic Server 12.1.1, labeling allows an application to attach arbitrary name/value pairs (labels) to a connection that has a particular initialization state. This allows the application to improve performance by minimizing the time and cost of re-initializing a connection. See "Labeling Connections" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

A.10.4 ONS Debugging

In WebLogic Server 12.1.1, UCP and ONS are no longer included in your installation. For information on how to set UPC and ONS debugging, see "Setting Debugging for UCP/ONS" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

A.10.5 Oracle Type 4 JDBC drivers from DataDirect

As of WebLogic Server 12.1.1, Oracle Type 4 JDBC drivers from DataDirect are referred to as WebLogic-branded DataDirect drivers. Oracle has retired the documentation in *Type 4 JDBC Drivers for Oracle WebLogic Server* and no longer provides detailed information on DataDirect drivers. Oracle continues to provide information on how WebLogic-branded drivers are configured and used in WebLogic Server environments at "Using WebLogic-branded DataDirect Drivers" in *Programming JDBC for Oracle WebLogic Server*. Oracle recommends reviewing DataDirect documentation for detailed information on driver behavior, see "Progress DataDirect for JDBC User's Guide Release 4.2" and "Progress DataDirect for JDBC Reference Release 4.2" at `http://www.datadirect.com/index.html`.

A.10.6 Deprecated JDBC Drivers

The following JDBC drivers are deprecated:

- WebLogic Type 4 JDBC driver for Oracle

  This driver was deprecated in WebLogic Server 10.3 and is now removed. Instead of using this deprecated driver, you should use the Oracle Thin Driver that is provided with WebLogic Server. For details about the Oracle Thin Driver, see "Using Third-Party JDBC Drivers with WebLogic Server" in *Type 4 JDBC Drivers for Oracle WebLogic Server*.

- The Sybase JConnect 5.5 and 6.0 drivers 5.5 and 6.0 are removed from WebLogic Server as of release 10.3.3 due to an Oracle security policy regarding default installation of code samples. You can download the driver from Sybase or you can use the Oracle-branded JDBC driver for Sybase that is packaged with WebLogic Server.

A.10.7 capacityIncrement Attribute

In WebLogic Server 10.3.1 and higher releases, the `capacityIncrement` attribute is no longer configurable and is set to a value of 1.

A.10.8 JDBC 4.0 Support

As of 10.3, WebLogic Server is compliant with the JDBC 4.0 specification, with the following enhancements and exceptions:

- SQLXML is fully supported on the server side. On the RMI client side, SQLXML is partially supported. You cannot use the getSource and setResult APIs on the client side.
- WebLogic Server JDBC supports standard Wrapper Pattern functionality and extends the functionality on the server side. The JDBC standard requires support for the Wrapper operation on the interface. WebLogic Server supports the Wrapper operation on both the interface and on the concrete class on the server side.
- WebLogic Server enhances statement pool management as follows.
  - For the `Statement` interface:
    * Call `isPoolable()` always returns false
    * Call `setPoolable()` does not change the poolable state.
  - For the `PreparedStatement` and `CallableStatement` interfaces:

        * Call `isPoolable()` returns the current poolable state, the default value is `true`
        * Call `setPoolable()` modifies the poolable state.
      - For the `PreparedStatement` or `CallableStatement` interface, the following occurs when you call the `close()` method:
        * If the current poolable state is `false`, `PreparedStatement` or `CallableStatement` is closed.
        * If the current poolable state is `true`, `PreparedStatement` or `CallableStatement` reverts to the statement cache.
  - Updated third-party JDBC drivers:
    - Oracle Thin driver updated from 10*g* to 11*g*
    - PointBase database server and driver updated from 5.1 to 5.7

A.10.9 Updated WebLogic-branded Data Direct Drivers

The WebLogic-branded Data Direct JDBC drivers included with WebLogic Server are provided from DataDirect. As of 10.3, the drivers were updated to DataDirect Version 3.7. For information about changes to these drivers, including to support JDBC 4.0, see "Using WebLogic-branded DataDirect Drivers" in the *Programming JDBC for Oracle WebLogic Server*.

A.11 JMS

Note the following changes to JMS:

- Default Message Mode Has Changed
- Modular Configuration and Deployment of JMS Resources
- JMS Message ID Format
- Improved Message Paging

A.11.1 Default Message Mode Has Changed

As of WebLogic Server 12.1.1, the default messaging mode has been changed from multicast to unicast.

A.11.2 Modular Configuration and Deployment of JMS Resources

As of WebLogic Server 9.0, JMS configurations are stored as modules, defined by XML documents that conform to the new weblogic-jmsmd.xsd schema. With modular deployment of JMS resources, you can promote your application and the JMS configuration from one environment to another. For example, you can promote your application and the required JMS configuration from a testing environment to a production environment, without opening an EAR file and without extensive manual JMS reconfiguration.

For more information, see:

- "New and Changed JMS Features in This Release" in *Configuring and Managing WebLogic JMS* (published for WebLogic Server 9.0, available at `http://download.oracle.com/docs/cd/E13222_01/wls/docs90/jms_admin/intro.html#WhatsNewJMS`).

- "Understanding JMS Resource Configuration" in *Configuring and Managing JMS for Oracle WebLogic Server*.
- "Deploying JDBC, JMS, and WLDF Application Modules" in *Deploying Applications to Oracle WebLogic Server*.

The WebLogic Upgrade Wizard automatically converts pre-9.0 JMS resources to a JMS Interop module file named interop-jms.xml, which is copied to the domain's config\jms directory. For more information, see "JMS Interop Modules" in *Configuring and Managing JMS for Oracle WebLogic Server*.

Please note the following JMS configuration changes:

- When generating new JMS resources, you must define all attributes in the JMS module (that is, not using the pre-9.0 configuration file).
- The `Allow Persistent Downgrade` option enables you to specify whether JMS clients receive an exception when they send persistent messages to a destination targeted to a JMS server that does not have a persistent store configured. This option is provided for backward compatibility with previous releases.

  By default, the option is set to `false`, specifying that clients receive an exception when they send persistent messages to a JMS server for which no store is configured. When the option is set to `true`, persistent messages are downgraded to non-persistent, but, the send operations are allowed to continue. This parameter is effective only when the `Store Enabled` parameter is disabled (that is, when it is set to `false`).

  For more information, see "AllowsPersistentDowngrade" in "JMSServerBean" in *Oracle WebLogic Server MBean Reference*.
- A `Temporary Template` is created, by default, for JMS Servers. In previous releases, no default template was provided. You can also configure a temporary template, using the JMS server's `Temporary Template` attribute.

  You can control whether the JMS Server can host a temporary destination by setting the `Hosts Temporary Destinations` attribute. In previous releases, a JMS Server was enabled to host temporary destinations if and only if the `TemporaryTemplate` attribute was set.
- JMS templates specified for distributed destinations are no longer supported as of WebLogic Server 9.0, and they are ignored. As of WebLogic Server 9.0, this functionality is replaced by uniform distributed destinations. For more information, see "Creating Uniform Distributed Destinations" in *Configuring and Managing JMS for Oracle WebLogic Server*.
- The `AllowCloseInOnMessage` attribute for JMS Connection Factories is enabled by default. For more information, see `"ClientParamsBean"` in *Oracle WebLogic Server MBean Reference*.
- The `getExpirationLoggingPolicy` attribute in the `DeliveryFailureParamsBean` has been deprecated. Oracle recommends that you update your applications to use the Message Life Cycle Logging feature described in "Message Life Cycle Logging" in *Configuring and Managing JMS for Oracle WebLogic Server*. It should also be noted that the `getExpirationLoggingPolicy` attribute now removes any leading and trailing white space that may have been embedded in an application.

A.11.3 JMS Message ID Format

As of WebLogic Server 9.0, the format of the JMS message ID has changed. Oracle continues to support the pre-9.0 format for existing consumers, producers, and servers. For example, existing JMS consumers may continue to view messages in the pre-9.0 format, even when received from a new JMS producer and JMS server.

A.11.4 Improved Message Paging

As of WebLogic Server 9.0, the message paging feature for freeing up JVM heap space during peak message load situations is always enabled on JMS servers. Additionally administrators are not required to create a dedicated message paging store because paged out messages can be stored in a directory on your file system. However, for the best performance you should specify that messages be paged to a directory other than the one used by the JMS server's persistent store.

See "Paging Out Messages To Free Up Memory" in *Performance and Tuning for Oracle WebLogic Server*.

A.12 Messaging

This section describes messages changes that you should be aware of when upgrading to WebLogic Server 12.1.1.

A.12.1 Changes to weblogic.jms.extension API

As of WebLogic Server 10.3.3, the following internal methods of the `weblogic.jms.extensions.WLMessage` interface have been removed from the *Oracle WebLogic Server API Reference*:

```
public void setSAFSequenceName(String safSequenceName);
public String getSAFSequenceName();
public void setSAFSeqNumber(long seqNumber);
public long getSAFSeqNumber();
```

Your applications should not use these internal methods. Internal methods may change or be removed in a future release without notice.

A.12.2 Persistent Store Updates

As of WebLogic Server 10.3.3, WebLogic File Store behavior and tuning have changed for default file stores and custom file stores.

A.13 Middleware Home Directory

As of WebLogic Server 10.3.1, the notion of the BEA Home directory is replaced by the Middleware Home. The default path of this directory is <*drive:*>Oracle/Middleware. This change has the following impact on WebLogic Server:

- A new environment variable is introduced in several WebLogic scripts in 10.3.1 to represent the Middleware Home directory: *`MW_HOME`*. The directory to which this variable is set generally is the same as *`BEA_HOME`*, which is also still used in WebLogic Server scripts.
- By default, the WebLogic Server installation program selects <*drive:*>`Oracle/Middleware` as the root product installation directory. However, if a directory containing an existing WebLogic Server installation is detected, that directory is selected instead by default.

- The WebLogic Server 10.3.1 documentation now uses the term *Middleware Home*, instead of BEA Home. However, this revision is functionally only a change in terminology and does not imply that any WebLogic software, custom domains, or applications must be moved, or that any existing environment variables that represent those locations must be changed.

This change does not affect any existing WebLogic Server installations, custom domains, applications, or scripts on your computer. You may continue to use the *BEA_HOME* environment variable as before.

A.14 JTA

The following sections describe JTA feature changes:

- Resource Registration Name
- JTA Transaction Log Migration

A.14.1 Resource Registration Name

As of WebLogic Server 10.3.1, the behavior of the resource registration name for XA data source configurations has changed. In previous releases, the JTA registration name was simply the name of the data source. Now, the registration name is a combination of data source name and domain.

For more information, see "Registering an XAResource to Participate in Transactions" in *Programming JTA for Oracle WebLogic Server*.

A.14.2 JTA Transaction Log Migration

All JTA domain configuration options are persisted from the legacy configuration file. The only changes are at the server level. As of WebLogic Server 9.0, the Transaction Manager uses the default WebLogic persistent store to store transaction log records. During the upgrade, the Upgrade Wizard copies transaction log records to the default store. The transaction log file prefix from the existing server configuration is used only to locate the transaction log (`.tlog`) files during an upgrade; it is not preserved after the upgrade.

If the entire domain resides on a single computer, the Upgrade Wizard handles the upgrade (and copies transaction log records to the default store) for all Managed Servers during the initial domain upgrade. If Managed Servers reside on separate machines, you must upgrade each Managed Server individually, as described in Section 2.3, "Upgrade Your Application Environment."

Please note the following:

- When an explicit upgrade is performed (see Section 5.3, "Upgrading a Domain"), transaction recovery does not run during the upgrade process, but it starts running when you start the server(s).
- When an implicit upgrade is performed, as described in Appendix E, "Upgrading a Domain at Administration Server Startup (Implicit Mode)," transaction recovery runs during the server boot process.

If you have put your transaction log files in network storage in preparation for Transaction Recovery Service migration, the log file location is not preserved after the upgrade. In this release, the WebLogic Server Transaction Manager uses the WebLogic default persistent store to store transaction log files. You can achieve the same result by moving the location of the WebLogic default persistent store to a network location.

Note that you must manually copy the `DAT` file from the default location of the current default store to the new location of the default store.

If transactions span multiple domains, you must configure your domain to enable inter-domain transactions. For more information, see "Configuring Domains for Inter-Domain Transactions" in *Programming JTA for Oracle WebLogic Server*.

A.15 Administration Console

The following sections describe changes to the Administration Console:

- Console Configuration Features
- Administration Console Extension Architecture

A.15.1 Console Configuration Features

WebLogic Server 10.3 introduced new options that were added for configuring Console behavior, including the ability to do the following:

- Lock a domain configuration so you can make changes to the configuration while preventing other accounts from making changes during your edit session
- Specify whether to deploy internal applications such as the Administration Console, UDDI, and the UDDI Explorer on demand (upon first access) instead of during server startup
- Locate any WebLogic Server Configuration MBean that contains the string specified in its name, using a new search feature added to the banner toolbar region
- Use additional capabilities for automatically migrating failed servers and services from one server to another
- Deploy and control Service Component Architecture (SCA) deployments
- Inspect Spring applications

For more information, see "What's New in WebLogic Server" in the *WebLogic Server 10.3 Release Notes*.

A.15.2 Administration Console Extension Architecture

In WebLogic Server version 9.0, the Administration Console was built on the WebLogic Portal Framework, which makes it more open and more readily extensible. The architecture necessitated new procedures for extending the Administration Console. WebLogic Server console extensions built for releases of WebLogic Server before 9.0 do not function with the new console infrastructure.

The following features were added in WebLogic Server 10.3.2:

- A sample Look and Feel is provided, which you can modify to create a custom Look and Feel for the WebLogic Server Administration Console.
- Online help can be created and associated with console extensions.

For more information about extending the WebLogic Server Administration Console, see *Extending the Administration Console for Oracle WebLogic Server*.

A.15.2.1 Important Console-Extension Information for Version 9.2

Version 9.2 of WebLogic Server introduced the following changes to console extensions:

- Before this release, Administration Console extensions could import a set of third-party JSP tag libraries by specifying a path name to the tag library file. For example,

  ```
  <%@ taglib uri="/WEB-INF/beehive-netui-tags-template.tld"
  prefix="beehive-template" %>
  ```

  As of 10.0, Administration Console extensions that use these third-party JSP tag libraries from the WebLogic Server installation must use pre-defined, absolute URIs to specify the tag libraries. For example:

  ```
  <%@ taglib uri="http://beehive.apache.org/netui/tags-template-1.0"
  prefix="beehive-template" %>
  ```

  The Administration Console's `web.xml` file maps these URIs to tag libraries within the WebLogic Server installation. This mapping facility enables Oracle to reorganize its installation directory without requiring you to change your JSPs.

  Any Administration Console extensions that use the old path name syntax to import Apache Struts, Apache Beehive, or the JSTL tag libraries must update all path names to the new URIs.

  The URI for the WebLogic Server Console Extension tag library (`console-html.tld`) remains unchanged: `/WEB-INF/console-html.tld`.

  For more information, see "JSP Templates and Tag Libraries" in *Extending the Administration Console for Oracle WebLogic Server*.

- By convention, portal include files (`.pinc`) files are now called portal book files (`.book`).

A.15.2.2 WebLogic Portal Skeleton URI References Should be Fully Qualified

WebLogic Portal requires that any explicit Skeleton URI references be fully qualified relative to the webapp. However, the documentation and some console extension examples have sometimes used relative references to these skeletons. Consider the following incorrect example:

```
<netuix:singleLevelMenu markupType="Menu" markupName="singleLevelMenu"
skeletonUri="singlelevelmenu_children2.jsp"/>
```

This example should have been correctly specified as:

```
<netuix:singleLevelMenu markupType="Menu" markupName="singleLevelMenu"
skeletonUri="/framework/skeletons/default/singlelevelmenu_children2.jsp"/>
```

For this release, relative skeleton URI references continue to work. However, any console extensions that you have written should be updated to use fully qualified skeleton URIs, because these relative references may no longer function correctly in a future release.

A.16 SNMP MIB Refresh Interval and Server Status Check Interval No Longer Used

The `"SNMPAgentMBean"` MBean "MibDataRefreshInterval" and "ServerStatusCheckIntervalFactor" attributes were deprecated in WebLogic Server 10.0 and are ignored.

A.17 JMX 1.2 Implementation

As of WebLogic Server 9.0, WebLogic Server uses the Java Management Extensions (JMX) 1.2 implementation that is included in JDK 5.0. Before 9.0, WebLogic Server used its own JMX implementation based on the JMX 1.0 specification.

The JMX 1.2 reference implementation introduces serialization incompatibilities. Despite these incompatibilities in the reference implementation, JMX clients created for WebLogic Server 8.1 can be used with 9.2 and later releases as follows:

- If your JMX client accesses only WebLogic Server MBeans and uses only `weblogic.management.MBeanHome`, it can be run in a WebLogic Server 9.2 or later instance without being upgraded.
- A JMX client in which WebLogic Server 8.1 classes are used can interact with JMX agents in WebLogic Server 9.2 or later if all of the following are true:
  - The client accesses only WebLogic Server MBeans.
  - The client uses only `weblogic.management.MBeanHome`; it does not use the JDK `MBeanServer` interface.
  - The WebLogic Server classes are from 8.1 SP4 with any appropriate patches applied.
- If the standard JMX MBeanServer interface is used in your JMX client, either to interact with WebLogic Server MBeans or to create and access custom MBeans, you must include the following JDK startup option for the WebLogic Server 9.2 or later instance: `-Djmx.serial.form=1.0`.

  This startup option causes the JVM to use JMX 1.0 class descriptions when it is serializing objects. The option is required when JMX 1.0 clients communicate with JMX 1.2 agents using the standard JDK classes.
- If your JMX client interacts with security provider MBeans, see Section A.7.9, "Security MBeans."

Oracle recommends that you update your JMX clients to be compliant with WebLogic Server 12.1.1. Before 9.0, WebLogic Server supported a typed API layer over its JMX layer. It was possible for your JMX application classes to import type-safe interfaces for WebLogic Server MBeans, retrieve a reference to the MBeans through the `weblogic.management.MBeanHome` interface, and invoke the MBean methods directly.

A.17.1 JMX Deprecated Features

As of 9.0, the `MBeanHome` interface is deprecated. Instead of using this API-like programming model, all JMX applications should use the standard JMX programming model, in which clients use the `javax.management.MBeanServerConnection` interface to discover MBeans, attributes, and attribute types at run time. In this JMX model, clients interact indirectly with MBeans through the `MBeanServerConnection` interface.

If any of your classes import the type-safe interfaces (available under `weblogic.management`), Oracle recommends that you update them to use the standard JMX programming model. For more information, see "Understanding WebLogic Server MBeans" in *Developing Custom Management Utilities With JMX for Oracle WebLogic Server*.

A.18 WebLogic Administration and Configuration Scripts

Due to changes with the MBean hierarchy, Oracle does not guarantee that pre-9.2 configuration and administration scripts (such as WLST, `wlconfig`, `weblogic.Admin`, Ant, and so on) run in 12.1.1. Oracle recommends that you update your scripts to take advantage of the new features provided in WebLogic Server in version 9.2 and later. For more information about new features and changes in the MBean hierarchy, see the documents listed in Table A–4:

Table A–4 New Features in WebLogic Server by Release

For the following release . . .	. . . see the following for a description of new features
9.2	"What's New in WebLogic Server 9.2" at: `http://download.oracle.com/docs/cd/E13222_01/wls/docs92/notes/new.html`
10.0	"What's New in WebLogic Server 10.0" at: `http://download.oracle.com/docs/cd/E13222_01/wls/docs100/notes/new.html`
10.3	"What's New in WebLogic Server" for version 10.3 at: `http://download.oracle.com/docs/cd/E12840_01/wls/docs103/notes/new.html`
10.3.1	*What's New in Oracle WebLogic Server* for 11*g* Release 1 (10.3.1) at: `http://download.oracle.com/docs/cd/E12839_01/web.1111/e13852/toc.htm`
10.3.2	*What's New in Oracle WebLogic Server* for 11*g* Release 1 (10.3.2) at: `http://download.oracle.com/docs/cd/E15523_01/web.1111/e13852/toc.htm`
10.3.3	*What's New in Oracle WebLogic Server* for 11*g* Release 1 (10.3.3) at: `http://download.oracle.com/docs/cd/E14571_01/web.1111/e13852/toc.htm`
10.3.4	*What's New in Oracle WebLogic Server* for 11*g* Release 1 (10.3.4) at: `http://download.oracle.com/docs/cd/E17904_01/web.1111/e13852/toc.htm`
12.1.1	*What's New in Oracle WebLogic Server* for 12*c* Release 1 (12.1.1) at: `http://download.oracle.com/docs/cd/E24329_01/web.1211/e24494/toc.htm`

For additional information about upgrading your application infrastructure and the scripting tools that have been deprecated, see Section 2.4.1, "Step 1: Upgrade Your Application Infrastructure."

A.19 Dynamic Configuration Management

Configuration attributes are classified as *dynamic* or *non-dynamic*.

- Changes to dynamic configuration attributes are available as soon as they are activated, without restarting the affected server or system resource. These changes are made available to the server and run-time hierarchies when they are activated.
- Changes to non-dynamic configuration attributes are not immediately available. When a non-dynamic configuration attribute is changed, the server or system resource must be restarted to make the change effective.

WebLogic Server 9.0 introduced a change management process to provide a secure, predictable means for applying configuration changes in a domain. A batch change mechanism changes the way dynamic changes are applied when they are mixed with non-dynamic changes. Specifically, when a configured server or system resource is affected by a change to a non-dynamic attribute, no other changes (even dynamic changes) take effect, in current or future batches, until after the server or system resource is restarted. In this case, Oracle recommends that you restart the entity as soon as possible after the batch change is completed to ensure the system is in a consistent state and to allow future changes to be accepted.

You should test your configuration scripts to determine whether a non-dynamic change has been applied, and if so, restart the server. To determine whether a change is non-dynamic and requires a server restart:

- Before you activate a change, you can:
  - View the change listed in the Change Center in the Administration Console, as described in "Dynamic and Non-Dynamic Changes" in *Understanding Oracle WebLogic Server*.
  - Use the following WLST commands: `isRestartRequired` or `showChanges`. For more information, see *WebLogic Scripting Tool Command Reference*.
- After you activate a change, you can:
  - Review the server log to identify whether the change is categorized as non-dynamic.
  - Check the value of the `RestartRequired` or `PendingRestartSystemResources` attribute that is associated with the changed object, if applicable.

To determine which security attributes are dynamic or non-dynamic, see "Security Configuration MBeans" in *Securing Oracle WebLogic Server*.

For more information, see "Managing Configuration Changes" in *Understanding Domain Configuration for Oracle WebLogic Server*.

A.20 Modular Configuration and Deployment of JDBC Resources

As of WebLogic Server 9.0, the number of JDBC resource types was reduced to simplify JDBC configuration and to reduce the likelihood of configuration errors. Now, instead of configuring a JDBC connection pool and then configuring a data source or transactional data source to point to the connection pool and bind to the JNDI tree, you can configure a data source that encompasses a connection pool. For more information about simplified JDBC resource configuration introduced in WebLogic Server 9.0, see "Simplified JDBC Resource Configuration" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*, available at `http://download.oracle.com/docs/cd/E13222_01/wls/docs90/jdbc_admin/jdbc_intro.html#simple_res_config`.

The WebLogic Upgrade Wizard automatically converts JDBC data sources, connection pools, MultiPools, and data source factories to their new counterparts in WebLogic Server 12.1.1, as described in the following sections:

- JDBC Data Sources and Connection Pools
- MultiPools
- Data Source Factories

> **Note:** Each upgraded JDBC module contains an internal properties section. WebLogic Server uses internal properties to manage the data sources for backward compatibility. Also, some legacy attributes are preserved as properties in the Properties attribute of the JDBC data source file. Do not manually edit any internal properties.

For information about JDBC features, methods, interfaces, and MBeans that were deprecated as of WebLogic Server 9.0, see "Deprecated JDBC Features, Methods, Interfaces, and MBeans" in the *Release Notes*, available at `http://download.oracle.com/docs/cd/E13222_01/wls/docs90/notes/new.html#deprecated_jdbc_features`.

A.20.1 JDBC Data Sources and Connection Pools

The Upgrade Wizard converts legacy JDBC data source/connection pool pairs to two data source system resource modules, one for the data source and one for the connection pool:

- The data source that replaces the existing data source or transactional data source defines the data source parameters and refers to the second data source for its connection pool and related attributes.
- The data source that replaces the connection pool contains the JDBC driver parameters, the connection pool parameters, and the XA parameters.

> **Note:** Only data sources that are converted as part of a domain upgrade can refer to another data source for its connection pool. In all other cases, each data source contains its own pool of database connections.

During an upgrade, the Upgrade Wizard sets the `GlobalTransactionsProtocol` parameter for a data source based on the type of data source being converted (transactional or non-transactional) and the type of driver used in the related connection pool, as noted in Table A–5.

Table A–5 Parameter Settings for Global Transaction Protocol Parameter Setting

Legacy Data Source Type	Driver Type	Emulate Two-Phase Commit	GlobalTransactionProtocol
Tx Data Source	XA	N/A	`TwoPhaseCommit`
Tx Data Source	Non-XA	False	`OnePhaseCommit` (by default; not explicitly set)
Tx Data Source	Non-XA	True	`EmulateTwoPhaseCommit`[1]
Data Source	Non-XA	N/A	None

[1] Depending on your environment, you may want to consider using the `LoggingLastResource` (LLR) transaction protocol in place of the `EmulateTwoPhaseCommit` protocol for transaction processing because of its performance benefits. For more information see "Understanding the Logging Last Resource Transaction Option" in *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server*.

A.20.2 MultiPools

The Upgrade Wizard converts a MultiPool to a multi-data source, which is another instance of a data source object that provides load balancing or failover between data sources.

A.20.3 Data Source Factories

Data source factories are deprecated as of WebLogic Server 9.0 and are included for backward compatibility only. No conversion of data source factories is required.

A.21 Thread Management

Oracle recommends using Work Manager concepts to manage threads because execute queues are no longer the default method used as of WebLogic Server 9.0. You define the rules and constraints for your application by defining a Work Manager and applying it either globally to a WebLogic domain or specifically to an application component. For more information, see "Using Work Managers to Optimize Scheduled Work" in *Configuring Server Environments for Oracle WebLogic Server*.

In WebLogic Server 8.1, processing was performed in multiple execute queues. If you had been using execute queues to improve performance in 8.1, you may continue to use them after you upgrade your application domains. Oracle provides a `use81-style-execute-queues` flag that enables you to disable the self-tuning execute pool and provide backward compatibility for upgraded applications to continue to use user-defined execute queues. For information about enabling the backward compatibility flag, and configuring and monitoring execute queues, see "How to Enable the WebLogic 8.1 Thread Pool Model" in WebLogic Server *Performance and Tuning for Oracle WebLogic Server*.

A.22 XML Implementation

Please note the following changes to XML support as of WebLogic Server 9.0:

- The default XML parser is the XML parser shipped with the Sun Java 2 JDK. The previous default XML parser, the Apache Xerces parser (`weblogic.apache.xerces.*`), is deprecated as of 9.0.

  You can modify the XML parser that is used by default using the Administration Console. For information about configuring the XML parser, see "Difference In Default Parsers Between Versions 8.1 and 9.0 of WebLogic Server" in *Programming XML for Oracle WebLogic Server*.

- As of 9.0, WebLogic Server supports Streaming API for XML (StAX), a standard specification from the Java Community Process that provides an easy and intuitive means of parsing and generating XML documents. StAX gives you more control over XML parsing than the WebLogic XML Streaming API, which is deprecated as of 9.0. For information about using StAX, see "Using the Streaming API for XML (StAX)" in *Programming XML for Oracle WebLogic Server*.

- You can no longer parse XML documents from within a servlet using the `setAttribute` and `getAttribute` methods without some preliminary setup. Specifically, as of 9.0, you must configure a WebLogic Server servlet filter called

`weblogic.servlet.XMLParsingHelper` (deployed, by default, on all WebLogic Server instances) as part of your Web application. For more information, see "Parsing XML Documents in a Servlet" in *Programming XML for Oracle WebLogic Server*.

A.23 XMLBeans and XQuery Implementation

As of 9.0, the XMLBean implementation in WebLogic Server has been moved from an internal library (`com.bea.xml`) to the Apache open source project (`org.apache.xmlbeans`).

If you used XMLBeans in your WebLogic Server 8.1 applications, you must perform the following steps:

1. Update the package name used by XMLBeans from `com.bea.xml` to `org.apache.xmlbeans`.
2. Recompile your XMLBean schemas to update the schema metadata (`.xsb`) files and generated code.

As of 9.0, the XMLQuery (XQuery) implementation conforms to the following specifications:

- *XQuery 1.0 and XPath 2.0 Data Model—W3C Working Draft 23 July 2004* available from the W3C Web site at `http://www.w3.org/TR/2004/WD-xpath-datamodel-20040723`.
- *XQuery 1.0: An XML Query Language—W3C Working Draft 23 July 2004* available from the W3C Web site at `http://www.w3.org/TR/2004/WD-xquery-20040723`.

In WebLogic Server 8.1, the XQuery implementation conformed to *XQuery 1.0 and XPath 2.0 Functions and Operators—W3C Working Draft 16 August 2002*, available from the W3C Web site at `http://www.w3.org/TR/2002/WD-xquery-operators-20020816/`. The 2002 XQuery implementation is deprecated as of 9.0.

In most cases, simple XQuery and XPath operations in pre-9.0 code behaves the same in 10.0. To ensure that the XQuery and XPath operations produce the expected results, you can review or update the existing `XMLObject.selectPath()` and `XMLObject.execQuery()` method calls using one of the following methods:

- To guarantee 8.1-style behavior, update the existing method calls to include a new parameter that specifies that the 2002 XQuery engine is to be used instead of the new 2004 XQuery engine. For example:

```
import org.apache.xmlbeans.impl.store.Path;
XmlObject xo = ?
xo.selectPath(".//c", (new XmlOptions()).put(Path._
forceXqr12002ForXpathXQuery));
```

> **Note:** The 2002 XQuery engine is deprecated as of WebLogic Server 9.0, and is available for backward compatibility. It is only used if you specify this parameter. Otherwise, the 2004 XQuery engine is used, by default.

- To guarantee conformance with the 2004 XQuery engine, review your pre-9.0 scripts to identify any changes that may be required with the syntax or semantics

of the XQuery strings that are passed to the method calls and update methods accordingly.

As of 9.0, the behavior of `XMLCursor.moveXML()` has changed. In 8.1, a cursor that was inside a moved fragment remained on the original document. As of 9.0, cursors move with fragments.

A.24 Deployment Descriptor Validation and Conversion

This section describes changes in the use of deployment descriptors in a WebLogic Server environment, as of release 9.0:

- Deployment descriptor validation is more strict as of the 9.0 release of the EJBGen and ejbc tools. For example, an error is returned if a `cmr-field` is defined in `@ejbgen:relation`, but there are no methods tagged with `@ejbgen:cmr-field` in the Bean class.

  Note: ejbc is deprecated as of WebLogic Server 9.0; you should use appc instead. For more information, see "appc Reference" in *Programming WebLogic Enterprise JavaBeans for Oracle WebLogic Server*.

- In pre-9.0 versions of WebLogic Server, applications that define multiple modules, as illustrated in the following excerpt from a configuration file, are deployed successfully regardless of whether a `META-INF\application.xml` deployment descriptor is defined as part of the application:

  ```
  <Application Deployed="true" Name="SessionBeanLifeCycleBean"
  Path="C:\bea\weblogic70\tools\deployment\ejb" TwoPhase="false">
              <EJBComponent Name="CMFinderTestBean" Targets="myserver"
  URI="CMFinderTestBean.jar"/>
              <EJBComponent Name="SessionBeanLifeCycleBean" Targets="myserver"
  URI="SessionBeanLifeCycleBean.jar"/>
  </Application>
  ```

  As of 9.0, the `META-INF\application.xml` deployment descriptor is *required* if a deployed application defines multiple modules. If this type of deployment descriptor is not provided, the upgrade fails with an error similar to the following:

  ```
  [J2EE Deployment SPI:260089]Unable to determine type of application at path
  'C:\bea\weblogic70\tools\deployment\ejb' and upgrade will not succeed.
  ```

  When upgrading a domain, make sure that the deployed applications adhere to the proper Java EE application format. For example, if required by the application, make sure that the applications define the `META-INF\application.xml` or `META-INF\weblogic-application.xml` deployment descriptors.

  For more information about the deployment descriptors, see "Enterprise Application Deployment Descriptor Elements" in *Developing Applications for Oracle WebLogic Server*.

- So that your applications can take advantage of the features in the current Java EE specification and release of WebLogic Server, Oracle recommends that you upgrade deployment descriptors when you upgrade applications to a new release of WebLogic Server. For more information, see "Upgrading Deployment Descriptors From Previous Releases of J2EE and WebLogic Server" in *Developing Applications for Oracle WebLogic Server*.

A.25 Deprecated Startup and Shutdown Classes

As of 9.0, application-scoped startup and shutdown classes were deprecated in WebLogic Server, in favor of applications that respond to application lifecycle events. Oracle recommends that you update your application environment to use the lifecycle events in place of application-scoped and domain-level startup and shutdown classes. For more information, see "Programming Application Life Cycle Events" in *Developing Applications for Oracle WebLogic Server*.

A.26 Resource Adapters

Table A–6 lists the configuration settings for resource adapters that are deprecated or no longer supported. For more information about new features and changes, see "What's New in WebLogic Server".

Table A–6 Deprecated or Unsupported Resource Adapter Configuration Settings

This element ...	As of WebLogic Server 9.0 ...
Link-Ref Mechanism	This element has been deprecated and replaced by the new Java EE libraries feature. For more information about Java EE libraries, see "Creating Shared J2EE Libraries and Optional Packages" in *Developing Applications for Oracle WebLogic Server*. The Link-Ref mechanism is still supported in this release for resource adapters developed under the J2CA 1.0 Specification. For more information about using the Link-Ref mechanism with 1.0 resource adapters, see "(Deprecated) Configuring the Link-Ref Mechanism" in "Configuring the weblogic-ra.xml File" in *Programming Resource Adapters for Oracle WebLogic Server*.
`<shrink-period-minutes>`	This element has been deprecated and replaced by `<shrink-frequency-seconds>`, which you can use to specify the shrink period in increments of seconds, instead of minutes. The `<shrink-frequency-seconds>` element overrides the `<shrink-period-minutes>` element if both are set.
`<connection-maxidle-time>`	This element has been deprecated and is replaced by `<inactive-connection-timeout-seconds>`, which you can use to specify the connection timeout in increments of seconds. The `<inactive-connection-timeout-seconds>` element overrides the `<connection-maxidle-time>` element if both are set.
`<security-principal-map>`	This element is no longer supported; the security principal map is configured using the Administration Console. You should remove the `<security-principal-map>` definition from the weblogic-ra.xml file. Otherwise, deployment of the resource adapter fails.
`<connection-cleanup-frequency>`	This element is no longer supported and is ignored during deployment.
`<connection-duration-time>`	This element is no longer supported and is ignored during deployment.

A.27 WLEC

WLEC was deprecated in WebLogic Server 8.1. WLEC users should move applications to the WebLogic Tuxedo Connector, as described in *WebLogic Tuxedo Connector Migration Guide for WLEC to Oracle WebLogic Server*.

B

WebLogic Domain Directory Structure Enhancements

This appendix describes enhancements to the structure of the WebLogic Server directory structure. The information in this appendix applies only if you are upgrading from WebLogic Server 8.1.

As of 9.0, WebLogic Server offers the following enhancements to the structure of the WebLogic domain directory:

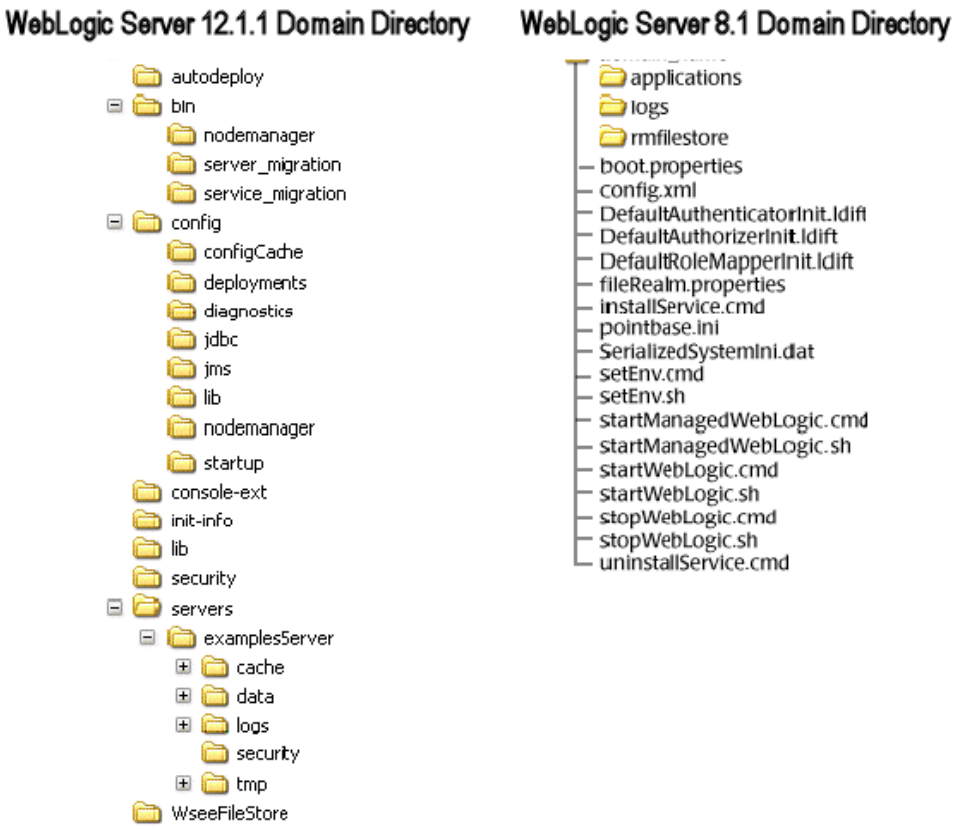
- To improve configuration management and promote XML file validation, WebLogic Server supports the specification of domain configuration data in multiple files, including `config.xml`, in the new *`domain_name`*`/config` directory. (Here, *`domain_name`* specifies the domain directory.) In previous releases, the `config.xml` file was the repository for all configuration information. Now, new subdirectories of config maintain configuration modules for diagnostic, JDBC, JMS, and Node Manager subsystems. Each configuration file adheres to an XML Schema definition.
- Startup and shutdown scripts are maintained in the *`domain_name`*`/bin` directory. In previous releases, they were stored in the root directory of the domain.

In addition to the structural enhancements to the domain directory, WebLogic Server supports utilities for managing changes to server configuration. These tools enable you to implement a secure, predictable means for distributing configuration changes in a domain. For more information, see "Managing Configuration Changes" in *Understanding Domain Configuration for Oracle WebLogic Server*.

For more details about the WebLogic domain directory structure, see "Domain Configuration Files" in *Understanding Domain Configuration for Oracle WebLogic Server*.

Figure B–1 compares the domain directory structures introduced in WebLogic Server 10.0 with WebLogic Server 8.1.

Figure B–1 WebLogic Server 12.1.1 and WebLogic Server 8.1 Directory Structures



The following sections describe the domain directory structures for WebLogic Server 8.1.

B.1 WebLogic Server 8.1 Domain Directory Structure

In WebLogic Server 8.1 environments, the domain directory structure created by the Configuration Wizard contains:

- A domain root directory with the same name as the domain, such as `mydomain` or `petstore`. This directory contains the following:
  - `config.xml` file for the domain
  - Scripts used to start server instances and establish the environment
  - Subdirectory for storing applications for the domain, typically named `applications`.

When you start a server instance in a domain for the first time, WebLogic Server creates the following subdirectories in the domain directory:

- Files containing security information
- `logs` directory for storing domain-level logs
- For each server running in the domain, a directory for storing server logs and HTTP access logs

For more details about the WebLogic Server 8.1 domain directory structure, see "Overview of WebLogic Server Domains" in *Configuring and Managing WebLogic Server*,

available at http://download.oracle.com/docs/cd/E13222_01/wls/docs81/adminguide/overview_domain.html. For a summary of the directory structure contents for the default configuration templates, see "Template Reference" in *Creating and Configuring WebLogic Domains Using the Configuration Wizard*, available at http://download.oracle.com/docs/cd/E13196_01/platform/docs81/confgwiz/tempref.html.

C

Upgrade Wizard Command-Line Reference

This appendix provides detailed information about how to invoke the WebLogic Upgrade Wizard from the command line.

The command to invoke the WebLogic Upgrade Wizard has the following syntax:

```
java weblogic.Upgrade [-help | -h | -? | -usage] [-type type] [-mode mode]
[-responses xmlfile] [-out file]
```

Table C–1 describes each of the arguments that may be specified in this command.

Table C–1 Command-Line Arguments for the WebLogic Upgrade Wizard

Argument	Description
`-help | -h | -? | -usage`	Displays help information.
`-type` *`type`*	Specifies one of the following types of upgrade: `domain`, `nodemanager`, or `securityproviders`. The default value is `domain`. **Note:** The `-type` domain option can only be used to upgrade a WebLogic domain in silent mode. For more information, see Section 5.3.2, "Upgrading a Domain in Silent Mode."
`-mode` *`mode`*	Specifies one of the following modes of upgrade: `gui` or `silent`. The default value is `gui`.
`-responses` *`xmlfile`*	Specifies the location of the XML file that contains the upgrade requirements to be used during a silent upgrade. This argument is valid only when `-mode` is set to `silent`. For more information about the format of the responses file, see Appendix D, "Silent Upgrade XML Script Reference."
`-out` *`file`*	Specifies the file in which standard output and errors are stored. By default, log output is sent to `stdout`.

C.1 Examples

- The following command upgrades a domain in interactive mode:

  ```
  java weblogic.Upgrade
  ```

- The following command upgrades a domain in silent mode:

  ```
  java weblogic.Upgrade -mode silent -type domain
  ```

D

Silent Upgrade XML Script Reference

This appendix describes how to run the WebLogic Upgrade Wizard in silent mode.

Before using the WebLogic Upgrade Wizard in silent mode, you have the option of creating an XML script that defines your upgrade requirements, and passing that script to the wizard on the command line.

When run in silent mode, the Upgrade Wizard searches the domain's root directory for an XML script with a name that indicates the type of upgrade task to be performed. If the wizard does not locate an XML script, it uses default system values.

This appendix provides the following sample XML upgrade scripts:

- Section D.2, "Security Provider Upgrade Script"
- Section D.3, "Node Manager Upgrade Script"
- Section D.4, "Domain Upgrade Script"

Depending on the particular configuration of the environment you are upgrading, you can modify specific values in these scripts, as explained in this appendix, and have the WebLogic Upgrade Wizard use them instead. This appendix identifies those values and their defaults, and explains how they can be modified.

Note: Only WebLogic domains can be upgraded using silent mode.

D.1 About Modifying the Sample XML Scripts

To create an XML script for a silent mode upgrade, modify one of the sample scripts provided in this appendix. Note the following:

- The XML definition (`<?xml version="1.0" encoding="UTF-8"?>`) must be at the very beginning of the XML script. No spaces or line breaks are allowed before the XML definition.
- You must follow XML guidelines for characters when modifying values. That is, you cannot use characters reserved for use in XML, such as `<`, `>`, `[`, and `]`.
- Save the sample script as an XML file in the root directory of the domain to upgrade. You must use the same name for the script as identified in this appendix.

D.2 Security Provider Upgrade Script

To upgrade the security providers in a domain, the WebLogic Upgrade Wizard looks for an XML script named

weblogic-upgrade-securityproviders-responses.xml, which is displayed in Example D–1. The values that may be modified are shown in **bold**.

Example D–1 Sample Silent-Mode XML Script for Upgrading Security Providers

```
<?xml version="1.0" encoding="UTF-8"?>

<plugin-silent-responses>
</plugin-silent-responses>

<!--
<plugin-silent-responses>
  <group name="SecurityProviderUpgradeGroup">
    <plugin name="SecurityProviderUpgradeStepOne">
      <input-adapter name="IA">
        <bind-property name="selectedFileNames">
          <value>__INPUT_DIRECTORY__</value>
        </bind-property>
      </input-adapter>
    </plugin>
    <plugin name="SecurityProviderUpgradeStepTwo">
      <input-adapter name="IA">
        <bind-property name="selectedFileNames">
          <value>__OUTPUT_DIRECTORY__</value>
        </bind-property>
      </input-adapter>
    </plugin>
  </group>
</plugin-silent-responses>
```

Table D–1 identifies the keywords contained in this script, the values you can specify for those keywords, and the default values that are used by the WebLogic Upgrade Wizard if you do not modify them.

Table D–1 Silent-Mode XML Script Values for Upgrading Security Providers

For this keyword...	Set the value to the...	This keyword defaults to the...
INPUT_DIRECTORY	Path of the directory that contains the security provider JARs to be upgraded. For example: d:/bea/weblogic81/server/lib/mbeantypes By default, security providers are located in *WL_HOME*\server\lib\mbeantypes, where *WL_HOME* specifies the root directory of the pre-9.0 installation.	Path of directory from which the Upgrade Wizard is run, such as: c:\Oracle\Middleware\wlserver_12.1\server\lib\mbeantypes
OUTPUT_DIRECTORY	Path of the directory in which you want to save the new security provider JAR files. For example: d:/bea/wlserver_10.3/server/lib/mbeantypes	*WL_HOME*\server\lib\mbeantypes, where *WL_HOME* specifies the root directory of the WebLogic Server 10.x installation.

D.3 Node Manager Upgrade Script

To upgrade the Node Manager from an existing domain, the WebLogic Upgrade Wizard looks for an XML script named weblogic-upgrade-nodemanager-responses.xml, which is displayed in Example D–2. The values that may be modified are shown in **bold**.

Example D–2 Sample Silent-Mode XML Script for Upgrading a Node Manager

```
<?xml version="1.0" encoding="UTF-8"?>

<plugin-silent-responses>
</plugin-silent-responses>

<!--
<plugin-silent-responses>
  <group name="NodeManagerPlugInGroup">
    <plugin name="NodeManagerPlugIn">
      <input-adapter name="IA">
        <bind-property name="selectedFileNames">
          <value>__NODE_MANAGER_HOME__</value>
        </bind-property>
      </input-adapter>
    </plugin>
  </group>
</plugin-silent-responses>
```

Table D–2 identifies the keyword contained in this script, the value you can specify for that keyword, and the default value used by the WebLogic Upgrade Wizard if you do not modify it.

Table D–2 Silent-Mode XML Script Value for Upgrading the Node Manager

For this keyword...	Set the value to the...	This keyword defaults to the...
NODE_MANAGER_HOME	Path of the directory of Node Manager to be upgraded by navigating the local directory hierarchy. By default, the Node Manager home directory is located in *WL_HOME*\common\nodemanager, where *WL_HOME* specifies the root directory of the pre-9.0 installation.	Directory from which the Upgrade Wizard is run. For example, c:\Oracle\Middleware\wlserver_12.1\common\nodemanager.

D.4 Domain Upgrade Script

To upgrade an existing domain, the WebLogic Upgrade Wizard looks for an XML script named weblogic-upgrade-domain-responses.xml, which is displayed in Example D–3. The values that may be modified are shown in **bold**.

> **Note:** Specifying the name of the domain Administration Server, as described in Table D–3 for the keyword ADMIN_SERVER_NAME, is required because there is no default value.

Example D–3 Sample Silent-Mode XML Script for Upgrading a Domain

```
<?xml version="1.0" encoding="UTF-8"?>

<plugin-silent-responses>
</plugin-silent-responses>

<!-- SAMPLE BELOW -->
<!--
<plugin-silent-responses>
  <group name="DomainSelectionGroup">
    <plugin name="SelectWebLogicVersionPlugIn">
      <input-adapter name="ChoiceIA">
```

```
        <bind-property name="selectedChoiceIds">
          <value>__WEBLOGIC_VERSION__</value>
        </bind-property>
      </input-adapter>
    </plugin>
    <plugin name="DomainDirectorySelectionPlugIn">
      <input-adapter name="IA">
        <bind-property name="selectedFile">
          <value>__DOMAIN_DIR__</value>
        </bind-property>
      </input-adapter>
    </plugin>
  </group>
  <group name="PostDirSelectionGroup">
    <plugin name="AdminServerSelectionPlugIn">
      <input-adapter name="IA">
        <bind-property name="selectedChoiceIds">
          <value>__ADMIN_SERVER_NAME__</value>
        </bind-property>
      </input-adapter>
    </plugin>
    <plugin name="NodeManagerCredentialsPlugIn">
      <input-adapter name="UsernameIA">
        <bind-property name="value">
          <value>__NODE_MANAGER_USERNAME__</value>
        </bind-property>
      </input-adapter>
      <input-adapter name="PasswordIA">
        <bind-property name="value">
          <value>__NODE_MANAGER_PASSWORD__</value>
        </bind-property>
      </input-adapter>
      <input-adapter name="PasswordConfirmIA">
        <bind-property name="value">
          <value>__NODE_MANAGER_PASSWORD__</value>
        </bind-property>
      </input-adapter>
    </plugin>
    <plugin name="OptionalGroupsSelectionPlugIn">
      <input-adapter name="IA">
        <bind-property name="selectedChoiceIds">
          <value>__OPTIONAL_ACTION_1__</value>
          <value> . . . </value>
        </bind-property>
      </input-adapter>
    </plugin>
  </group>
  <group name="PostDirSelectionPost81Group">
    <plugin name="AdminServerSelectionPlugIn">
      <input-adapter name="IA">
        <bind-property name="selectedChoiceIds">
          <value>__ADMIN_SERVER_NAME__</value>
        </bind-property>
      </input-adapter>
    </plugin>
    <plugin name="OptionalGroupsSelectionPlugIn">
      <input-adapter name="IA">
        <bind-property name="selectedChoiceIds">
          <value>__OPTIONAL_ACTION_1__</value>
          <value> . . . </value>
```

```
        </bind-property>
      </input-adapter>
    </plugin>
  </group>
  <group name="DomainBackupGroup">
    <plugin name="DomainDirectoryBackupPlugIn">
      <input-adapter name="FileSelectionIA">
        <bind-property name="selectedFileNames">
          <value>__BACKUP_DIR__</value>
        </bind-property>
      </input-adapter>
      <input-adapter name="TextIA">
        <bind-property name="value">
          <value>___BACKUP_FILE_NAME__</value>
        </bind-property>
      </input-adapter>
    </plugin>
  </group>
</plugin-silent-responses>

-->
```

Table D–3 identifies the keywords contained in this script, the values you can specify for the keywords, and the default values used by the WebLogic Upgrade Wizard if you do not modify them.

Table D–3 Silent-Mode XML Script Value for Upgrading a Domain

For this keyword...	Set the value to...	The keyword defaults to...
WEBLOGIC_VERSION	The version of WebLogic Server contained in the domain being upgraded[1].	The version of WebLogic Server identified in the configuration file (config.xml) of the domain being upgraded. For example, 8.1.4.0. If the software version number is not specified in the domain configuration file, the wizard displays the version number as 8.1.0.0, by default. In this case, there is no impact if the default value does not match the actual version number of the pre-9.0 domain.
ADMIN_SERVER_NAME	Name of the domain's Administration Server instance. **Note:** Assigning a value for this keyword is required because there is no default. If you do not provide the Administration Server name, this silent upgrade script generates an exception and terminates.	No default value is assigned.
DOMAIN_DIR	The path of the directory that contains the WebLogic domain to be upgraded.	The directory from which the Upgrade Wizard is run. For example, c:\bea\user_projects\domains\mydomain.
NODE_MANAGER_USERNAME	The username for Node Manager.	weblogic

Table D–3 (Cont.) Silent-Mode XML Script Value for Upgrading a Domain

For this keyword...	Set the value to...	The keyword defaults to...
NODE_MANAGER_PASSWORD	The password for Node Manager.	weblogic
NODE_MANAGER_PASSWORD	The confirmation password for Node Manager.	weblogic
OPTIONAL_ACTION_1	One or more of the following options: ■ DOMAIN_DIRECTORY_BACKUP_SELECTED_VALUE If this option is specified, the wizard backs up the original domain directory and stores it in a zip file. ■ DOMAIN_DIRECTORY_BACKUP_LOG_FILES_INCLUDED_SELECTED_VALUE If this option is specified, log files are included in the backup zip file. The number and size of log files can be large, so you may want to exclude them from the backup file by disabling this option. By default, log files are included in the backup file. ■ SKIP_BACKWARDS_COMPATIBILITY_FLAGS_SELECTED_VALUE Some behavior supported in pre-9.0 releases of WebLogic Server has been changed as of 9.0 to comply with J2EE 1.4. By default, the wizard sets flags to enable the previous behavior in the new domain. If you specify this option, these flags for backward compatibility are not set. Note that OPTIONAL_ACTION_1 may be specified so that one of multiple options is used. For example: DOMAIN_DIRECTORY_BACKUP_SELECTED_VALUE OR DOMAIN_DIRECTORY_BACKUP_LOG_FILES_INCLUDED_SELECTED_VALUE OR SKIP_BACKWARDS_COMPATIBILITY_FLAGS_SELECTED_VALUE	DOMAIN_DIRECTORY_BACKUP_SELECTED_VALUE, DOMAIN_DIRECTORY_BACKUP_LOG_FILES_INCLUDED_SELECTED_VALUE
BACKUP_DIR	Path of the directory in which you want to save the backup zip file.	Directory from which the Upgrade Wizard is run. For example, c:\Oracle\Middleware\user_projects\domains\mydomain.
BACKUP_FILE_NAME	Name of the backup zip file.	weblogic-domain-backup-mydomain.zip

[1] You can specify multiple versions of WebLogic Server, as in this example: 8.1 OR "9.0 or higher"

E

Upgrading a Domain at Administration Server Startup (Implicit Mode)

This appendix describes how to upgrade a domain using implicit mode.

The implicit mode of upgrade enables you to upgrade a WebLogic domain automatically at server startup. This mode should be used only in a development environment, and it is valid only for the computer on which the Administration Server resides. The implicit mode of upgrade is not recommended for use in a production environment.

> **Note:** The implicit mode of upgrade can only be used to upgrade a WebLogic domain.

Before proceeding, make sure that you have performed the prerequisite steps described in Section 2.2, "Prepare to Upgrade."

To start the WebLogic Upgrade Wizard in implicit mode:

1. Verify that the WebLogic domain is not running.
2. Open an MS-DOS command prompt window (on Windows) or a command shell (on UNIX) and set up the environment as described in Section 2.2.6, "Step 6: Set Up the Environment."
3. Update the startup scripts in the domain to reference the WebLogic Server 12.1.1 installation.

   For example, make sure *WL_HOME* is set to the WebLogic Server 12.1.1 installation. For example: `c:\Oracle\Middleware\wlserver_12.1`.

   By default, the WebLogic Upgrade Wizard prompts you for confirmation before starting the upgrade process. You can disable the prompt by specifying, in the startup script, the following command argument for starting the server:

   ```
   -Dweblogic.ForceImplicitUpgradeIfNeeded=true
   ```

4. Start up the Administration Server in your domain.

   Respond to the upgrade confirmation prompt, if necessary.

   The WebLogic Upgrade Wizard checks whether the domain directory was generated using a valid version of WebLogic Server (8.1), and, if the result is true, it automatically upgrades the domain. This process occurs only once: the first time you start the Administration Server.

Note: Implicit upgrade is not available for WebLogic Server version 9.x or 10.0 to 12.1.1 because upgrade is optional between these two versions. You can run a 9.x or 10.0 domain under 12.1.1 without modifications.