# BEA WebLogic Server™

## BEA WebLogic Server 8.1 Upgrade Guide

Release 8.1
Revised: December 17, 2002

## Copyright



## Restricted Rights Legend



## Trademarks or Service Marks



BEA WebLogic Server 8.1 Upgrade Guide

| Part Number | Revised | Software Version |
|---|---|---|
| N/A | December 17, 2002 | BEA WebLogic Server Version 8.1 |

## About This Document



## 1. Upgrading WebLogic Server 7.0 to Version 8.1

## 2. Upgrading WebLogic Server 6.x to Version 8.1

## 3. Upgrading WebLogic Server 4.5 and 5.1 to Version 8.1

# About This Document

This document provides procedures and other information you need to upgrade earlier versions of BEA WebLogic Server to WebLogic 8.1. It also provides information about moving applications from an earlier version of WegLogic Server to 8.1.

The document is organized as follows:

- Chapter 2, "Upgrading WebLogic Server 6.x to Version 8.1,"describes how to upgrade to WebLogic Server 8.1 from WebLogic Server 6.x.
- Chapter 3, "Upgrading WebLogic Server 4.5 and 5.1 to Version 8.1,"describes how to upgrade to WebLogic Server 8.1 from WebLogic Server 4.5 or 5.1.
- Appendix A, "The weblogic.properties Mapping Table," shows which `config.xml, web.xml, or weblogic.xml` attribute handles the function formerly performed by `weblogic.properties` properties.

## Audience

This document is written for all users of WebLogic Server 4.5, 5.1, 6.0, and 6.1 who want to upgrade to WebLogic Server 8.1.

## e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

# How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File—Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

# Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

# Documentation Conventions

The following documentation conventions are used throughout this document.

| Convention | Usage |
|---|---|
| Ctrl+Tab | Keys you press simultaneously. |
| *italics* | Emphasis and book titles. |
| `monospace text` | Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.<br>*Examples*:<br>`import java.util.Enumeration;`<br>`chmod u+w *`<br>`config/examples/applications`<br>`.java`<br>`config.xml`<br>`float` |
| *`monospace italic text`* | Variables in code.<br>*Example*:<br>`String CustomerName;` |
| UPPERCASE TEXT | Device names, environment variables, and logical operators.<br>*Examples*:<br>LPT1<br>BEA_HOME<br>OR |
| `{ }` | A set of choices in a syntax line. |

| Convention | Usage |
|---|---|
| `[ ]` | Optional items in a syntax line. *Example*:<br>`java utils.MulticastTest -n` *`name`* `-a` *`address`*<br>`[-p` *`portnumber`*`] [-t` *`timeout`*`] [-s` *`send`*`]` |
| `\|` | Separates mutually exclusive choices in a syntax line. *Example*:<br>`java weblogic.deploy [list\|deploy\|undeploy\|update]`<br>`password {application} {source}` |
| `...` | Indicates one of the following in a command line:<br>■ An argument can be repeated several times in the command line.<br>■ The statement omits additional optional arguments.<br>■ You can enter additional parameters, values, or other information |
| .<br>.<br>. | Indicates the omission of items from a code example or from a syntax line. |

CHAPTER

# 1 Upgrading WebLogic Server 7.0 to Version 8.1

Upgrading WebLogic Server 7.0 to version 8.1 involves changing your WebLogic Server start command scripts and environment settings. In some cases, it is necessary to move your domain directory.

For example procedures to upgrade applications from earlier versions of WebLogic Server to WebLogic Server 8.1, see Upgrading Example Applications to WebLogic Server 8.1.

For answers to specific questions on upgrading to WebLogic Platform 8.1, see the Upgrading section of the WebLogic Server FAQs.

The following sections of this document contain information for upgrading your system from WebLogic Server 7.0 to WebLogic Server 8.1:

# Upgrading Your WebLogic Server Configuration: Main Steps

Take the following steps to upgrade from WebLogic Server 7.0 to WebLogic Server 8.1:

1. Make a backup copy of your 7.0 domain before you begin the upgrade procedure. After you start running your domain using WebLogic Server 8.1, you cannot convert it back to 7.0.
2. Install WebLogic Server 8.1. See the the *Installation Guide*.

   **Note:** The installer will prevent you from installing the new version directly over the old version. You must select a new directory location.
3. Modify your 7.0 startup scripts to work with WebLogic Server 8.1. See "Modifying Startup Scripts" on page 1-3.
4. Port your applications to WebLogic Server 8.1. See "Upgrading an Application from WebLogic Server 7.0 to WebLogic Server 8.1" on page 1-4.
5. If necessary, perform other upgrade procedures as described in Additional Upgrade Procedures and Information.

To upgrade a cluster of servers, follow the above steps for each server and then follow the steps outlined in *Setting Up WebLogic Clusters* in *Using WebLogic Server Clusters*. In cases where you invoke an application by using RMI/T3 or RMI/IIOP, WebLogic Server 6.1, 7.0, and 8.1 are interoperable. Within a domain, however, all servers must be of the same version.

For information on upgrading WebLogic Server license files, see *Upgrading Licenses from WebLogic Server 7.0* in the *Installation Guide*.

# Modifying Startup Scripts

If you used WebLogic Server startup scripts with a previous version of the product, modify them to work with WebLogic Server 8.1.

For other examples of modifying the startup scripts, see Upgrading Example Applications to WebLogic Server 8.1.

1. If you are using a script based on the startup script that the WebLogic Server 7.0 Domain Configuration Wizard generated for your WebLogic Server 7.0 domain, open the script in your editor and change the path to the script that starts the WebLogic Server 7.0 server:

   ```
   @rem Call Weblogic Server 7.0
   call "WL_HOME\server\bin\startWLS.cmd"
   ```

   to call instead the WebLogic Server 8.1 server.

   ```
   @rem Call WebLogic Server 8.1
   call "WL_HOME\server\bin\startWLS.cmd"
   ```

You may need to modify other settings in your startup script. Some possibilibities are described below.

2. Modify `bea.home` property

   to point to your BEA home directory containing the `license.bea` file for WebLogic Server 8.1. For example:

   ```
   -Dbea.home=C:\bea810
   ```

3. Modify `WL_HOME`

   must point to your WebLogic Server 8.1 installation directory. For example:

   ```
   WL_HOME=c:\bea810\weblogic810
   ```

4. Modify `PATH`

   so that it includes your `%WL_HOME%` 8.1 home. For example:

   ```
   PATH=%WL_HOME%\bin;%PATH%
   ```

5. Modify `CLASSPATH`

   so that it points to the WebLogic Server 8.1 classes. For example:

```
CLASSPATH=%WL_HOME%\lib\weblogic.jar
```

6. Modify or eliminate any WebLogic Server 7.0 startup script directory structure tests. For example, if your script tries to verify a relative path, either fix the directory structure test or remove it.

WebLogic Server 8.1 installs the JVM, JDK 1.4.1, with the server installation. The `setenv.sh` scripts provided with the server all point to the JVM. The latest information regarding certified JVMs is available at the *Certifications Page*.

# Understanding the WebLogic Server 8.1 Directory Structure

The directory structure in WebLogic Server 8.1 is different from that of 7.0. For complete information on the updated directory structure see *Understanding the WebLogic Server Directory Structure* in *Performing Post-Installation Tasks* in the *Installation Guide*.

If you are booting your WebLogic Server 7.0 domain with the WebLogic Server 8.1 environment, the new directory structure is created automatically. However, if you have custom tools or scripts that rely on the WebLogic Server 7.0 domain directory structure, you need to update those tools relative to the new directory structure. Similarly, if you have a scripted tool for creating domains in the WebLogic Server 7.0 environment, you will have to change those scripts. It is best to use the *Configuration Wizard*, which can be scripted.

# Upgrading an Application from WebLogic Server 7.0 to WebLogic Server 8.1

The following steps upgrade an application from WebLogic Server 7.0 to WebLogic Server 8.1:

- Install WebLogic Server 8.1
- Create or select a directory for your application's WebLogic Server 8.1 configuration domain
- Copy your configuration information to the new domain directory
- Make sure that relative paths to external files are correct.
- Edit server start scripts.

This section describes these steps in detail.

1. Install WebLogic Server 8.1, if you have not already done so. See the *Installation Guide* for instructions.

   **Note:** Installing the new version in the exact location of the old version is explicitly prohibited by the installer.

2. Create or select a directory for the WebLogic Server 8.1 domain that will house all the configuration information for your application after you have upgraded it.

   Each 7.0 and 8.1 domain must have its own directory, because it is not possible to have multiple `config.xml` files in the same directory.

   If your 7.0 `/config/domain` directory is not located in the WebLogic Server 7.0 installation, you can re-use your WebLogic 7.0 location for WebLogic Server 8.1.

3. If you are not reusing the directory your application used for its WebLogic Server 7.0 domain, copy the application's `/config/`*`domain`* directory to the directory location you chose or created for the WebLogic Server 8.1 domain.

   Tip: You can delete any directories that begin with a dot ("**.**"), which are files or directories that WebLogic Server has created for its own use.

4. Make the relative paths in the application's deployment descriptor files (`web.xml` and `weblogic.xml`) point to the actual locations of any external files to which they refer.

   WebLogic Server configurations rely on a number of files that may be stored on the file system. Typically, these files are persistence repositories (log files, file-based repositories, etc.) or utilities (Java compiler). These files can be configured using fully qualified or relative paths.

   If all external files are defined using relative paths and are located in or below the domain directory, skip the remainder of this step.

For external files that are defined using relative paths that are located outside the domain directory, re-create the directory structure relative to the new config directory and copy the associated files into the new directories. For external files that are defined using fully qualified paths, determine whether it is appropriate to re-use these files in the WebLogic Server 8.1 deployment.

For example, log files and persistence stores can be re-used; however, you may want to update utilities such as the Java compiler to use the latest version. For files that should be updated, use the WebLogic Server 8.1 Administration Console to configure the appropriate attribute to use the new file or utility before proceeding to the next step.

5. If you have not already edited the server start scripts, do so now. See "Modifying Startup Scripts" on page 1-3 for detailed instructions.

**Note:** WebLogic Server 8.1 will not deploy applications that have deployment descriptor errors.

# Additional Upgrade Procedures and Information

This section contains upgrade information about the following topics:

- Apache Xalan XML Transformer
- Apache Xerces XML Parser
- Web Applications
- Web Services
- EJB 2.0
- Security
- JTA
- JMS
- JDBC

- Prepared Statement Cache Algorithm
- Pool, JTS/JTA, and RMI Connections

# Apache Xalan XML Transformer

The built-in transformer in WebLogic Server 8.1 is based on the Apache Xalan 2.2 transformer.

Use of the Xalan APIs directly has been deprecated. If you are still using those APIs and encounter difficulties, you should use the *Java API for XML Processing* (JAXP) to use XSLT.

Changes were made to Apache's Xalan code to enable Xerces and Xalan to work together. You may encounter problems if you use Xalan from Apache, because it will not include these changes.

In general, it is best to use JAXP and to port any vendor-specific code to a neutral API such as JAXP for SAX, DOM, and XSL processing.

Previous versions of WebLogic Server included the unmodified versions of the Xerces parser and Xalan transformer from www.apache.org in the *WL_HOME*\server\ext\xmlx.zip file. The ZIP file no longer includes these classes and interfaces. Download the unmodified Xerces parser and Xalan transformer directly from the Apache Web site.

# Apache Xerces XML Parser

The Xerces XML parser used in Weblogic Server 8.1 is stricter than the one used in WebLogic Server 7.0, and may refuse to parse erroneous XML that was accepted by the WebLogic Server 7.0 parser.

The built-in XML parser for WebLogic Server 8.1 is based on the Apache Xerces 1.4.4 parser. The parser implements version 2 of the SAX and DOM interfaces. Users who used older parsers that were shipped in previous versions may receive deprecation messages.

WebLogic Server 8.1 also includes the WebLogic FastParser, a high-performance XML parser specifically designed for processing small to medium size documents, such as SOAP and WSDL files associated with WebLogic Web services. Configure WebLogic Server to use FastParser if your application handles mostly small to medium size (up to 10,000 elements) XML documents. For more information, see Administering WebLogic Server XML.

Previous versions of WebLogic Server included the unmodified versions of the Xerces parser and Xalan transformer from www.apache.org in the *WL_HOME*\server\ext\xmlx.zip file. The ZIP file no longer includes these classes and interfaces. Download the unmodified Xerces parser and Xalan transformer directly from the Apache Web site.

To see the explicit differences between the old and new parsers, use a tool contained in the apichange.zip file located on dev2dev Online.

# Web Applications

- The `weblogic.management.runtime.ServletRuntimeMBean.getName()` API (in WebLogic Server 6.0) has changed to `weblogic.management.runtime.ServletRuntimeMBean.getServletName()` in WebLogic Server 6.1 and 7.0. You will have to update your source code and recompile if you are using this interface.
- With Java Servlet Specification 2.3, authorization-on-forward is no longer default behavior. To obtain authorization when you forward to a secure resource, add `<check-auth-on-forward>` to the `weblogic.xml` file.
- Servlet Request and Response objects have a new API. Some serializable, lightweight implementations of these may no longer compile without implementing the new API. It is strongly recommended that you use the new Servlet 2.3 model and substitute your implementations of Servlet Request and Response objects. If you did this in WebLogic Server 6.0, you were probably relying on the undocumented, internal implementations of these objects. WebLogic Server 7.0 and later supports Servlet 2.3, so you should be able to take advantage of the new ServletRequest/ResponseWrapper objects.
- HTML Kona, marked deprecated in previous releases, has been removed.

# Web Services

When upgrading Web Services from WebLogic Server 7.0 to WebLogic Server 8.1, you will need to re-run servicegen to regenerate the webservice deployment units.

For detailed information on upgrading a 7.0 WebLogic Web service to 8.1, see *Upgrading 7.0 WebLogic Web Services to 8.1* at http://e-docs.bea.com/wls/docs81b/webServices/migrate.html.

For examples of using JAX-RPC to invoke WebLogic Web services, see *Invoking Web Services* at http://e-docs.bea.com/wls/docs81b/webServices/client.html.

For general information on the differences between 7.0 and 8.1 Web services, see *Overview of WebLogic Web Services* at http://e-docs.bea.com/wls/docs81b/webServices/overview.html.

# EJB 2.0

To see a complete listing of the specification changes, you can view and download the *EJB 2.0 final specification* at http://java.sun.com/products/ejb/2.0.html.

## Run DDConverter Before Using New Features

Before using any features new in this release, be sure to run the DDConverter tool to convert existing EJB deployment descriptors to 8.1 descriptors. For more information, see the discussion of the DDConverter tool at http://e-docs.bea.com/wls/docs81b/ejb/EJB_tools.html#ddconverter.

## New Default Values for EJB Deployment Descriptor Elements

As of this release, there are new default values for the following EJB deployment descriptor elements:

| Element | Deployment Descriptor | New Default Value |
|---|---|---|
| `enable-call-by-reference` | `weblogic-ejb-jar.xml` | `False` |

| Element | Deployment Descriptor | New Default Value |
|---|---|---|
| include-updates | weblogic-cmp-rdbms-jar.xml | The default value is False for beans that use optimistic concurrency.<br>The default value is True for beans that use other concurrency types, such as database. |
| check-exists-on-method | weblogic-cmp-rdbms-jar.xml | True |

**Note:** The new default values are only in effect when using the 8.1 version of the deployment descriptors. Existing beans that use pre-8.1 versions of the deployment descriptors can be deployed on the 8.1 release with no change in behavior.

For more information on EJB deployment descriptors, see The weblogic-ejb-jar.xml Deployment Descriptor at http://http://e-docs.bea.com/wls/docs81b/ejb/reference.html and The weblogic-cmp-rdbms-jar.xml Deployment Descriptor at http://http://e-docs.bea.com/wls/docs81b/ejb/EJB_reference.html.

## Inserting a New CMP Bean at Commit Time

In WebLogic Server 7.0, to have the EJB container perform bulk inserts, you set the weblogic-cmp-rdbms-jar.xml element delay-database-insert-until to commit.

As of the current release, the commit value is no longer supported for delay-database-until-insert. To permit bulk inserts, set the new weblogic-cmp-rdbms-jar.xml element enable-batch-operations to true.

enable-batch-operations takes effect on the jar level, so you need only set this tag once per jar. By contrast, delay-database-insert-until had to be set for every bean.

For more information on entity batch operations see Entity Batch Operations at http://http://e-docs.bea.com/wls/docs81b/ejb/EJB_environment.html#entity_batch_operations.

# Connectors

## New weblogic-ra-xml Deployment Descriptor Elements

- The `<highest-num-waiters>` element is the maximum number of waiters that can concurrently block waiting to reserve a connection from the pool.

  This is an optional element.

  Default Value:0

- The `highest-num-unavailable` element is the maximum number of physical connections (Managed Connections) in the pool that can be made unavailable (to the application) for purposes like refreshing the connection etc. Note that in cases likes the backend system being unavailable, this specified value could be exceeded due to factors outside the pools control.

  This is an optional element.

  Default Value:0

- The `<connection-creation-retry-frequency-seconds>` element is the periodicity of retry attempts by the pool to establish connections.

  This is an optional element.

  Default Value:0

- The `<connection-reserve-timeout-seconds>` element is the number of seconds after which the call to reserve a connection from the pool will timeout.

  This is an optional element.

  Default Value:-1

- The `<check-on-create-enabled>` element is the flag to enable testing of newly created connections.

  This is an optional element.

  Value Range:true|false

  Default Value:false

- The `<check-on-release-enabled>` element is the flag to enable testing of connections when they are being released back into the pool.

This is an optional element.

Value Range:true|false

Default Value:false

- The <check-on-reserve-enabled> element is the flag to enable testing of connections when they are being reserved.

  This is an optional element.

  Value Range:true|false

  Default Value:false

- The <test-frequency-seconds> element is the periodicity at which connections in the pool are tested.

  This is an optional element.

  Default Value:0

- The <max-num-match-retries> element specifies the number of times that the matchManagedConnections() method is retried on the resource adapter's ManagedConnection object before WebLogic Server creates a new Managed Connection using the ManagedConnection object's createManagedConnection() method is called.

  This is an optional element.

  Default Value: 0

- The <match-connections-supported> element indicates whether or not the resource adapter supports the ManagedConnectionFactory.matchManagedConnections() method.

  If the resource adapter does not support this method (i.e. always return null for this method) then WebLogic Server will bypass this method call during a connection request.

  This is an optional element.

  Value Range: true|false

  Default Value: true

- The <shrink-frequency-seconds> element identifies the amount of time (in seconds) the Connection Pool Management will wait between attempts to reclaim unused Managed Connections.

This is an optional element.

Default Value:900 seconds

- The `<inactive-connection-timeout-seconds>` element identifies the amount of time (in seconds) a Connection handle can remain inactive. This element prevents leaks when an Application may have not closed a connection after completing usage. Inactive connections will be terminated as soon as they are detected.

  This is an optional element.

  Default Value:0

## Deployment Descriptor Editing

WebLogic Server 6.1 and 7.0 provided a Deployment Descriptor Editor in the Administration Console. This editor provided the capability to view the current ra.xml and weblogic-ra.xml deployment descriptor elements for resource adapters. Descriptor elements could also be modified through this editor and persisted to the descriptor files within the resource adapter. However, the new element values did not take effect dynamically at runtime. The resource adapter needed to be redeployed for the new descriptor elements to take effect.

WebLogic Server 8.1 deprecated the Deployment Descriptor Editor in the Administration Console and replaced it with a Descriptor tab. In this tab, descriptor elements can be viewed, modified and persisted to the descriptor file within the resource adapter in the same manner that they were using the Deployment Descriptor Editor. However, these descriptor elements take place dynamically at runtime without requiring the resource adapter to be redeployed. The descriptor elements contained in the Descriptor tab are limited to only those descriptor elements that may be dynamically changed at runtime. These include the following weblogic-ra.xml elements:

## Last-resource Commit Optimization

Normally, if a client wants operations on multiple resource adapter connections to participate in a global / XA transaction then the resource adapters involved are required to support XATransaction. However, resource adapters that only support Local Transactions may also be involved in a global / XA transaction, but in a limited manner, since they do not receive 2-phase commit messages from the transaction manager.

In WebLogic Server 8.1, if the server detects a Local Transaction capable resource adapter connection in a global transaction, the transaction manager first issues prepare messages to the XAResources involved in the transaction. Then, after all XAResources have prepared successfully, the operation on the Local Transaction capable resource adapter is performed. If the operation is successful, the global transaction is committed. If the operation fails, then the global transaction is rolled back. This prevents the possibility of the Local Transaction resource adapter's commit failing after a XA resource has already been committed.

**Notes:** This optimization allows multiple XAResources but not more than one Local Transaction capable resource adapter to participate in a global transaction. Attempts to include more than one Local Transaction capable resource adapter in a global / XA transaction will fail with an exception.

Previous to WebLogic Server 8.1, there was no restriction on the number of Local Transaction capable resource adapters that could be involved in a global / XA transaction. Also, the ordering of the prepares and commits was not coordinated specially for Local Transaction capable resource adapters as it now is in WebLogic Server 8.1. It was previously possible for a call to commit on a Local Transaction capable resource adapter to fail after some of the XA Transaction resource adapters had already been successfully committed. This created the case where some resources would be committed and some rolled back for a global / XA transaction. The optimization resolves this previous issue.

## weblogic-ra.xml Deployment Descriptor Changes

The `<connection-maxidle-time>` has been deprecated. It is replaced by inactive-connection-timeout-seconds.

The `<connection-maxidle-time>` element identifies the amount of time (in seconds) a Connection handle can remain idle. This element prevents leaks when an Application may have not closed a connection after completing usage. Idle connections will only be terminated in the case where the connection pool becomes full, and a new connection request is about to fail because of this.

This is an optional element.

Default Value:0

`<shrink-period-minutes>` has been deprecated. It has been replaced by shrink-frequency-seconds.

The <shrink-period-minutes> element identifies the amount of time the Connection Pool Management will wait between attempts to reclaim unused Managed Connections.

This is an optional element.

Default Value:15

<connection-cleanup-frequency> and connection-duration-time were deprecated in WebLogic Server 7.0. In WebLogic Server 8.1, these elements have no effect if specified. These elements have been replaced by inactive-connection-timeout-seconds.

The connection-cleanup-frequency element identifies the amount of time (in seconds) the Connection Pool Management will wait between attempts to destroy Connection handles which have exceeded their usage duration. This element, used in conjunction with connection-duration-time, prevents connection leaks when an Application may have not closed a connection after completing usage.

This is an optional element.

Default Value:-1

The <connection-duration-time> element identifies the amount of time (in seconds) a Connection handle can be active. This element, used in conjunction with connection-cleanup-frequency, prevents leaks when an Application may have not closed a connection after completing usage.

This is an optional element.

Default Value:-1

## Connection Proxy Wrapper

When a connection request is made, WebLogic Server returns to the client (via the resource adapter) a Proxy object that wraps the connection object. WebLogic Server uses this proxy to provide features that assist applications using the WebLogic Server implementation of the J2EE Connector Architecture. These features include (1) connection leak detection capabilities and (2) late XAResource enlistment when a connection request is made before starting a global transaction that uses that connection.

If the connection object returned from a connection request is cast as a Connection class, a ClassCastException can occur. This exception is caused by either (1) the resource adapter doing the cast or (2) the client doing the cast during a connection request.

In WebLogic Server 8.1, an attempt is made to detect the ClassCastException caused by the resource adapter (case (1), above). If the server detects that this cast is failing, it turns off the proxy wrapper feature and proceeds by returning the connection object during a connection request (unwrapped). The server logs a warning message to indicate that the proxy wrapper has been turned off.

**Note:** When this occurs, connection leak detection and late XAResource enlistment features are also turned off (but currently no indication of this is given in the console monitoring).

**Note:** The attempt for WebLogic Server to detect the ClassCastException is made by acting as a client using container-managed security. This requires the resource adapter to be deployed with security credentials defined.

If the client is doing the cast and getting a ClassCastException, the customer (client) code can be modified as follows:

Example: The client is casting the connection object to MyConnection

Instead of MyConnection being a class that implements the resource adapter's Connection interface, modify it to be an interface that extends Connection.

Implement a MyConnectionImpl class that implements the MyConnection interface.

## Application Container Managed Security Log Messages

Messages indicating whether the client is using application or container managed security are no longer written to server log.

## Classloader and Packaging Issues

In WebLogic Server 8.1 each resource adapter will have its own classloader to load its classes in the same manner as Web applications. With this change in affect, components like Web applications and EJBs that are packaged along with a resource adapter in an application archive (.ear file), do not have visibility into the resource

adapter's classes. If such visibility is required then the classes of the resource adapter may be placed in APP-INF/classes or jarred and placed in APP-INF/lib of the application archive.

# Deployment

WebLogic Server 8.1 will not deploy an application that has any errors in its deployment descriptor. Previous versions of WebLogic Server would deploy an application that had errors in its deployment descriptor. For example, if your 7.0 application was missing a reference description stanza in the deployment descriptor, the application will not deploy in the 8.1 server until you add that stanza. A typical stanza looks like:

```
<ejb-reference-description>
<ejb-ref-name>ejb/acc/Acc</ejb-ref-name>
<jndi-name>estore/account</jndi-name>
</ejb-reference-description>
```

See "Deprecated APIs and Features" on page 2-17 for information on deprecated MBean attributes and operations.

# Security

## Security Data Is Now Written to the config.xml file

The persistence model for the COMMO MBeans changed in this release of WebLogic Server. All security configuration data is now stored in the `config.xml` file. Existing security configuration data is written to the `config.xml` file when the server is initially booted. The `config.xml` file must have write permissions for this upgrade to occur.

## Web Resource Is Replaced by URL Resource

The Web resource available in previous releases of WebLogic Server has been replaced by the URL resource. If you wrote a custom Authorization provider that uses the Web resource (instead of the URL resource), enable the Use Deprecated Web Resource attribute. This attribute changes the runtime behavior of the Servlet container to use a Web resource rather than a URL resource when performing authorization.

To use an existing Web resource:

1. Expand the Security node.
2. Expand the Realms node.

   All the security realms available for the WebLogic domain are listed in the Realms table.
3. In the Realms table, click the name of the desired security realm.
4. Click the General tab.
5. Check the Use Deprecated Web Resource attribute.
6. Reboot WebLogic Server.

## Keystores Are Supported

A new keystore implementation is available in this release of WebLogic Server. The keystore retrieves trusted CAs, private keys, and server certificates from JDK keystores. The keystore implementation in this release of WebLogic Server offers several improvements:

- All JDK keystore types are supported. In WebLogic Server 7.x, only jks keystores were supported.
- The keystore retrieves the server's certificate from a keystore. In WebLogic Server 7.x, a server's certificate could only be retrieved from a file.
- Each server has its own keystore configuration.
- The keystore can be protected with a password thus adding a layer of security to the WebLogic Server deployment.

By default, WebLogic Server has SSL enabled and the server's identity and trust is established with a demonstration certificate and demonstration and standard (JDK) certificate authorities. The following sections describe how existing identity and trust mechanisms can be used with this release of WebLogic Server.

## Custom Keystore Providers Are No Longer Supported

Custom Keystore providers cannot be used with this release of WebLogic Server. Private keys and trusted CAs should be stored in keystores associated with a particular instance of WebLogic Server. For more information about configuring a keystore for a server and loading private keys and trusted CAs into the keystore, see "Configuring Keystores and SSL" in the Security section of the Administration Console online help.

## The WebLogic Keystore Provider Is Deprecated

The WebLogic Keystore provider is deprecated in this release of WebLogic Server. If you are using the WebLogic Keystore provider to store private keys and trusted CAs, the server, when first booted with this release of WebLogic Server, will update the SSL MBean in the `config.xml` file to include the following attribute:

```
IdentityAndTrustLocations="FilesorKeystoreProviders"
```

This attribute tells the server to use the 7.x SSL configuration rules instead of the new SSL rules. BEA recommends using this configuration only until you can upgrade to the keystores available in this release of WebLogic Server.

## Using Flat Files for Identity and Trust Is Deprecated

Storing identity (private keys and certificates) and trust (certificate authorities) in files is no longer supported. To upgrade to the keystores available in this release of WebLogic Server:

1. Create a trust keystore.
2. Load the trusted CAs into the keystore. Use the JDK `keytool` utility to perform this step.
3. Create an identity keystore.
4. Load the private key and certificate into the keystore. Use the WebLogic Server `ImportPrivateKey` utility to perform this step.

5. In the Administration Console, reconfigure SSL to use these trust and identity keystores.

For more information, see "Configuring Keystores and SSL" in the Security section of the Administration Console online help.

## Using SSL from Java Clients

In WebLogic Server 7.x, Java clients (fat clients) retrieved trusted CAs from a jks keystore. The following command-line argument specified the pathname to the keystore:

```
-Dweblogic.security.SSL.trustCAkeystore
```

If the command-line argument was not specified, WebLogic Server defaulted to the trusted CAs in the JDK `jre/lib/security/cacerts` keystore.

The way Java clients retrieve trusted CAs has been improved in the following ways:

- All JDK keystore types are now supported. In WebLogic Server 7.x, only jks keystores were supported.
- Keystore passwords can be specified.
- Clients can trust more types of certificate authorities. Command-line arguments specify whether to use standard trust (that is, cacerts in the JDK), demo trust (that is, cacerts in the JDK and the demo trusted CA provided by WebLogic Server), or custom trust.

This release of WebLogic Server still supports the trust command-line argument in WebLogic Server 7.0. As in WebLogic Server 7.0, WebLogic Server defaults to the trusted CAs in the JDK `jre/lib/security/cacerts` keystore.

To upgrade to the new trust mechanism in this release of WebLogic Server, specify one of the following command-line arguments.

To trust the JDK standard trusted CAs, use:

```
-Dweblogic.security.TrustKeystore=JavaStandardTrust
```

To trust the JDK standard trusted CAs and the demo trusted CAs provided by WebLogic Server, use:

```
-Dweblogic.security.TrustKeystore=DemoTrust
```

To use a custom keystore, use:

```
-Dweblogic.security.TrustKeystore=CustomTrust
-Dweblogic.security.CustomTrustKeystorePathname=keystorepathname
```

# JTA

The JTS driver has become a non-XA resource. In 7.0 the JTS driver emulated an XA driver by always responding successfully to the `XAResource.prepare` directive and issuing a one-phase commit on the JDBC connection. In WebLogic Server 8.1, the last-agent commit optimization is used where the non-XA commit operation is used by the transaction manager as the basis of the commit decision. Users may see fewer heuristic exceptions for the case where the JTS driver commit operations fails due to the last-agent protocol semantics. They may also see exceptions if the application attempts to enlist more than one JTS driver in the transaction since the non-XA support limits the number of participating non-XA resources to one.

# JMS

Configuration checking in WebLogic Server 8.1 is slightly more strict than in 7.0. In 7.0, administrators can configure multiple JMS destinations (not distributed destination memebers) with the same JNDI name. They were also allowed to configure a connection factory having the same JNDI name with default ones. Those are not allowed any more in WebLogic Server 8.1. An existing 7.0 configuration may fail to boot when upgrading to WebLogic Server 8.1.

# JDBC

Several interfaces that were previously marked deprecated have been removed:

- DB Kona
- JDBCService
- The Sybase Jdriver is no longer shipped in the box.

- The Weblogic MSSQL Jdriver is no longer supported or shipped.
- When a WebLogic Server 8.1 Client is talking to an older server, it is not possible to use Oracle extensions.

# Prepared Statement Cache Algorithm

A new prepared statement cache algorithm has been introduced. It removes the least recently used statements from the cache. The old algorithm kept a fixed number of statements in the cache (the first n, where n is the configured size of the cache).If a customer wants to get the old behavior (e.g., if they have the use case that we documented about loading the cache in a startup class), the connection pool can be configured to use the "FIXED" algorithm for the pool.

# Pool, JTS/JTA, and RMI Connections

In past releases, pool, JTS/JTA, and RMI connections were handled different. Pool connections waited for 5 seconds before timing out. JTS/JTA connections waited 10 seconds before timing out. RMI Datasource connections were non-blocking.In the WebLogic Server 8.1 release, all connections block for up to 10 seconds before timing out.

# Writable config.xml File

WebLogic Server 8.1 automatically updates configuration information read from the 7.0 `config.xml` file to include WebLogic Server 8.1 information. In order for these changes to be retained between invocations of the server, the `config.xml` file must be writable. To allow the file to be writable, make a backup copy of your `config.xml` file from your 7.0 configuration and change the file attributes.

CHAPTER

# 2 Upgrading WebLogic Server 6.x to Version 8.1

Upgrading WebLogic Server 6.x to version 8.1, under the simplest circumstances, involves changing your WebLogic Server start command scripts and environment settings.

BEA Systems recommends copying your WebLogic Server 6.x domain directory to a new directory. If you do this, ensure that relative paths to external files remain accurate.

Upgrading may also require changes specific to the subsytem being upgraded.

The following sections contain information necessary to upgrade your system from WebLogic Server 6.x to WebLogic Server 8.1:

For instructions on how to upgrade the Pet Store application from WebLogic Server 6.1 to WebLogic Server 8.1 and how to upgrade the WebLogic 6.0 and 6.1 Examples Servers to WebLogic Server 8.1, see Upgrading Example Applications to WebLogic Server 8.1.

For information on upgrading to WebLogic Platform 8.1, see the Upgrading section of the WebLogic Server FAQs.

**Note:** Throughout this document "upgrade" refers to upgrading to a later version of WebLogic Server and "port" refers to moving your applications from an earlier version of WebLogic Server to a later version.

# Upgrading Your WebLogic Server Configuration: Main Steps

Take the following steps to upgrade from WebLogic Server 6.x to WebLogic Server 8.1:

1. Make a backup copy of your 6.x domain before you begin the upgrade procedure. After you start running your domain using WebLogic Server 8.1, you cannot convert it back to 6.1.
2. Install WebLogic Server 8.1. See the the *Installation Guide*.

   **Note:** The installer will prevent you from installing the new version directly over the old version. You must select a new directory location.
3. Modify your 6.x startup scripts to work with WebLogic Server 8.1. See "Modifying Startup Scripts" on page 2-3.
4. Ensure that you have considered differences in the WebLogic Server 8.1 directory structure that may require you to make file location changes before startup. See "Understanding the WebLogic Server 8.1 Directory Structure" on page 2-4.
5. Upgrade your applications to WebLogic Server 8.1. See "Upgrading an Application from WebLogic Server 6.x to WebLogic Server 8.1" on page 2-4.

6. If necessary, perform other upgrade procedures as described in "Additional Upgrade Procedures and Information" on page 2-6.

To upgrade a cluster of servers, follow the above steps for each server and then follow the steps outlined in *Setting Up WebLogic Clusters* in *Using WebLogic Server Clusters*. In cases where you invoke an application by using RMI/T3 or RMI/IIOP, WebLogic Server 6.1 and 8.1 are interoperable. Within a domain, however, all servers must be of the same version.

For information on upgrading WebLogic Server license files, see *Upgrading Licenses from WebLogic Server 6.x* in the *Installation Guide*.

# Modifying Startup Scripts

If you used WebLogic Server startup scripts with a previous version of the product, modify them to work with WebLogic Server 8.1.

Modify the startup scripts as described here. For another example of how to modify the startup scripts, see Upgrading WebLogic Server 6.x to Version 8.1.

Modify the startup scripts as described here.

1. Modify `bea.home` property

   to point to your BEA home directory containing the `license.bea` file for WebLogic Server 8.1. For example:

   ```
   -Dbea.home=C:\bea700
   ```

2. Modify `WL_HOME`

   must point to your WebLogic Server 8.1 installation directory. For example:

   ```
   WL_HOME=c:\bea700\weblogic700
   ```

3. Modify `PATH`

   so that it includes your `%WL_HOME%` 8.1 home. For example:

   ```
   PATH=%WL_HOME%\bin;%PATH%
   ```

4. Modify `CLASSPATH`

   so that it points to the WebLogic Server 8.1 classes. For example:

```
CLASSPATH=%WL_HOME%\lib\weblogic_sp.jar;%WL_HOME%\lib\weblog
ic.jar
```

5. Modify or eliminate any WebLogic Server 6.x startup script directory structure tests. For example, if your script tries to verify a relative path, either fix the directory structure test or remove it.

WebLogic Server 8.1 installs the JVM, JDK 1.4.1, with the server installation. The `setenv.sh` scripts provided with the server all point to the JVM. The latest information regarding certified JVMs is available at the *Certifications Page*.

# Understanding the WebLogic Server 8.1 Directory Structure

The directory structure in WebLogic Server 8.1 is different from that of 6.x. For complete information on the updated directory structure see *Understanding the WebLogic Server Directory Structure* in *Performing Post-Installation Tasks* in the *Installation Guide*.

If you are booting your WebLogic Server 6.x domain with the WebLogic Server 8.1 environment, the new directory structure is created automatically. However, if you have custom tools or scripts that rely on the WebLogic Server 6.x domain directory structure, you need to update those tools relative to the new directory structure. Similarly, if you have a scripted tool for creating domains in the WebLogic Server 6.x environment, you will have to change those scripts. It is best to use the *Configuration Wizard* which can be scripted.

# Upgrading an Application from WebLogic Server 6.x to WebLogic Server 8.1

The following steps upgrade an application from WebLogic Server 6.1 to WebLogic Server 8.1:

- Install WebLogic Server 8.1
- Create or select a directory for your application's WebLogic Server 8.1 configuration domain
- Copy your configuration information to the new domain directory
- Make sure that relative paths to external files are correct.
- Edit server start scripts.

This section describes these steps in detail.

1. Install WebLogic Server 8.1, if you have not already done so. See the *Installation Guide* for instructions.

   **Note:** Installing the new version in the exact location of the old version is explicitly prohibited by the installer.

2. Create or select a directory for the WebLogic Server 8.1 domain that will house all the configuration information for your application after you have upgraded it.

   Each 6.1 and 8.1 domain must have its own directory, because it is not possible to have multiple `config.xml` files in the same directory.

   If your 6.1 `/config/domain` directory is not located in the WebLogic Server 6.1 installation, you can re-use your WebLogic 6.1 location for WebLogic Server 8.1.

3. If you are not reusing the directory your application used for its WebLogic Server 6.1 domain, copy the application's `/config/`*`domain`* directory to the directory location you chose or created for the WebLogic Server 8.1 domain.

   Tip: You can delete any directories that begin with a dot ("**.**"), which are files or directories that WebLogic Server has created for its own use.

4. Make the relative paths in the application's deployment descriptor files (`web.xml`, `weblogic.xml`, `ejb-jar.xml`, `weblogic-ejb-jar.xml`, `application.xml`, and others) point to the actual locations of any external files to which they refer.

   WebLogic Server configurations rely on a number of files that may be stored on the file system. Typically, these files are persistence repositories (log files, file-based repositories, etc.) or utilities (Java compiler). These files can be configured using fully qualified or relative paths.

If all external files are defined using relative paths and are located in or below the domain directory, skip the remainder of this step.

For external files that are defined using relative paths that are located outside the domain directory, re-create the directory structure relative to the new config directory and copy the associated files into the new directories. For external files that are defined using fully qualified paths, determine whether it is appropriate to re-use these files in the WebLogic Server 8.1 deployment.

For example, log files and persistence stores can be re-used; however, you may want to update utilities such as the Java compiler to use the latest version. For files that should be updated, use the WebLogic Server 8.1 Administration Console to configure the appropriate attribute to use the new file or utility before proceeding to the next step.

5. If you have not already edited the server start scripts, do so now. See "Modifying Startup Scripts" on page 2-3 for detailed instructions.

**Note:** WebLogic Server 8.1 will not deploy applications that have deployment descriptor errors.

# Additional Upgrade Procedures and Information

The following sections provide additional information that may be useful about deprecated features, upgrades, and the important changes that have been made in WebLogic Server 8.1.

**Note:** WebLogic Server 8.1 uses PointBase 4.2 as a sample database and does not bundle the Cloudscape database.

# Apache Xalan XML Transformer

The built-in transformer in WebLogic Server 8.1 is based on the Apache Xalan 2.2 transformer.

Use of the Xalan APIs directly has been deprecated. If you are still using those APIs and encounter difficulties, you should use the *Java API for XML Processing* (JAXP) to use XSLT.

Changes were made to Apache's Xalan code to enable Xerces and Xalan to work together. You may encounter problems if you use Xalan from Apache, because it will not include these changes.

In general, it is best to use JAXP and to port any vendor-specific code to a neutral API such as JAXP for SAX, DOM, and XSL processing.

Previous versions of WebLogic Server included the unmodified versions of the Xerces parser and Xalan transformer from www.apache.org in the *WL_HOME*\server\ext\xmlx.zip file. The ZIP file no longer includes these classes and interfaces. Download the unmodified Xerces parser and Xalan transformer directly from the Apache Web site.

# Apache Xerces XML Parser

The built-in XML parser for WebLogic Server 8.1 is based on the Apache Xerces 1.4.4 parser. The parser implements version 2 of the SAX and DOM interfaces. Users who used older parsers that were shipped in previous versions may receive deprecation messages.

WebLogic Server 8.1 also includes the WebLogic FastParser, a high-performance XML parser specifically designed for processing small to medium size documents, such as SOAP and WSDL files associated with WebLogic Web services. Configure WebLogic Server to use FastParser if your application handles mostly small to medium size (up to 10,000 elements) XML documents.

Previous versions of WebLogic Server included the unmodified versions of the Xerces parser and Xalan transformer from www.apache.org in the *WL_HOME*\server\ext\xmlx.zip file. The ZIP file no longer includes these classes and interfaces. Download the unmodified Xerces parser and Xalan transformer directly from the Apache Web site.

To see the explicit differences between the old and new parsers, use a tool contained in the apichange.zip file located on dev2dev Online.

# Applications Directory

In WebLogic Server 6.1, 7.0, and 8.1 there is a division between runtime modes. The two modes are "development" and "production." The runtime mode is selected using a command line parameter when starting the Weblogic Server (`-Dweblogic.ProductionModeEnabled=true | false`). If this parameter is not set, the server runs in development mode. In development mode the server behavior is

consistent with WebLogic Server 6.0. In production mode, however, the auto-deployment feature is disabled. Deployment units in the applications directory that are not explicitly deployed in the configuration repository (`config.xml`) will not be automatically deployed.

# Deployment

WebLogic Server 8.1 uses a two-phase deployment model. For more information on this deployment model and other 8.1 deployment features, see *WebLogic Server Deployment*. By default, statically configured applications use the 6.x deployment model. Deployments initiated through the console or the new `weblogic.Deployer` command line utility use the new two-phase deployment model.

WebLogic Server 8.1 will not deploy an application that has any errors in its deployment descriptor. Previous versions of WebLogic Server would deploy an application that had errors in its deployment descriptor. For example, if your 6.x application was missing a reference description stanza in the deployment descriptor, the application will not deploy in the 8.1 server until you add that stanza. A typical stanza looks like:

```
<ejb-reference-description>
<ejb-ref-name>ejb/acc/Acc</ejb-ref-name>
<jndi-name>estore/account</jndi-name>
</ejb-reference-description>
```

Using WebLogic Server 8.1, you can no longer deploy through the console using the 6.x protocol. As a result, you must use the new deployment APIs. If your application is previously deployed in 6.x and you're just starting your server, the applications will get deployed with one-phase deployment. The `weblogic.deploy` and `weblogic.refresh` command line utilities and the `weblogic.management.tools.WebAppComponentRefreshTool`, and are deprecated in 8.1.

See "Deprecated APIs and Features" on page 2-17 for information on deprecated MBean attributes and operations.

The applications in the applications directory in development mode on the Administration Server are now staged. In 6.x, they were not. For more information see *Application Staging* in the section called *Two-PhaseDeployment* in *Developing WebLogic Server Applications*.

# EJB 2.0

The EJB 2.0 specification has changed substantially between WebLogic Server 6.0 and WebLogic Server 8.1, and somewhat between WebLogic Server 6.1 and WebLogic Server 8.1.

Some of the prominent changes are listed here. To see a complete listing of the specification changes from WebLogic Server 6.0 to WebLogic Server 8.1, you can view and download the *EJB 2.0 final specification* at http://java.sun.com/products/ejb/2.0.html.

For more information about the changes between WebLogic Server 6.0 and WebLogic Server 6.1, see *EJB Enhancements in WebLogic Server* in *Introducing WebLogic Server Enterprise JavaBeans* in the WebLogic Server 6.1 documentation. EJB 1.1 beans that worked in WebLogic Server 6.x should work just as well in WebLogic Server 8.1 with no alteration.

You may have to make the following changes to EJB 2.0 beans:

- If your deployment descriptor contains a 6.0 element that has a different name in 8.1, you have to manually change the name in your deployment descriptor. The following are some examples of element names that you may need to change in 8.1:
  - In 8.1, the name of the element that is used to identify a particular EJB that participates in a relationship is `relationship-role-source`. In 6.0, the element name was `role-source`.
  - In 8.1, the name of the element that specifies whether the destination is a queue or a topic is `destination-type`. In 6.0, the element name was `jms-destination-type`.
  - In 8.1, the name of the element that specifies whether the destination is a queue or a topic is `run-as`. In 6.0, the element name was `run-as-specified-identity`.
- In 8.1, `EJB-QL` queries require a `SELECT` clause.
- All EJB 2.0 CMP beans must have an `abstract-schema-name` element specified in their `ejb-jar.xml` in WebLogic Server 8.1.

Other major changes that resulted from the EJB 2.0 specification changes are as follows:

- There were no local interfaces in 6.0, but they exist in 6.1 and 7.0 and 8.1.
- Remote relationships do not exist in 6.1 and 7.0 and 8.1.
- The new implementation of read-only beans in Acadia does not require `NOT_SUPPORTED` for a transaction attribute. It also doesn't do exclusive locking, but gives each transaction its own instance of the read-only bean to use.
- In pre-2.0 final versions of our EJB implementation you could refer to a CMP or CMR field in a `EJB QL` query without using a qualified path. However, as of the final specification, all `EJB QL` queries must have a qualified path.
- The new implementation of read-only beans in Acadia does not require `NOT_SUPPORTED` for a transaction attribute. The new implementation also does not do exclusive locking, but gives each transaction its own instance of the read-only bean to use.

If you have trouble with a servlet within the scope of application deployment see "Deployment" on page 2-9.

## weblogic.management.configuration.EJBComponentMBean Changes

Beginning in Weblogic Server 6.1 and continuing in WebLogic Server 8.1, the interface `weblogic.management.configuration.EJBComponentMBean` changed so that it extends both `ComponentMBean` and `EJBContainerMBean`. Any methods that implement `EJBComponentMBean` (for example, `getVerboseEJBDeploymentEnabled`) must be changed to support `EJBContainerMBean` when you upgrade from WebLogic Server 6.0 to 8.1.

## max-beans-in-cache Parameter

In WebLogic Server 8.1 the `max-beans-in-cache` parameter controls the maximum number of beans in the cache for Database concurrency. In earlier WebLogic Server versions, `max-beans-in-cache` was ignored; the size of the cache was unlimited. You may need to increase the size of this parameter.

### Fully Qualified Path Expressions

In an EJB QL Query on WebLogic Server 8.1, all path-expressions must be fully qualified. This is a change from WebLogic Server 6.x. If you see an ejbc error while running ejbc on WebLogic Server 8.1 using a 6.x EJB, you need to correct either your `ejb-ql` elements in your `ejb-jar.xml` file or your `weblogic-ql` elements in your `weblogic-cmp-jar.xml` file. For example:

- WebLogic Server 6.x would allow the following query to compile:

  ```
  SELECT address FROM CustomerBean AS c WHERE zip = ?1
  ```

- For WebLogic Server 8.1 to allow the same query to be compiled, the address and zip fields must be qualified:

  ```
  SELECT c.address FROM CustomerBean AS c WHERE c.zip = ?1
  ```

## jCOM

For information about upgrading from WebLogic jCOM 6.1 to WebLogic jCOM 8.1 see *Upgrading Considerations* in *Programming WebLogic jCOM*.

## JMS

WebLogic Server 8.1 supports the *JavaSoft JMS specification version 1.0.2*.

All WebLogic JMS 6.x applications are supported in WebLogic JMS 8.1. However, if you want your applications to take advantage of the new highly available JMS features, you will need to configure your existing physical destinations (queues and topics) to be part of a single distributed destination set. For more information on using JMS distributed destinations, see *Using Distributed Destinations* in *Programming WebLogic JMS*.

For more information on porting your WebLogic JMS applications, see *Porting WebLogic JMS Applications* in *Programming WebLogic JMS*.

# JMX

All public WebLogic Server 6.x MBeans and attributes are supported in WebLogic Server 8.1. However, if you are employing internal MBeans or attributes, you may encounter porting issues.

See "Deprecated APIs and Features" on page 2-17 for information on deprecated MBean attributes and operations.

# Jolt Java Client

Jolt users will need to upgrade to Jolt Java Client 8.0.1 to enable BEA Tuxedo services for the Web using BEA WebLogic Server 8.1. The Jolt Java Client 8.0.1 is available from the *BEA Product Download Center* at http://commerce.bea.com/downloads/products.jsp. Follow the link to BEA WebLogic Server 7.0 and select the Jolt Java Client 8.0.1 from the Modules for WebLogic Server list.

# JSP

Due to changes in the JSP specification, null request attributes return the string "null" instead of an empty string. For more information, see information about issue 074843 in the *Release Notes*.

# Managed Servers

A Managed Server running WebLogic Server 6.x cannot obtain its configuration and boot using an Administration Server running WebLogic Server 8.1. Make sure that you do not copy the `running-managed-servers.xml` file from your WebLogic Server 6.x installation directory to your WebLogic Server 8.1 installation directory.

# MBean API Change

In previous versions, the `weblogic.management.Admin` class had the static method, `getActiveDomain()`. In Weblogic Server 8.1, `getActiveDomain()` is no longer static. In its place, use
`Admin.getInstance().getAdminMBeanHome().getActiveDomain();`

For an example of how to use this alternative API, see *Determining the Active Domain and Servers* in *Programming WebLogic Server JMX Services*.

# Servlets

Update your web.xml file so that it uses the following new classes:

`weblogic.servlet.proxy.HttpClusterServlet`

instead of

`weblogic.servlet.internal.HttpClusterServlet`

and

`weblogic.servlet.proxy.HttpProxyServlet`

instead of

`weblogic.t3.srvr.HttpProxyServlet`

If you have trouble with a servlet within the scope of application deployment see "Deployment" on page 2-9.

# Thread Pool Size

In WebLogic Server 6.0, the number of worker threads was specified via the `ThreadPoolSize` parameter on the server MBean. Starting in WebLogic Server 6.1, the number of worker threads is defined via an `ExecuteQueue` on the Server MBean.

WebLogic Server 8.1 provides a porting path for this parameter, so that if it is specified in the `config.xml` file, or if it is passed to the client or server on the command line (`-Dweblogic.ThreadPoolSize=<xx>`), WebLogic Server ports your `ThreadPoolSize` to the `ThreadCount` setting automatically.

Setting the number of worker threads:

In WebLogic Server 6.x:

```
<Server
Name="myserver"
ThreadPoolSize="23"
...
/Server>
```

Starting in WebLogic Server 7.0:

```
<Server
Name="myserver"
... >
<ExecuteQueue
Name="default"
ThreadCount="23" />
/Server>
```

To change the thread count value via the Console:

1. In the console, select Servers > myServer > Monitoring > .
2. Click on Monitor all Active Queues.
3. Click on "default" queue (a list of threads and what they are doing appears).
4. Click on Configure Execute Queues (at the top of the page).
5. Click on "default" queue.
6. Enter the number of threads associated with this server.
7. Restart the server to make changes take effect.

# Web Applications

The `weblogic.management.runtime.ServletRuntimeMBean.getName()` API (in WebLogic Server 6.0) has changed to `weblogic.management.runtime.ServletRuntimeMBean.getServletName()` in WebLogic Server 6.1 and 7.0 and 8.1. You will have to update your source code and recompile if you are using this interface.

With Java Servlet Specification 2.3, authorization-on-forward is no longer default behavior. To obtain authorization when you forward to a secure resource, add `<check-auth-on-forward>` to the `weblogic.xml` file.

Servlet Request and Response objects have a new API. Some serializable, lightweight implementations of these may no longer compile without implementing the new API. It is strongly recommended that you use the Servlet 2.3 model and substitute your implementations of Servlet Request and Response objects. If you did this in WebLogic Server 6.0, you were probably relying on the undocumented, internal implementations of these objects. WebLogic Server 8.1 supports Servlet 2.3, so you should be able to take advantage of the new ServletRequest/ResponseWrapper objects.

# WebLogic Server Clusters on Solaris

Certain applications (heavy EJB apps) deployed in a WebLogic Server cluster on Solaris will perform better using the client JVM rather than the server JVM. This is especially true under heavy loads.

# Web Services

Due to changes in the Web service runtime system and architecture between versions 6.1 and 8.1 of WebLogic Server, you must upgrade Web services created in version 6.1 to run on version 8.1.

The WebLogic Web services client API included in WebLogic Server 6.1 of has been deprecated and you cannot use it to invoke 8.1 Web services. WebLogic Server 8.1 includes a new client API, based on the Java API for XML-based RPC (JAX-RPC).

For detailed information on upgrading a 6.1 WebLogic Web service to 8.1, see *Upgrading 6.1 WebLogic Web Services to 8.1* at http://e-docs.bea.com/wls/docs81b/webServices/migrate.html.

For examples of using JAX-RPC to invoke WebLogic Web services, see *Invoking Web Services* at http://e-docs.bea.com/wls/docs81b/webServices/client.html.

For general information on the differences between 6.1 and 8.1 Web services, see *Overview of WebLogic Web Services* at http://e-docs.bea.com/wls/docs81b/webServices/overview.html.

# Writable config.xml File

WebLogic Server 8.1 automatically updates configuration information read from the 6.x `config.xml` file to include WebLogic Server 8.1 information. In order for these changes to be retained between invocations of the server, the `config.xml` file must be writable. To allow the file to be writable, make a backup copy of your `config.xml` file from your 6.x configuration and change the file attributes.

# Deprecated APIs and Features

- WebLogic Time Services is deprecated and should be replaced by JMX Timer Service. For documentation of JMX Timer Service, see *Interface TimerMBean* and *Class Timer*.
- WebLogic Workspaces
- Zero Administration Client (ZAC) is deprecated in this release of WebLogic Server 8.1. It is replaced by JavaWebStart.
- WebLogic Enterprise Connectivity (WLEC) is deprecated in this release of WebLogic Server 8.1. Tuxedo CORBA applications using WLEC should migrate to WebLogic Tuxedo Connector. For more information, see *WebLogic Tuxedo Connector*.
- `weblogic.deploy` is deprecated in this release of WebLogic Server 8.1 and has been replaced by `weblogic.Deployer`. For more information, see *Deployment Tools and Procedures* in *WebLogic Server Deployment*.
- `weblogic.management.tools.WebAppComponentRefreshTool` and `weblogic.refresh` are both deprecated in this release of WebLogic Server 8.1. They have been replaced by `weblogic.Deployer`.
- If you did programmatic deployment or used the `weblogic.Admin` command in order to create application and component MBeans, set the attributes on the MBeans, and invoke operations on those MBeans to cause them to get deployed in the system, the following MBean attributes and operations have been deprecated:

  Deprecated `ApplicationMBean` operations:

  `deploy`

`load`

`undeploy`

Deprecated `ApplicationMBean` attributes:

`LastModified`

`LoadError`

`isDeployed`

Please refer to the WebLogic Server 8.1 *javadocs* for the `ApplicationMBean` interface to see what has replaced these attributes and operations.

# Removed APIs and Features

WebLogic Enterprise Connectivity (WLEC) examples have been removed.

# 3 Upgrading WebLogic Server 4.5 and 5.1 to Version 8.1

Upgrading WebLogic Server 4.5 and 5.1 to version 8.1 is a multi-step process that involves defining web applications and converting your `weblogic.properties` file(s) into a new XML file format. There are also several specification changes that affect the upgrade process. This chapter addresses the majority of upgrade issues, but may omit issues that are unique to a specific environment.

The following sections provide procedures and other information you need to upgrade your system from WebLogic Server 4.5 or 5.1 to WebLogic Server 8.1; to port your applications from WebLogic Server 4.5 or 5.1 to WebLogic Server 8.1; and deploy these applications. Instructions apply to upgrades from both WebLogic Server 4.5 and 5.1 to WebLogic Server 8.1.

**Note:** Throughout this document the word, upgrade, is used to refer to upgrading to a later version of WebLogic Server and the word, port, is used to refer to moving your applications from an earlier version of WebLogic Server to a later version.

# Upgrading Your WebLogic Server Configuration: Main Steps

Take the following steps to upgrade from WebLogic Server 4.5 or 5.1 to WebLogic Server 8.1:

1. Install WebLogic Server 8.1. See the *Installation Guide*.

   Prior to WebLogic Server 6.0, a separate download was required if you wanted to get 128-bit encryption instead of 56-bit encryption. WebLogic Server 8.1 has a single download for both 56-bit encryption and 128-bit encryption. For details of how to enable 128-bit encryption see *Enabling 128-Bit Encryption* in the *Installation Guide*.

   **Note:** Installing the new version in the exact location of the old version is explicitly prohibited by the installer.

2. Upgrade your server license files. For instructions on how to convert your licenses to the new format, see "Upgrading WebLogic Server License Files" on page 3-4.

3. Convert your `weblogic.properties` file. For instructions on how to convert your `weblogic.properties` file, see "Converting the weblogic.properties File to XML Files" on page 3-5 and the *Console Help* documentation.

4. Enter a name for you new Administration Server in the provided window in the Console. This name is used as the Server Name for the Administration Server. If the name is not specified, the name will be `myserver` by default.
5. Enter a directory location for the resulting conversion output configuration. All files and subdirectories created as a result of the conversion of your original domain will be placed in this directory location.
6. Add classes to your Java system `CLASSPATH`. For more information see "Classloading in WebLogic Server 8.1" on page 3-7.
7. Modify your existing startup scripts to work with WebLogic 8.1. See "Modifying Startup Scripts" on page 3-8.
8. Package and port your WebLogic server-side business object implementations (referred to as Web applications beginning with WebLogic Server 6.0) to run on WebLogic 8.1. See "Converting Your Existing Applications into Web Applications" on page 3-9.
9. If you need to port EJBs, see "Porting and Converting Enterprise JavaBeans Applications" on page 3-15.
10. Upgrade JMS. Many new configuration attributes have been added to JMS since WebLogic Server 4.5. For more information, see "Upgrading JMS" on page 3-23.

To upgrade a cluster of servers, follow the above steps for each server and then follow the steps outlined in *Setting up WebLogic Clusters* in *Using WebLogic Clusters*. All servers in the cluster should be running 8.1 since there is no interoperability between 4.5 and 8.1 or between 5.1 and 8.1.

**Note:** The directory structure in WebLogic Server 8.1 is different from that of 4.5 and 5.1. For complete information on the updated directory structure see *Understanding the WebLogic Server Directory Structure* in *Performing Post-Installation Tasks*.

# Upgrading WebLogic Server License Files

The Java format license file (`WebLogicLicense.class`) and the XML-format license file (`WebLogicLicense.XML`) are no longer supported. These files were used with earlier releases of WebLogic Server and must be converted to a new format. The new license file is called `license.bea`.

## Converting a WebLogicLicense.class License

If a `WebLogicLicense.class` license file is used in your existing WebLogic Server installation, perform the following tasks before you install WebLogic Server 7.x:

1. Convert the `WebLogicLicense.class` license file to a `WebLogicLicense.XML` file using the `licenseConverter` utility at `http://www.weblogic.com/docs51/techstart/utils.html #licenseConverter`.
2. Convert the `WebLogicLicense.XML` file as described in Converting a WebLogicLicense.XML License.

## Converting a WebLogicLicense.XML License

To convert a `WebLogicLicense.XML` file to a `license.bea` file (compatible with WebLogic Server 6.x and 7.x), complete the following steps. Be sure the `WebLogicLicense.XML` license file is available on the machine on which you perform this procedure.

1. Log in to the BEA Customer Support Web site at `http://websupport.beasys.com/custsupp`.
2. Click the link to update a WebLogic Server license. You may need to scroll down to see the link.
3. Browse and select the pathname for the directory containing the license file to be converted, or enter the pathname in the box provided. Then click Submit License.

4. You will receive the converted `license_wls`*xx*`.bea` file through e-mail. To update the `license.bea` file on your system, see Updating Your license.bea File in the *Installation Guide* at `http://e-docs.bea.com/wls/docs81b/install/instlic.html#update_license`.

# Converting the weblogic.properties File to XML Files

Prior to WebLogic Server 6.0, WebLogic Server releases used a `weblogic.properties` file to configure applications. In WebLogic Server 8.1, configuration of applications is handled through XML descriptor files and the Administration Console. Converting a `weblogic.properties` file to the `config.xml` file creates a new domain for your applications and adds XML files that define how your applications are set up. BEA Systems recommends that you change the name of the default domain name that is created during the conversion so that the name is relevant to your work.

The `config.xml` file is an XML document that describes the configuration of an entire Weblogic Server domain. The `config.xml` file consists of a series of XML elements. The `domain` element is the top-level element, and all elements in the domain are children of the domain element. The `domain` element includes child elements, such as the `server`, `cluster`, and `application` elements. These child elements may have children themselves. Each element has one or more configurable attributes.

The `weblogic.xml` file contains WebLogic-specific attributes for a Web application. You define the following attributes in this file: HTTP session parameters, HTTP cookie parameters, JSP parameters, resource references, security role assignments, character set mappings, and container attributes.

The deployment descriptor `web.xml` file is defined by the servlet 2.3 specification from Sun Microsystems. The `web.xml` file defines each servlet and JSP page and enumerates enterprise beans referenced in the Web application. This deployment descriptor can be used to deploy a Web Application on any J2EE-compliant application server.

Convert your existing `weblogic.properties` file to the appropriate XML file by following these steps:

1. Start the WebLogic Server 8.1 examples server. For information on starting the WebLogic Server 8.1 examples server, see *Starting the Examples, Pet Store, and Workshop Examples Servers* in *Performing Post-Installation Tasks*.

   You will be prompted for a user name and password.

2. At the home page for the WebLogic Administration Console (for example: `http://localhost:7001/console/index.jsp`) click on the "Convert `weblogic.properties`" link under the heading Helpful Tools.

3. Use the Console's links to navigate the server's file system and find the root directory of your previous version of WebLogic Server (for example: `C:\weblogic`). When you have found the correct directory, click on the icon next to it to select it.

4. If you have additional per server `weblogic.properties` files or clustering `weblogic.properties` files in other directories, select them using the provided windows. **If you have chosen the correct root directory of your previous version of WebLogic Server, your global** `weblogic.properties` **file will be converted regardless of any additional properties files that you select.**

5. Enter a name for your new domain in the provided window in the Console. Click Convert.

When you convert your properties file, the `web.xml and weblogic.xml` files for the default web application are created for you and placed inside a `domain\applications\DefaultWebApp_myserver\WEB-INF` directory. The process of converting your `weblogic.properties` file also creates the `config.xml` file located in `domain`. This file contains configuration information specific to your domain.

**Note:** The conversion utility described above specifies the Java home location in the `weblogic.xml` file. It reads this location using the `System.getProperty(java.home)`, which means that it will specify the Java home location on which WebLogic Server was started for the conversion.

- It is strongly recommended that you not edit the `config.xml` file directly. Access the configuration through the Administration Console, a command line utility, or programmatically through the configuration API. For details on

configuring WebLogic Server Web Components, see *Configuring WebLogic Server Web Components* in the *Administration Guide*.

- Security properties are stored in the `fileRealm.properties` file located in *`domain`*.
- The `weblogic.common.ConfigServicesDef` API, which provided methods to get properties out of the `weblogic.properties` file, has been removed from this version.

For more procedures for converting your `weblogic.properties` file, see the *Console Help* documentation.

For a list of which `config.xml, web.xml, or weblogic.xml` attribute handles the function formerly performed by `weblogic.properties` properties, see The weblogic.properties Mapping Table.

The startup scripts, which are generated when a `weblogic.properties` file is converted, are named:

- `startdomainName.cmd` (for Windows users)
- `startdomainName.sh` (for UNIX users)

where *`domainName`* is the name of your *`domain`* directory.

These scripts exist under the *`domain`* directory in your WebLogic Server 8.1 distribution and start the administration server in the new domain. You may need to edit this startup script to further specify your startup preferences for the domain.

See *Starting and Stopping WebLogic Servers* in the *Administration Guide* for more information on scripts and starting servers.

# Classloading in WebLogic Server 8.1

Earlier versions of WebLogic Server used the WebLogic classpath property (`weblogic.class.path`) to facilitate dynamic classloading. In WebLogic 6.0 and later, the `weblogic.class.path` is no longer needed. You can now load classes from the Java system classpath.

To include the classes that were formerly specified in `weblogic.class.path` in the standard Java system classpath, set the `CLASSPATH` environment variable, or use the `-classpath` option on the command line as in the following example:

```
java -classpath %CLASSPATH%;%MyOldClassspath% weblogic.Server
```

where *`%MyOldClasspath%`* contains only the directories that point to your old applications.

# Modifying Startup Scripts

The weblogic.properties converter created a new startup script for your new WebLogic Server 8.1 domain. If you need to edit it to specify your domain startup preferences, keep the following in mind.

- Modify the startup scripts as described in *Setting the Classpath* in the *Administration Guide*. The WebLogic classpath is no longer used; use the Java system classpath as described in the preceding section, "Classloading in WebLogic Server 8.1" on page 3-7.
- WebLogic Server 8.1 is started from the domain directory. Make sure that your startup script starts the server from the domain directory.
- It is no longer necessary to include the license file in the classpath.
- With the new management system, there is a distinction between an Administration Server and Managed Servers. Consequently, scripts that start servers must be rewritten according to how you plan to administer your servers. For the new commands and their required arguments, see *Starting and Stopping WebLogic Servers* in the *Administration Guide*.

# WebLogic Server 8.1 J2EE Application Types

Applications on J2EE-compliant servers such as WebLogic Server 8.1 are created and deployed as one of the following four types: Web Applications, Enterprise JavaBeans, Enterprise Archives, and client applications. To port your existing components to WebLogic Server 8.1, create the appropriate J2EE deployment units. For more information on J2EE deployment units, see *Deploying Web Applications as Part of an Enterprise Application* in *Assembling and Configuring Web Applications*. Web Applications are usually a collection of servlets, JSPs, and HTML files, packaged as WAR files. Enterprise JavaBeans (packaged as JAR files) are server-side Java components written according to the EJB specification. Enterprise Archives (EAR files) contain all of the JAR and WAR component archive files for an application and an XML descriptor that describes the bundled components. Client Applications are Java classes that connect to WebLogic Server through Remote Method Invocation (RMI). Later sections discuss the aforementioned J2EE deployment units in greater detail.

# Converting Your Existing Applications into Web Applications

In order to convert an application to a Web Application and upgrade it to WebLogic Server 8.1, the application's files must be placed within a directory structure that follows a specific pattern. For development, these files can be left in an exploded directory format. However, for production situations, it is highly recommended that you bundle your applications into a WAR file as a single Web Application. For more information on Web Applications see *Understanding WebLogic Server J2EE Applications*, *Assembling and Configuring Web Applications*, and *Web Applications Basics*.

The following sections provide information you need to know about porting and deploying Web Applications, including a procedure for porting a simple servlet from WebLogic Server 5.1 to WebLogic Server 8.1:

# XML Deployment Descriptors

The Web Application Deployment Descriptor (`web.xml`) file is a standard J2EE descriptor used to register your servlets, define servlet initialization parameters, register JSP tag libraries, define security constraints, and define other Web Application parameters. For detailed instructions on creating the deployment descriptor, see *Writing the web.xml Deployment Descriptor* in *Assembling and Configuring Web Applications*.

There is also a WebLogic-specific Deployment Descriptor (`weblogic.xml`). In this file you define JSP properties, JNDI mappings, security role mappings, and HTTP session parameters. The WebLogic-specific deployment descriptor also defines how named resources in the `web.xml` file are mapped to resources residing elsewhere in WebLogic Server. For detailed instructions on creating the WebLogic-specific deployment descriptor, see *Writing the WebLogic-Specific Deployment Descriptor*. This file may not be required if you do not need the preceding properties, mappings, or parameters.

Use the `web.xml` and `weblogic.xml` files, in conjunction with the Administration console, to configure your applications. The XML files can be viewed through any text editor. To edit them, simply make your changes and save the file as `web.xml` or `weblogic.xml` with the appropriate path as specified by the prescribed directory structure under "Web Applications Directory Structure" on page 3-10. See *Assembling and Configuring Web Applications* for more information. If you do not want to deploy your applications together as a single Web Application, you need to split up the XML files that have been created for you, creating the appropriate XML files specific to each Web Application. Each Web Application needs a `weblogic.xml` file and a `web.xml` file as well as whichever files you choose to put in it.

# WAR Files

A WAR file is a Web Application archive. If you have correctly followed the prescribed directory structure of a Web Application and created the appropriate `web.xml` and `weblogic.xml` files, it is strongly recommended that in production environments your applications be bundled together in a Web Application deployed as a WAR file. Once you have bundled your applications into a WAR file, it is important to remove the previously existing directory structure so that WebLogic Server only has one instance of each application.

Use the following command line from the root directory containing your Web Application to create a WAR file, replacing '*webAppName*' with the specific name you have chosen for your Web Application:

```
jar cvf webAppName.war *
```

You now have created a WAR file that contains all the files and configuration information for your Web Application.

# Deploying Web Applications

To deploy your bundled Web Applications properly you should install the application through the Administration Console. To do so, go to the console home and choose Install Applications under the Getting Started menu. Select the correct WAR file, and it will be installed automatically. If your server is running in development mode, you can also install the WAR file by placing it in the `domain\applications` directory.

Web Applications should be deployed automatically after they are installed. Check to see that they are deployed under the Deployments node in the left hand pane of the Administration Console.

You can configure certain deployment attributes for your Web Application using the Administration Console. Select the Web Applications node under the Deployments heading. Select your Web Application. Click on the appropriate tab to configure. For more information on setting attributes in the console, see the Web Applications section of the *Console Help*.

# Session Porting

WebLogic Server 8.1 does not recognize cookies from previous versions because cookie format changed with WebLogic Server 6.0. WebLogic Server will ignore cookies with the old format and create new sessions. Be aware that new sessions are created automatically.

The default name for cookies has changed from 5.1, when it was `WebLogicSession`. In WebLogic Server 8.1, cookies are named `JSESSIONID` by default.

See *weblogic.xml Deployment Descriptor Elements* in *Assembling and Configuring Web Applications*.

# JavaServer Pages (JSPs) and Servlets

This section contains information specific to JSPs and servlets that may be pertinent to your applications.

- Some changes will be necessary in code (both Java and HTML) where the code refers to URLs that may be different when servlets and JSPs are deployed in a Web Application other than the default Web Application. See *Administration and Configuration* in *Programming WebLogic Server HTTP Servlets* for more information. If relative URLs are used and all components are contained in the same Web Application, these changes are not necessary.
- Only serializable objects may be stored in a session if your application is intended to be distributable.
- You must convert your `weblogic.properties` file to `XML` attributes in `web.xml` and\or `weblogic.xml`. For additional information on this process, see the conversion section of the *Console Help*.
- Access control by an ACL has been replaced with security-constraint based access control in the web application deployment descriptor.
- Server-side-includes are not supported. You must use JSP to achieve this functionality.
- WebLogic Server 8.1 is fully compliant with the Servlet 2.3 specification.

- Update your web.xml file so that it uses the following new classes:

  `weblogic.servlet.proxy.HttpClusterServlet`

  instead of

  `weblogic.servlet.internal.HttpClusterServlet`

  and

  `weblogic.servlet.proxy.HttpProxyServlet`

  instead of

  `weblogic.t3.srvr.HttpProxyServlet`

# Porting a Simple Servlet from WebLogic Server 5.1 to WebLogic Server 8.1

The following procedure ports the simple Hello World Servlet that was provided with WebLogic 5.1 Server to WebLogic Server 8.1.

1. In WebLogic Server 8.1, create the correct directory structure, as described in *Administration and Configuration* in *Programming WebLogic Server HTTP Servlets*. This involves creating a root application directory, such as `C:\hello`, as well as a `C:\hello\WEB-INF` directory and a `C:\hello\WEB-INF\classes` directory. Place the `HelloWorld.Servlet.java` file inside the `C:\hello\WEB-INF\classes` directory.

2. Create a `web.xml` file for this servlet. If you have converted your `weblogic.properties` file, a `web.xml` file has already been created for you. If you registered HelloWorldServlet in your `weblogic.properties` file before you converted it, the servlet will be properly configured in your new `web.xml` file. An XML file can be created with any text editor. The following is an example of a basic `web.xml` file that could be used with the HelloWorldServlet.

```
<!DOCTYPE web-app (View Source for full doctype...)>
- <web-app>
- <servlet>
<servlet-name>HelloWorldServlet</servlet-name>
<servlet-class>examples.servlets.HelloWorldServlet</servlet-cla
ss>
</servlet>
- <servlet-mapping>
<servlet-name>HelloWorldServlet</servlet-name>
```

```
<url-pattern>/hello/*</url-pattern>
</servlet-mapping>
</web-app>
```

For more information on `web.xml` files, see *Writing Web Application Deployment Descriptors* in *Assembling and Configuring Web Applications*. A `weblogic.xml` file is not necessary with such a simple, stand-alone servlet as HelloWorld.

For more information on `weblogic.xml` files, see *Writing the WebLogic-Specific Deployment Descriptor* in *Assembling and Configuring Web Applications*.

3. Move the `web.xml` file from `domain\applications\DefaultWebApp_myserver\WEB-INF` to `C:\hello\WEB-INF\`.

4. Set up your development environment (see *Establishing a Development Environment* in *Developing WebLogic Server Applications* for more information) and compile the HelloWorldServlet with a command like the following:

   ```
   C:\hello\WEB-INF\classes>javac -d  . HelloWorldServlet.java
   ```

   This should compile the file and create the correct package structure.

5. The servlet can now be bundled into an archive WAR file with the following command:

   ```
   jar cvf hello.war *
   ```

   This command will create a `hello.war` file and place it inside the `C:\hello` directory.

6. To install this Web Application, start your server and open the Administration Console. Under the Getting Started menu, choose Install Applications. Browse to the newly created WAR file and click Upload.

   The servlet should now be deployed and appear under the Web Applications node under Deployments, in the left-hand pane of the console.

7. To call the servlet, type the following in your browser URL window: `http://localhost:7001/hello/hello`.

In this case `/hello/` is the context path of the servlet. This is determined by the naming of the WAR file, in this case `hello.war`. The second `/hello` was mapped in the servlet mapping tags inside the `web.xml` file.

# Porting and Converting Enterprise JavaBeans Applications

The following sections describe Enterprise Java Beans porting and conversion procedures.

## EJB Porting Considerations

Consider the following when porting Enterprise JavaBeans to WebLogic Server 8.1.

- WebLogic Server Version 8.1 supports the Enterprise JavaBeans 1.1 and 2.0 specifications.
- The XML parser is stricter with XML deployment descriptors in WebLogic 8.1 than it was in WebLogic 5.1. Some errors allowed in earlier versions are no longer permitted. This is described in *Introducing WebLogic Server Enterprise Java Beans*.
- EJB 1.1 beans are deployable in WebLogic Server 8.1. However, if you are developing new beans, it is recommended that you use EJB 2.0. EJB 1.1 beans can be converted to 2.0 using the DDConverter utility. For more information, see the *DDConverter documentation* in *Programming WebLogic Enterprise JavaBeans*.
- You can upgrade EJB 1.0 deployment descriptors to EJB 2.0 using the DDConverter utility, but first those descriptors must be upgraded to 1.1. WebLogic Server 5.1 deployment descriptors can be upgraded to 8.1 to take advantage of new features in WebLogic Server 8.1. Details on the DDConverter utility are provided in the *WebLogic Server EJB Utilities* section of *Programming WebLogic Enterprise JavaBeans*.
- The finder expressions feature of EJB 1.1 is no longer supported. This is the only non-supported feature of EJB 1.1.
- Deploying beans is described in the *Packaging EJBs for the WebLogic Server Container* section of *Programming WebLogic Enterprise JavaBeans*.

- If `ejbc` has not been run on an EJB, WebLogic Server 8.1 will run `ejbc` automatically when the bean is deployed. You do not need to compile beans with `ejbc` before deploying. If you wish to run `ejbc` during startup, you may do so. See details in *Packaging EJBs for the WebLogic Server Container* in *Programming WebLogic Enterprise JavaBeans*.
- An EJB deployment includes a standard deployment descriptor in the `ejb-jar.xml` file. The `ejb-jar.xml` must conform to either the EJB 1.1 DTD (document type definition) or the EJB 2.0 DTD.
- An EJB deployment needs the `weblogic-ejb-jar.xml` file, a WebLogic Server-specific deployment descriptor that includes configuration information for the WebLogic Server EJB container. This file must conform to the WebLogic Server 5.1 DTD or the WebLogic Server 8.1 DTD.
- In order to specify the mappings to the database, container-managed persistence entity beans require a CMP deployment descriptor that conforms to either the WebLogic Server 5.1 CMP DTD, the WebLogic Server 8.1 EJB 1.1 DTD, or the WebLogic Server 8.1 EJB 2.0 DTD.
- In WebLogic Server 8.1 the `max-beans-in-cache` parameter controls the maximum number of beans in the cache for Database concurrency. In earlier WebLogic Server versions, `max-beans-in-cache` was ignored; the size of the cache was unlimited. You may need to increase the size of this parameter.

# EJB Porting Recommendations

- Use `TxDataSource`

  EJBs should always get their database connections from a `TxDataSource`. This allows the EJB container's transaction management to interface with the JDBC connection, and it also supports XA transactions.

  The WebLogic Server 8.1 CMP Deployment Descriptor supports `TxDataSources` and should be used instead of the WebLogic Server 5.1 CMP Deployment Descriptor which only specifies a connection pool.

- Use a fast compiler with `ejbc`

  The WebLogic Server EJB compiler (`weblogic.ejbc`) generates Java code that is then compiled by the Java compiler. By default, WebLogic Server uses the `javac` compiler included with the bundled JDK. The EJB compiler runs much

faster when a faster Java compiler is used. Use the `-compiler` option to specify an alternate compiler as in the following example:

```
java weblogic.ejbc -compiler sj pre_AccountEJB.jar
AccountEJB.jar
```

- Correct errors before deploying the EJB on WebLogic Server 8.1

  The WebLogic Server 8.1 EJB compiler (`ejbc`) includes additional verification that was missing from earlier WebLogic Server releases. It is possible that an EJB deployed in a previous WebLogic Server version without error, but WebLogic Server 8.1 finds and complains about the error. These errors must be corrected before the EJB is deployed in WebLogic Server 8.1.

  For instance, WebLogic Server 8.1 ensures that a method exists if a transaction attribute is set for that method name. This helps identify a common set of errors where transaction attributes were mistakenly set on non-existent methods.

- The following table shows the descriptor combinations supported by WebLogic Server 8.1.

**Table 3-1**

| EJB Version | WebLogic Server Version | The CMP Version |
|---|---|---|
| Any existing WebLogic Server 5.1 deployment uses the following combination and can be deployed without changing descriptors or code in WebLogic Server 8.1. | | |
| 1.1 | 5.1 | 5.1 |
| The below combinations include a WebLogic Server 8.1 CMP deployment descriptor. The WebLogic Server 8.1 EJB 1.1 CMP deployment descriptor allows multiple EJBs to be specified within a single EJB JAR file, and it supports using a `TxDataSource` which is required when an EJB is enlisted in a two-phase /XA transaction. | | |
| 1.1 | 5.1 | 8.1 |
| 1.1 | 8.1 | 8.1 |
| EJB 2.0 beans always use the WebLogic Server 6.x or 8.1 deployment descriptors. | | |
| 2.0 | 6.x | 8.1 |
| 2.0 | 8.1 | 8.1 |

For more information on Enterprise JavaBeans, see *Enterprise JavaBean Components* and *Programming WebLogic Enterprise Java Beans*.

# Steps for Porting a 1.0 EJB from WebLogic Server 4.5.x to WebLogic Server 8.1

WebLogic Server 3.1.x, 4.0.x, and 4.5.x supported the EJB 1.0 specification. To port a 1.0 EJB from WebLogic Server 4.5 to WebLogic Server 8.1:

1. Convert the EJB 1.0 deployment descriptor to either the EJB 1.1 or the EJB 2.0 XML deployment descriptor. You can do this automatically using the *DDCreator* tool.
2. Package the deployment descriptor in a JAR file which includes the deployment descriptor's output from step one above and the bean classes.
3. Run the WebLogic Server EJB compiler (ejbc) to compile the JAR file. The `ejbc` tool ensures that when the EJB compiles, it conforms to either the EJB 1.1 or EJB 2.0 specifications.
4. Correct any compliance errors before deploying the EJB in the EJB container.

To ensure EJB 1.1 or 2.0 compliance, make the following changes to the EJB 1.0 beans:

- EJB 1.0 beans referred to the `SessionContext` or `EntityContext` as transient. When EJB 1.1 or 2.0 beans are deployed, the reference cannot be transient. For example:

  ```
  private transient SessionContext ctx;
  ```

  should be:

  ```
  private SessionContext ctx;
  ```

- The `ejbCreate` method for EJB 1.0 CMP entity beans had a void return type. When EJB 1.1 or 2.0 beans are deployed, the return type must be the primary key class which allows you to write a bean-managed persistent entity bean and then sub-class it with a CMP implementation. For example:

  ```
  public void ejbCreate (String name) {
  firstName = name;
  }
  ```

should be:

```
public AccountPK ejbCreate (String name) {
firstName = name;
return null; // required by the EJB specification
}
```

- In EJB 1.1 or 2.0, entity beans cannot use bean-managed transactions. Instead, they must run with a container-managed transaction attribute (for example, Required, Mandatory, etc.).

# Steps for Porting a 1.1 EJB from WebLogic Server 5.1 to WebLogic Server 8.1

The WebLogic Server 5.1 deployment descriptor only allows the exclusive or read-only concurrency options. The database concurrency option is available when upgrading to the WebLogic Server 8.1 `weblogic-ejb-jar.xml` file. For more information about this option, see information on database concurrency in *weblogic-ejb-jar.xml Document Type Definitions* in *Programming WebLogic Enterprise JavaBeans*.

The WebLogic Server 8.1 CMP deployment descriptor allows multiple EJBs to be specified and it supports using a `TxDataSource` instead of a connection pool. Using a `TxDataSource` is required when XA is being used with EJB 1.1 CMP.

To port a 1.1 EJB from WebLogic Server 5.1 to WebLogic Server 8.1:

1. Open the Administration Console. From the home page, click on Install Applications under the Getting Started heading.
2. Locate the JAR file you wish to port by clicking the Browse button, then click Open and then Upload. Your bean should now be automatically deployed on WebLogic Server 8.1.
3. Run a `setEnv` script in a client window and set your development environment. (For more information, see *Establishing a Development Environment* in *Developing WebLogic Server Applications*.)
4. Compile all the needed client classes. For example, using the Stateless Session Bean sample that was provided with WebLogic Server 8.1, you would use the following command:

```
javac -d %CLIENTCLASSES% Trader.java TraderHome.java
TradeResult.java Client.java
```

5. To run the client, enter this command:

   ```
   java -classpath %CLIENTCLASSES%;%CLASSPATH%
   examples.ejb.basic.statelessSession.Client
   ```

   This command ensures that the EJB interfaces are referenced in your client's classpath.

# Steps for Converting an EJB 1.1 to an EJB 2.0

To convert an EJB 1.1 bean to an EJB 2.0 bean, you can use the WebLogic Server DDConverter utility.

BEA Systems recommends that you develop EJB 2.0 beans in conjunction with WebLogic Server 8.1. For 1.1 beans already used in production, it is not necessary to convert them to 2.0 beans. EJB 1.1 beans are deployable with WebLogic Server 8.1. If you do wish to convert 1.1 beans to 2.0 beans, see the *DDConverter* documentation in *Programming WebLogic Enterprise JavaBeans* for information on how to do this conversion.

The basic steps required to convert a simple CMP 1.1 bean to a 2.0 bean are as follows:

1. Make the bean class abstract. EJB 1.1 beans declare CMP fields in the bean. CMP 2.0 beans use abstract `getXXX` and `setXXX` methods for each field. For instance, 1.1 Beans will use `public String name`. 2.0 Beans should use `public abstract String getName()` and `public abstract void setName(String n)`. With this modification, the bean class should now read the container-managed fields with the `getName` method and update them with the `setName` method.

2. Any CMP 1.1 finder that used `java.util.Enumeration` should now use `java.util.Collection`. CMP 2.0 finders cannot return `java.util.Enumeration`. Change your code to reflect this:

   ```
   public Enumeration findAllBeans()
       Throws FinderException, RemoteException;
   ```

   becomes:

   ```
   public Collection findAllBeans()
       Throws FinderException, RemoteException;
   ```

## Porting EJBs from Other J2EE Application Servers

Any EJB that complies with the EJB 1.1 or EJB 2.0 specifications may be deployed in the WebLogic Server 8.1 EJB container. Each EJB JAR file requires an `ejb-jar.xml` file, a `weblogic-ejb-jar.xml` deployment descriptor, and a CMP deployment descriptor if CMP entity beans are used. The WebLogic Server EJB examples located in `samples\examples\ejb11` and `samples\examples\ejb20` of the WebLogic Server distribution include sample weblogic deployment descriptors.

# Creating an Enterprise Application

An Enterprise Application is a JAR file with an EAR extension. An EAR file contains all of the JAR and WAR component archive files for an application and an XML descriptor that describes the bundled components. The `META-INF\application.xml` deployment descriptor contains an entry for each Web and EJB module, and additional entries to describe security roles and application resources such as databases.

```
EnterpriseApplicationStagingDirectory\
                                      |
                                      + .jar files
                                      |
                                      + .war files
                                      |
                                      +META-INF\-+
                                                 |
                                                 + application.xml
```

To create an EAR file:

1. Assemble all of the WAR and JAR files for your application.
2. Copy the WAR and EJB JAR files into the staging directory and then create a `META-INF\application.xml` deployment descriptor for the application. Follow the directory structure depicted above.

   The `application.xml` file contains a descriptor for each component in the application, using a DTD supplied by Sun Microsystems. For more information on the application.xml file, see *Application Deployment Descriptor Elements* in

*Developing WebLogic Server Applications*. Note that if you are using JSPs and want them to compile at run time you must have the home and remote interfaces of the bean included in the classes directory of your WAR file.

3. Create the Enterprise Archive by executing a jar command like the following in the staging directory:

   ```
   jar cvf myApp.ear *
   ```

4. Click on the Install Applications link under the Getting Started heading in the home page of the console and place the EAR file in the *domain*`\applications` directory. For more information on Enterprise Applications, see *Packaging Enterprise Applications* in *Developing WebLogic Server Applications*.

# Understanding J2EE Client Applications

WebLogic Server supports J2EE client applications, packaged in a JAR file with a standard XML deployment descriptor. Client applications in this context are clients that are not Web browsers. They are Java classes that connect to WebLogic Server using Remote Method Invocation (RMI). A Java client can access Enterprise JavaBeans, JDBC connections, messaging, and other services using RMI. Client applications range from simple command line utilities that use standard I/O to highly interactive GUI applications built using the Java Swing/AWT classes.

To execute a WebLogic Server Java client, the client computer needs the `weblogic_sp.jar` file, the `weblogic.jar` file, the remote interfaces for any RMI classes and Enterprise Beans that are on WebLogic Server, as well as the client application classes. To simplify maintenance and deployment, it is a good idea to package a client-side application in a JAR file that can be added to the client's classpath along with the `weblogic.jar` and `weblogic_sp.jar` files. The `weblogic.ClientDeployer` command line utility is executed on the client computer to run a client application packaged to this specification. For more information about J2EE client applications, see *Packaging Client Applications* in *Developing WebLogic Server Applications.*

# Upgrading JMS

WebLogic Server 8.1 supports the *JavaSoft JMS specification version 1.0.2*.

- Weblogic Server 4.5.1 — Porting is supported *only* for SP15. Customers running all service packs should contact BEA Support.
- Weblogic Server 5.1 — Customers running SP07 or SP08 should contact BEA Support before porting existing JDBC stores to version 8.1.
  - In order to port object messages, the object classes need to be in the Weblogic Server 8.1 server CLASSPATH.
  - For destinations that are not configured in Weblogic Server 8.1, the ported messages will be dropped and the event will be logged.

For more information on porting your WebLogic JMS applications, see *Porting WebLogic JMS Applications* in *Programming WebLogic JMS*. Note that WebLogic Events are deprecated and are replaced by JMS messages with NO_ACKNOWLEDGE or MULTICAST_NO_ACKNOWLEDGE delivery modes. Each of these delivery modes is described in *WebLogic JMS Fundamentals* in *Programming WebLogic JMS*.

# Upgrading Oracle

BEA Systems, mirroring Oracle's support policy, supports the Oracle releases listed in the *Platform Support for WebLogic jDriver JDBC Drivers* on the *WebLogic Server Certifications* page. BEA no longer supports the following Oracle client versions: 7.3.4, 8.0.4, 8.0.5, and 8.1.5.

To use the Oracle Client Version 7.3.4, use the backward compatible oci816_7 shared library. As stated above, BEA no longer supports this configuration.

To upgrade to Oracle Client Version 9i, or read detailed documentation on the WebLogic jDriver and Oracle databases, see *Configuring WebLogic jDriver for Oracle* in *Installing and Using WebLogic jDriver for Oracle*.

For supported platforms, as well as DBMS and client libraries, see the BEA *Certifications Page*. The most current certification information will always be posted on the Certifications page.

# Additional Porting and Deployment Considerations

The following sections provide additional information that may be useful when you deploy applications on WebLogic Server 8.1. Deprecated features, upgrades, and the important changes that have been made in WebLogic Server 8.1 are noted.

**Note:** WebLogic Server 8.1 uses PointBase 4.2 as a sample database and does not bundle the Cloudscape database.

# Applications and Managed Servers

By default, applications are deployed to the Administration Server. However, in most cases, this is not good practice. You should use the Administration Server only for administrative purposes. Use the Administration Console to define new managed servers and associate the applications with those servers. For more information, see *Using WebLogic Server Clusters* and *Overview of WebLogic System Administration* in the *Administration Guide*.

# Deployment

By default, WebLogic Server version 8.1 uses the two-phase deployment model. For more information on this deployment model and other 8.1 deployment features, see *WebLogic Server Deployment* in *Developing WebLogic Server Applications*. Therefore, if you deploy a 4.5 or 5.1 application in your 8.1 server, the deployment model is unspecified and, thus, uses a two-phase deployment. For more information, see the *Release Notes*.

# Plug-ins

The communication between the plug-in and WebLogic Server 4.5 and 5.1 is clear text. The plug-ins in WebLogic Server 8.1 support SSL communication between the plug-in and the back-end WebLogic Server.

To upgrade the plug-in, copy the new plug-in over the old one and restart the IIS, Apache, or iPlanet web server.

# Internationalization (I18N)

Several internationalization and localization changes have been made in this version:

- Changes to the log file format affect the way that messages are localized. The new message format also has additions to the first line: *begin marker*, *machine name*, *server name*, *thread id*, *user id*, *tran id*, and *message id*.
- A new internationalized logging API enables users to log messages in the server and clients.
- Clients log to their own logfiles, which are in the same format as the server logfiles, with the exception of the *servername* and *threadid* fields.
- `LogServicesDef` is deprecated. Instead, use the internationalized API or `weblogic.logging.NonCatalogLogger` (when internationalization is not required).

For details on internationalization in this version, see the *Internationalization Guide*.

# Java Transaction API (JTA)

JTA has changed as follows:

- WebLogic Server 8.1 supports the JTA 1.0.1 specification. Updated JTA documentation is provided in *Programming WebLogic JTA*.
- Based on the inclusion of support for JTA, the JTS JDBC driver (with properties in `weblogic.jts.*` and URL `jdbc:weblogic:jts`:..) has been replaced by a JTA JDBC/XA driver. Existing properties are available for backward compatibility, but you should change the class name and properties to reflect the JTS to JTA name change.

# Java Database Connectivity (JDBC)

The following changes have been made to JDBC:

- The WebLogic T3 API was deprecated in WebLogic Server 6.1; use the RMI JDBC driver in its place. This also applies to users porting from WebLogic Server 4.5.x.
- The `weblogic.jdbc20.*` packages are being replaced with `weblogic.jdbc.*` packages. All WebLogic JDBC drivers are now compliant with JDBC 2.0.
- If you have a current connection and are using a `preparedStatement`, and the stored procedure gets dropped in the DBMS, use a new name to create the stored procedure. If you recreate the stored procedure with the same name, the `preparedStatement` will not know how to access the newly created stored procedure—it is essentially a different object with the same name.

# JVM

WebLogic Server 8.1 installs the Java Virtual Machine (JVM), JDK 1.4.1, with the server installation. The `setenv.sh` scripts provided with the server all point to the JVM. The latest information regarding certified JVMs is available at the *Certifications Page*.

# RMI

The following tips are for users porting to WebLogic Server 8.1 who used RMI in their previous version of WebLogic Server:

- Re-run the WebLogic RMI compiler, `weblogic.rmic`, on any existing code to regenerate the wrapper classes so they are compatible with WebLogic Server 8.1.
- Use `java.rmi.Remote` to tag interfaces as remote. Do not use `weblogic.rmi.Remote`.
- Use `java.rmi.*Exception` (e.g., `import java.rmiRemoteException;`). Do not use `weblogic.rmi.*Exception`.
- Use JNDI instead of `*.rmi.Naming`.
- Use `weblogic.rmic` to generate dynamic proxies and bytecode; with the exception of RMI IIOP, stubs and skeletons classes are no longer generated.

**Note:** For more information, see *WebLogic RMI Fea tures and Guidelines* in *Programming WebLogic RMI*.

- Use `weblogic.rmi.server.UnicastRemoteObject.exportObject()` to get a stub instance.
- The RMI examples have not currently been updated to use java.rmi.* and JNDI. The examples will be revised to reflect `java.rmi.*` and JNDI in a future release.

# Security

For specific questions and answers, see the security section of the *WebLogic Server Frequently Asked Questions*. Upgrading WebLogic Server 4.5 or 5.1 to WebLogic Server 8.1 with the WebLogic Server 8.1 security functionality is a two-step process involving first upgrading to the WebLogic Server 6.x security functionality and then upgrading from WebLogic Server 6.x to WebLogic Server 8.1.

The following list of tips and issues regard upgrading from WebLogic Server 4.5 or 5.1 to the WebLogic Server 6.x functionality:

- The name of the default security realm changed from `WLPropertyRealm` to the File realm. Realm attributes are now stored in the `fileRealm.properties` file instead of the `weblogic.properties` file.
- Redefine your realm and authorization attributes through the Administration Console. The resulting information is stored in the `fileRealm.properties` file.
- It is highly recommended that at the end of installation, you check all security settings to make sure they are the appropriate ones for their environment.
- ACLs can no longer be used to specify security for stand-alone servlets because stand-alone servlets have been completely replaced by web applications. Web applications can only be secured using the web app's deployment descriptors as defined in the Servlet 2.3 specification.

## Converting the weblogic.properties File

A large portion of implementing security in the WebLogic Server environment is configuration. To port a current security configuration, convert the Mbean attributes in the `weblogic.properties` file to XML attributes in the `config.xml` file using the Administration Console. Details on converting `weblogic.properties` files are described in the *Console Help* documentation.

## Porting Security Realms

In WebLogic 6.x, WebLogic Server provides a new management architecture for security realms. The management architecture implemented through MBeans allows you to manage security realms through the Administration Console. If you have a security realm from a previous release of WebLogic Server, use the following information to port to the new architecture:

- If you are using the Windows NT, UNIX, or LDAP security realms, use the Convert weblogic.properties option in the Administration Console to convert the security realm to the new architecture. Note that you can view users, groups, and ACLs in a Windows NT, UNIX, or LDAP security realm in the Administration Console. However, you still need to use the tools in the Windows NT, UNIX, or LDAP environments to manage users and groups.
- If you are using a custom security realm, follow the steps in *Installing a Custom Security Realm* in the section called *Specifying a Security Realm* in the *Administration Guide* to specify information about how the users, groups, and optional ACLs are stored in your custom security realm.
- The Delegating security realm is no longer supported. If you are using the Delegating security realm, you will have to use another type of security realm to store users, groups, and ACLs.
- If you are using the RDBMS security realm, use one of the following options to convert the security realm:
  - If you did not change the source for the RDBMS security realm, follow the steps in *Configuring the RDBMS Security Realm* in the section called *Specifying a Security Realm* in the *Administration Guide* to instantiate a new class for your existing RDBMS security realm and define information about the JDBC driver being used to connect to the database and the schema used by the security realm. In this case, you are creating a MBean in WebLogic Server for the RDBMS security realm.

- If you customized the RDBMS security realm, convert your source to use the MBeans. Use the code example in the `\samples\server\src\examples\security\rdbmsrealm` directory as a guide to converting your RDBMS security realm. Once you have converted your RDBMS security realm to MBeans, follow the instructions in *Configuring the RDBMS Security Realm* in the section called *Specifying a Security Realm* in the *Administration Guide* to define information about the JDBC driver being used to connect to the database and the schema used by the security realm.

- Once you have configured WebLogic Server to use the 6.1 security functionality, see Upgrading WebLogic Server 6.x to Version 8.1 to read about how to upgrade to the WebLogic Server 8.1 security functionality.

# Session Porting

WebLogic Server 6.0 and later does not recognize cookies from previous versions because cookie format changed with 6.0. WebLogic Server ignores cookies with the old format, and creates new sessions.

The default name for cookies has changed from 5.1, when it was `WebLogicSession`. Beginning in WebLogic 6.0, cookies are named `JSESSIONID` by default.

See *weblogic.xml Deployment Descriptor Elements* in *Assembling and Configuring Web Applications* for more information.

# Standalone HTML and JSPs

In the original domain provided with WebLogic Server 8.1, as well as in any domains that have been created using the `weblogic.properties` file converter, `domain\applications\DefaultWebApp_myserver` directory is created. This directory contains files made available by your Web server. You can place HTML and JSP files here and make them available, separate from any applications you install. If necessary, you can create subdirectories within the `DefaultWebApp_myserver` directory to handle relative links, such as image files.

# Web Components

The following tips are for users porting to WebLogic Server 8.1 who used Web components in their previous version of WebLogic Server:

- All Web components in WebLogic Server now use Web Applications as the mechanism for defining how WebLogic Server serves up JSPs, servlets, and static HTML pages. In a new installation of WebLogic Server, the server will configure a default Web Application. Customers upgrading to WebLogic Server 8.1 should not need to perform any registrations because this default Web Application closely approximates the document root, the JSPServlet, and servlet registrations performed using the `weblogic.properties` file contained in earlier versions.
- Convert your existing `weblogic.properties` file to XML files using the Administration Console. See the *Console Help* for more details.
- SSI is no longer supported.
- URL ACLs are deprecated. Use Servlet 2.3 features instead.
- Some information has moved from `web.xml` to `weblogic.xml`. This reorganization allows a third-party Web application based strictly on Servlet 2.3 to be deployed without modifications to its J2EE standard deployment descriptor (`web.xml`). WebLogic Server 5.1 style settings made in the `web.xml` file using `<context-param>` elements are supported for backward compatibility, but you should adopt the new way of deploying. The following sets of parameters previously defined in `web.xml` are now defined in `weblogic.xml`:

  **JSP Parameters** (`keepgenerated, precompile compileCommand, verbose, packagePrefix, pageCheckSeconds, encoding`)

  **HTTP sessionParameters** (`CookieDomain, CookieComment, CookieMaxAgeSecs, CookieName, CookiePath, CookiesEnabled, InvalidationIntervalSecs, PersistentStoreDir, PersistentStorePool, PersistentStoreType, SwapIntervalSecs, IDLength, CacheSize, TimeoutSecs, JDBConnectionTimeoutSecs, URLRewritingEnabled`)
- For more information, see *Writing Web Application Deployment Descriptors* in *Assembling and Configuring Web Applications*.

# Wireless Application Protocol Applications

To run a Wireless Application Protocol (WAP) application on WebLogic Server 8.1, you must now specify the MIME types associated with WAP in the `web.xml` file of the web application. In WebLogic Server 5.1, MIME types were defined in the `weblogic.properties` file. For information on required MIME types see *Programming WebLogic Server for Wireless Services*. For information on creating and editing a `web.xml` file, see *Writing Web Application Deployment Descriptors* in *Assembling and Configuring Web Applications*.

# Writable config.xml File

WebLogic Server 8.1 automatically updates configuration information read from the 6.x `config.xml` file to include version 8.1 information. In order for these changes to be retained between invocations of the server, the `config.xml` file must be writable. To allow the file to be writable, make a back-up copy of your `config.xml` file from your 6.x configuration and change the file attributes.

# XML 8.1 Parser and Transformer

The built-in parser and transformer in WebLogic Server 8.1 have been updated to Xerces 1.4.4 and Xalan 2.2, respectively. If you used the APIs that correspond to older parsers and transformers that were shipped in previous versions of WebLogic Server, and if you used classes, interfaces, or methods that have been deprecated, you might receive deprecation messages in your applications .

WebLogic Server 8.1 also includes the WebLogic FastParser, a high-performance XML parser specifically designed for processing small to medium size documents, such as SOAP and WSDL files associated with WebLogic Web services. Configure WebLogic Server to use FastParser if your application handles mostly small to medium size (up to 10,000 elements) XML documents.

The WebLogic Server 8.1 distribution no longer includes the unmodified Xerces parser and Xalan transformer in the *`WL_HOME`*`\server\ext\xmlx.zip` file.

To see the explicit differences between the old and new parsers, use a tool contained in the apichange.zip file located on dev2dev Online.

# Deprecated APIs and Features

The following APIs and features are deprecated in anticipation of future removal from the product:

- WebLogic Events

  WebLogic Events are deprecated and should be replaced by JMS messages with NO_ACKNOWLEDGE or MULTICAST_NO_ACKNOWLEDGE delivery modes. See *Non-transacted session* in *Programming WebLogic JMS* for more information.

- WebLogic HTMLKona
- WebLogic JDBC t3 Driver
- WebLogic Enterprise Connectivity
- WebLogic Time Services is deprecated and should be replaced by JMX Timer Service. For documentation of JMX Timer Service, see *Interface TimerMBean* and *Class Timer*.
- WebLogic Workspaces
- Zero Administration Client (ZAC) is deprecated and should be replaced by JavaWebStart.
- `-Dweblogic.management.host`
- `weblogic.deploy` is deprecated in this release of WebLogic Server 8.1 and is replaced by `weblogic.Deployer`. For more information, see *Deployment Tools and Procedures* in *Developing WebLogic Server Applications*.
- `weblogic.management.tools.WebAppComponentRefreshTool` and `weblogic.refresh` are both deprecated in this release of WebLogic Server 8.1. They have been replaced by `weblogic.Deployer`.

# Removed APIs and Features

The following APIs and features have been removed:

- The old administrative console GUI

- The Deployer Tool
- WebLogic Beans
- WebLogic jHTML
- WebLogic Remote
- WorkSpaces
- WebLogic Server Tour
- T3Client
- Jview support
- SSI
- Weblogic Bean Bar
- RemoteT3
- Jview support
- Weblogic COM

  This feature relied on the Microsoft JVM (Jview), which is no longer supported.

CHAPTER

# A The weblogic.properties Mapping Table

The weblogic.properties mapping table shows which `config.xml, web.xml, or weblogic.xml` attribute handles the function formerly performed by `weblogic.properties` properties.

| **weblogic.properties file Property** | **.xml Configuration Attribute** | **Console Label** |
|---|---|---|
| `weblogic.administrator.email` | `config.xml: EmailAddress (Administrator element)` | |
| `weblogic.administrator.location` | `config.xml: Notes (freeform, optional) (Administrator element)` | |
| `weblogic.administrator.name` | `config.xml: Name (Administrator element)` | |
| `weblogic.administrator.phone` | `config.xml: PhoneNumber (Administrator element)` | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| `weblogic.cluster.defaultLoadAlgorithm` | `config.xml: DefaultLoadAlgorithm` (`Cluster` element) | Clusters: *`clustername`*: Configuration: General: Default Load Algorithm |
| `weblogic.cluster.multicastAddress` | `config.xml: MulticastAddress` (`Cluster` element) | Clusters: *`clustername`*: Configuration: Multicast: Multicast Address |
| `weblogic.cluster.multicastTTL` | `config.xml: MulticastTTL` (`Cluster` element) | Clusters: *`clustername`*: Configuration: Multicast: Multicast TTL |
| `weblogic.cluster.name` | `config.xml Cluster Address` (`Cluster` element) | Clusters: *`clustername`*: Configuration: General: Cluster Address |
| `weblogic.httpd.authRealmName` | `config.xml: AuthRealmName` (`WebAppComponent` element) | Deployments: Web Applications: *`applicationname`*: Configuration: Other: Auth Realm Name |
| `weblogic.httpd.charsets` | `config.xml: Charsets` (`WebServer` element) | |
| `weblogic.httpd.clustering.enable` | `config.xml: ClusteringEnabled` (`WebServer` element) | |
| `weblogic.httpd.defaultServerName` | `config.xml DefaultServerName` (`WebServer` element) | Servers: *`servername`*: Configuration: HTTP: Default Server Name |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.httpd.defaultServlet | web.xml:<br>define a servlet-mapping with the URL pattern of /<br><servlet-mapping> element | |
| weblogic.httpd.defaultWebApp | config.xml:<br>DefaultWebApp<br>(WebServer element) | |
| weblogic.httpd.enable | config.xml:<br>HttpdEnabled<br>(Server element) | |
| weblogic.httpd.enableLogFile | config.xml:<br>LoggingEnabled<br>(WebServer element) | |
| weblogic.httpd.http.keepAliveSecs | config.xml:<br>KeepAliveSecs<br>(WebServer element) | |
| weblogic.httpd.https.keepAliveSecs | config.xml:<br>HttpsKeepAliveSecs<br>(WebServer element) | |
| weblogic.httpd.indexDirectories | config.xml:<br>IndexDirectoryEnabled<br>(WebAppComponent element) | Deployments: Web Applications: *applicationname*: Configuration: Files: Index Directories |
| weblogic.httpd.keepAlive.enable | config.xml:<br>KeepAliveEnabled<br>(WebServer element) | Servers: *servername*: Configuration: HTTP: Enable Keep Alives |
| weblogic.httpd.logFileBufferKBytes | config.xml:<br>LogFileBufferKBytes<br>(WebServer element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.httpd.logFileFlushSecs | config.xml: LogFileFlushSecs (WebServer element) | |
| weblogic.httpd.logFileFormat | config.xml: LogFileFormat (WebServer element) | Services: Virtual Host: Log File Format |
| weblogic.httpd.logFileName | config.xml: LogFileName (WebServer element) | Services: Virtual Host: Log File Name |
| weblogic.httpd.logRotationPeriodMins | config.xml: LogRotationTimeBegin (WebServer element) | |
| weblogic.httpd.logRotationPeriodMins | config.xml: LogRotationPeriodMins (WebServer element) | |
| weblogic.httpd.logRotationType | config.xml: LogRotationType (WebServer element) | Servers: *servername:* Logging: HTTP: Rotation Type |
| weblogic.httpd.maxLogFileSizeKBytes | config.xml: MaxLogFileSizeKBytes (WebServer element) | Servers: *servername*: Logging: HTTP: Max Log File Size Kbytes |
| weblogic.httpd.mimeType | web.xml: mime-type (<mime-mapping> element) | |
| weblogic.httpd.postTimeoutSecs | config.xml: PostTimeoutSecs (WebServer element) | Servers: *servername*: Configuration: HTTP: Post Timeout Secs |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| `weblogic.httpd.servlet.extension CaseSensitive` | `config.xml: ServletExtensionCaseSensitive` (`WebAppComponent` element) | Deployments: Web Applications: *`applicationname`*: Configuration: Files: Case Sensitive Extensions |
| `weblogic.httpd.servlet.reloadCheckSecs` | `config.xml: ServletReloadCheckSecs` (`WebAppComponent` element) | Deployments: Web Applications: *`applicationname`*: Configuration: Files: Reload Period |
| `weblogic.httpd.servlet.SingleThreadedModelPoolSize` | `config.xml: SingleThreadedServletPoolSize` (`WebAppComponent` element) | Deployments: Web Applications: *`applicationname`*: Configuration: Files: Single Threaded Servlet Pool Size |
| `weblogic.httpd.session.cacheEntries` | `weblogic.xml: CacheSize` `<param-name>/<param-value>` element pair | Servers: *`servername`*: Configuration: SSL: Certificate Cache Size |
| `weblogic.httpd.session.cookie.comment` | `weblogic.xml: CookieComment` `<param-name>/<param-value>` element pair | |
| `weblogic.httpd.session.cookie.domain` | `weblogic.xml: CookieDomain` `<param-name>/<param-value>` element pair | |
| `weblogic.httpd.session.cookie.maxAgeSecs` | `weblogic.xml: CookieMaxAgeSecs` `<param-name>/<param-value>` element pair | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.httpd.session.cookie.name | weblogic.xml: CookieName <param-name>/<param-value> element pair | |
| weblogic.httpd.session.cookie.path | weblogic.xml: CookiePath <param-name>/<param-value> element pair | |
| weblogic.httpd.session.cookies.enable | weblogic.xml: CookiesEnabled <param-name>/<param-value> element pair | |
| weblogic.httpd.session.debug | weblogic.xml: SessionDebuggable <param-name>/<param-value> element pair | |
| weblogic.httpd.session.enable | weblogic.xml: SessionTrackingEnabled <param-name>/<param-value> element pair | |
| weblogic.httpd.session.invalidationintervalSecs | weblogic.xml: InvalidationIntervalSecs <param-name>/<param-value> element pair | |
| weblogic.httpd.session.jdbc.connTimeoutSecs | weblogic.xml: JDBCConnectionTimeoutSecs <param-name>/<param-value> element pair | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.httpd.session.persistentStoreDir | weblogic.xml: PersistentStoreDir <param-name>/<param-value> element pair | |
| weblogic.httpd.session.persistentStorePool | weblogic.xml: PersistentStorePool <param-name>/<param-value> element pair | |
| weblogic.httpd.session.persistentStoreShared | weblogic.xml: SessionPersistentStoreShared <param-name>/<param-value> element pair | |
| weblogic.httpd.session.persistentStoreType | weblogic.xml: PersistentStoreType <param-name>/<param-value> element pair | |
| weblogic.httpd.session.sessionIDLength | weblogic.xml: IDLength <param-name>/<param-value> element pair | |
| weblogic.httpd.session.swapintervalSecs | weblogic.xml: SwapIntervalSecs <param-name>/<param-value> element pair | |
| weblogic.httpd.session.timeoutSecs | weblogic.xml: TimeoutSecs <param-name>/<param-value> element pair | Servers: *servername*: Configuration: HTTP: Post Timeout Secs |
| weblogic.httpd.session.URLRewriting.enable | weblogic.xml: URLRewritingEnabled <param-name>/<param-value> element pair | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| `weblogic.httpd.tunneling.clientPingSecs` | `config.xml: TunnelingClientPingSecs` (`Server` element) | Servers: *`servername:`* Configuration: Tuning: Tunneling Client Ping |
| `weblogic.httpd.tunneling.clientTimeoutSecs` | `config.xml: TunnelingClientTimeoutSecs` (`Server` element) | Servers: *`servername:`* Configuration: Tuning: Tunneling Client Timeout |
| weblogic.httpd.tunnelingenabled | config.xml TunnelingEnabled (`Server` element) | Servers: *`servername:`* Configuration: Tuning: Enable Tunneling |
| `weblogic.httpd.URLResource` | `config.xml: URLResource` (`WebServer` element) | |
| `weblogic.iiop.password` | `config.xml: DefaultIIOPPassword` (`Server` element) | Servers: *`servername:`* Configuration: Protocols: Default IIOP Password |
| `weblogic.iiop.user` | `config.xml: DefaultIIOPUser` (`Server` element) | Servers: *`servername:`* Configuration: Protocols: Default IIOP User |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| `weblogic.jdbc.connectionPool`<br>■ `url`=*URL for JDBC Driver*<br>■ `driver`=*full package name for JDBC driver*<br>■ `loginDelaySecs`=*seconds between connections*<br>■ `initialCapacity`=*initial number of JDBC connections*<br>■ `maxCapacity`=*maximum number of JDBC connections*<br>■ `capacityIncrement`=*increment interval*<br>■ `allowShrinking`=*true to allow shrinking*<br>■ `shrinkPeriodMins`=*interval before shrinking*<br>■ `testTable`=*name of table for autorefresh test*<br>■ `refreshTestMinutes`=*interval for autorefresh test*<br>■ `testConnsOnReserve`=*true to test connection at reserve*<br>■ `testConnsOnRelease`=*true to test connection at release*<br>■ `props`=*props for JDBC connection* | `URL`<br>`DriveName`<br>`LoginDelaySeconds`<br>`InitialCapacity`<br>`MaxCapacity`<br>`CapacityIncrement`<br>`AllowShrinking`<br>`ShrinkPeriodMinutes`<br>`TestTableName`<br>`RefreshMinutes`<br>`TestConnectionsOnReserve`<br>`TestConnectionsOnRelease`<br>`Properties`<br>`JDBCConnectionPool Element`<br>`ConnLeakProfilingEnabled`<br>`ACLName`<br>`CapacityEnabled`<br>`SupportsLocalTransaction`<br>`KeepLogicalConnOpenOnRelease`<br>`Password` | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.jdbc.enableLogFile | config.xml: JDBCLoggingEnabled (Server element) | |
| weblogic.jdbc.logFileName | config.xml: JDBCLogFileName (Server element) | |
| weblogic.jms.ConnectionConsumer | config.xml JMSConnectionConsumer element MessagesMaximum Selector Destination | |
| weblogic.jms.connectionFactoryArgs.<<factoryName>><br>■ ClientID<br>■ DeliveryMode<br>■ TransactionTimeout | config.xml: JMSConnectionFactory element ClientID DefaultDeliveryMode TransactionTimeout UserTransactionsEnabled AllowCloseInOnMessage | |
| weblogic.jms.connectionFactoryName | config.xml: JMSConnectionFactory element JNDIName | |
| weblogic.jms.connectionPool | ConnectionPool (JMSJDBCStore element) | |
| weblogic.jms.queue | config.xml: JNDIName StoreEnabled (JMSDestination element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.jms.queueSessionPool | config.xml: ConnectionConsumer ConnectionFactory ListenerClass AcknowledgeMode SessionsMaximum Transacted (JMSSessionPool element) | |
| weblogic.jms.tableNamePrefix | config.xml: PrefixName | |
| weblogic.jms.topic | config.xml JNDIName StoreEnabled (JMSDestination element) | Services: JMS: Connection Factories: JNDI Name |
| weblogic.jms.topicSessionPool | config.xml: ConnectionConsumer ConnectionFactory ListenerClass AcknowledgeMode SessionsMaximum Transacted (JMSSessionPool element) | |
| weblogic.jndi.transportableObjectFactories | config.xml: JNDITransportableObjectFactoryList (Server element) | Servers: *servername:* |
| weblogic.login.readTimeoutMillisSSL | config.xml LoginTimeoutMillis (SSL element) | Servers: *servername:* |
| weblogic.security.audit.provider | config.xml AuditProviderClassName (Security element) | Security: General: Audit Provider Class |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.security.certificate.authority | config.xml ServerCertificateChainFileName (SSL element) | Servers: *servername*: Configuration: SSL: Server Certificate Chain File Name |
| weblogic.security.certificate.server | config.xml: ServerCertificateFileName (SSL element) | Servers: *servername*: Configuration: SSL: Server Certificate File Name |
| weblogic.security.certificateCacheSize | config.xml: CertificateCacheSize (SSL element) | Servers: *servername*: Configuration: SSL: Certificate Cache Size |
| weblogic.security.clientRootCA | config.xml: TrustedCAFileName (SSL element) | Servers: *servername*: Configuration: SSL: Trusted CA File Name |
| weblogic.security.disableGuest | config.xml: GuestDisabled (Security element) | Security: General: Guest Disabled |
| weblogic.security.enforceClientCert | config.xml: ClientCertificateEnforced (SSL element) | Servers: *servername*: Configuration: SSL: Client Certificate Enforced |
| weblogic.security.key.export.lifespan | config.xml: ExportKeyLifespan (SSL element) | Servers: *servername*: Configuration: SSL: Export Key Lifespan |
| weblogic.security.key.server | config.xml: ServerKeyFileName (SSL element) | Servers: *servername*: Configuration: SSL: Server Key File Name |
| weblogic.security.ldaprealm.authentication | config.xml: AuthProtocol (LDAPRealm element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.security.ldaprealm.credential | config.xml: Credential (LDAPRealm element) | |
| weblogic.security.ldaprealm.factory | config.xml LdapProvider (LDAPRealm element) | |
| weblogic.security.ldaprealm.groupDN | config.xml: GroupDN (LDAPRealm element) | |
| weblogic.security.ldaprealm.groupIsContext | config.xml: GroupIsContext (LDAPRealm element) | |
| weblogic.security.ldaprealm.groupNameAttribute | config.xml: GroupNameAttribute (LDAPRealm element) | |
| weblogic.security.ldaprealm.groupUsernameAttribute | config.xml: GroupUsernameAttribute (LDAPRealm element) | |
| weblogic.security.ldaprealm.principal | config.xml: Principal (LDAPRealm element) | |
| weblogic.security.ldaprealm.ssl | config.xml: SSLEnable (LDAPRealm element) | |
| weblogic.security.ldaprealm.url | config.xml: LDAPURL (LDAPRealm element) | |
| weblogic.security.ldaprealm.userAuthentication | config.xml: UserAuthentication (LDAPRealm element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.security.ldaprealm.userDN | config.xml: UserDN (LDAPRealm element) | |
| weblogic.security.ldaprealm.userNameAttribute | config.xml: UserNameAttribute (LDAPRealm element) | |
| weblogic.security.ldaprealm.userPasswordAttribute | config.xml: UserPasswordAttribute (LDAPRealm element) | |
| weblogic.security.net.connectionFilter | config.xml: ConnectionFilter (Security element) | |
| weblogic.security.ntrealm.domain | config.xml: PrimaryDomain (NTRealm element) | |
| weblogic.security.realm.cache.acl.enable | config.xml: ACLCacheEnable (CachingRealm element) | |
| weblogic.security.realm.cache.acl.size | config.xml: ACLCacheSize (CachingRealm element) | |
| weblogic.security.realm.cache.acl.ttl.negative | config.xml: ACLCacheTTLNegative (CachingRealm element) | |
| weblogic.security.realm.cache.acl.ttl.positive | config.xml: ACLCacheTTLPositive (CachingRealm element) | |
| weblogic.security.realm.cache.auth.enable | config.xml: AuthenticationCacheEnable (CachingRealm element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.security.realm.cache.auth.size | config.xml: AuthenticationCacheSize (CachingRealm element) | |
| weblogic.security.realm.cache.auth.ttl.negative | config.xml: AuthenticationCacheTTLNegative (CachingRealm element) | |
| weblogic.security.realm.cache.auth.ttl.positive | config.xml: AuthenticationCacheTTLPositive (CachingRealm element) | |
| weblogic.security.realm.cache.caseSensitive | config.xml: CacheCaseSensitive (CachingRealm element) | |
| weblogic.security.realm.cache.group.enable | config.xml: GroupCacheEnable (CachingRealm element) | |
| weblogic.security.realm.cache.group.size | config.xml: GroupCacheSize (CachingRealm element) | |
| weblogic.security.realm.cache.group.ttl.negative | config.xml: GroupCacheTTLNegative (CachingRealm element) | |
| weblogic.security.realm.cache.group.ttl.positive | config.xml: GroupCacheTTLPositive (CachingRealm element) | |
| weblogic.security.realm.cache.perm.enable | config.xml: PermissionCacheEnable (CachingRealm element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.security.realm.cache.perm.size | config.xml: PermissionCacheSize (CachingRealm element) | |
| weblogic.security.realm.cache.perm.ttl.negative | config.xml: PermissionCacheTTLNegative (CachingRealm element) | |
| weblogic.security.realm.cache.perm.ttl.positive | config.xml: PermissionCacheTTLPositive (CachingRealm element) | |
| weblogic.security.realm.cache.user.enable | config.xml: UserCacheEnable (CachingRealm element) | |
| weblogic.security.realm.cache.user.size | config.xml: UserCacheSize (CachingRealm element) | |
| weblogic.security.realm.cache.user.ttl.negative | config.xml: UserCacheTTLNegative (CachingRealm element) | |
| weblogic.security.realm.cache.user.ttl.positive | config.xml: UserCacheTTLPositive (CachingRealm element) | |
| weblogic.security.realm.certAuthenticator | config.xml: CertAuthenticator (SSL element) | Servers: *servername*: Configuration: SSL: Cert Authenticator |
| weblogic.security.SSL.ciphersuite | config.xml Ciphersuites (SSL element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.security.ssl.enable | config.xml: Enabled (SSL element) | Servers: *servername*: Configuration: SSL: Enabled |
| weblogic.security.SSL.hostnameVerifier | config.xml HostnameVerifier (SSL element) | Servers: *servername*: Configuration: SSL: Hostname Verifier |
| weblogic.security.SSL.ignoreHostnameVerification | config.xml HostNameVerificationIgnored (SSL element) | |
| weblogic.security.SSLHandler.enable | config.xml: HandlerEnabled (SSL element) | Servers: *servername*: Configuration: SSL: Handler Enabled |
| weblogic.security.unixrealm.authProgram | config.xml: AuthProgram (UnixRealm element) | |
| weblogic.system.AdministrationPort | config.xml AdministrationPort (Server element) | Servers: *servername:* Configuration: General: Administration Port |
| weblogic.system.bindAddr | config.xml: ListenAddress (Server element) | |
| weblogic.system.defaultProtocol | config.xml: DefaultProtocol (Server element) | Servers: *servername:* Configuration: Protocols: Default Protocol |
| weblogic.system.defaultSecureProtocol | config.xml: DefaultSecureProtocol (Server element) | Servers: *servername:* Configuration: Protocols: Default Secure Protocol |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.system.enableConsole | config.xml: StdoutEnabled (Kernel element) | Servers: *servername:* Logging: General: Log to Stdout |
| weblogic.system.enableIIOP | config.xml: IIOPEnabled (Server element) | |
| weblogic.system.enableReverseDNS Lookups | config.xml: ReverseDNSAllowed (Server element) | |
| weblogic.system.enableSetGID, | config.xml: PostBindGID | |
| weblogic.system.enableSetUID, | config.xml: PostBindUIDEnabled | |
| weblogic.system.enableTGIOP | config.xml TGIOPEnabled (Server element) | Servers: *servername:* |
| weblogic.system.helpPageURL | config.xml HelpPageURL (Server element) | Servers: *servername:* |
| weblogic.system.home | config.xml: RootDirectory (Server element) | |
| weblogic.system.ListenPort | config.xml ListenPort (Server element) | Servers: *servername*: Configuration: SSL: Listen Port |
| weblogic.system.logFile | config.xml: FileName (Log element) | |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| weblogic.system.MagicThreadBackToSocket | config.xml: MagicThreadDumpBackToSocket (ServerDebug element) | |
| weblogic.system.MagicThreadDumpFile | config.xml: MagicThreadDumpFile (ServerDebug element) | |
| weblogic.system.MagicThreadDumpHost | config.xml: MagicThreadDumpHost (ServerDebug element) | |
| weblogic.system.magicThreadDumps | config.xml: MagicThreadDumpEnabled (ServerDebug element) | |
| weblogic.system.maxLogFileSize | config.xml: FileMinxSize (Log element) | |
| weblogic.system.nativeIO.enable | config.xml: NativeIOEnabled (Server element) | Servers: *servername:* Configuration: Tuning: Enable Native IO |
| weblogic.system.nonPrivGroup | config.xml PostBindGID (UnixMachine element) | |
| weblogic.system.nonPrivUser | config.xml PostBindUID (UnixMachine element) | |
| weblogic.system.percentSocketReaders | config.xml: ThreadPoolPercentSocketReaders (Kernel element) | Servers: *servername:* Configuration: Tuning: Socket Readers |

| weblogic.properties file Property | .xml Configuration Attribute | Console Label |
|---|---|---|
| `weblogic.system.readTimeoutMillis` | `config.xml: LoginTimeoutMillis` (`Server` element) | Servers: *`servername:`* |
| `weblogic.system.SSL.useJava` | `config.xml: UseJava` (`SSL` element) | Servers: *`servername`*: Configuration: SSL: Use Java |
| `weblogic.system.SSLListenPort` | `config.xml: ListenPort` (`SSL` element) | Servers: *`servername`*: Configuration: SSL: Listen Port |
| `weblogic.system.startupFailureIsFatal` | `config.xml FailureIsFatal` (`StartupClass` element) | |
| `weblogic.system.user` | `config.xml: SystemUser` (`Security` element) | |
| `weblogic.system.weight` | `config.xml ClusterWeight` (`Server` element) | Servers: *`servername:`* Configuration: Cluster: Cluster Weight |

CHAPTER

# B Upgrading Example Applications to WebLogic Server 8.1

This appendix presents examples of application upgrades from earlier versions of WebLogic Server. It contains the following sections:

- Upgrading the Pet Store Application from WebLogic Server 7.0 to WebLogic Server 8.1
- Upgrading the Pet Store Application from WebLogic Server 6.1 Service Pack 4 to WebLogic Server 8.1
- Upgrading the Banking Application from WebLogic Server 5.1 to WebLogic Server 8.1

**Note:** WebLogic Server 8.1 examples and PetStore are configured to use the default security configuration. It is not possible to run the WebLogic Server 8.1 examples and PetStore using Compatibility security.

## Terms Used in This Document

Where several versions of WebLogic Server are being discussed, the instructions use version-specific terms for WebLogic home directories.

In this document WL_HOME signifies a root directory of one of your WebLogic Server installations.

For 6.0 WL_HOME=D:\WLS_6.0\wlserver6.0

For 6.1 WL_HOME=D:\WLS_6.1\wlserver6.1

For 7.0 WL_HOME=D:\WLS_7.0\weblogic700

For 8.1 WL_HOME=D:\WLS_8.0\weblogic810

# Upgrading the Pet Store Application from WebLogic Server 7.0 to WebLogic Server 8.1

The procedures described below assume that WebLogic Server 7.0 and WebLogic Server 8.1 are both installed.

The procedure involves moving your application domain to a new directory and then updating paths in the config.xml and start scripts.

You may not need to move your domain. BEA Systems recommends that application domains reside outside the WebLogic Server installation directory.

1. Create a new directory in which to upgrade the WebLogic Server 7.0 domain to a WebLogic Server 8.1 domain. In this walkthrough, the new directory is called C:\petstorefrom70to81.
2. From the WebLogic Server 7.0 installation, copy the WL_HOME\samples\server directory and its contents to C:\petstorefrom70to81.
3. Open the config.xml file in C:\petstorefrom70to81\server\config\petstore. Make the following edits.

   a. Edit the application paths to point to C:\petstorefrom70to81. Replace:

      ```
      Path="WL_HOME\samples\server\stage\petstore\petstore.ear"
      TwoPhase="true">
      ```

      with:

```
Path="c:\petstorefrom70to81\server\stage\petstore\petstore.ear" TwoPhase="true">
```

b. For `petstoreadmin.ear`, `opc.ear`, and `supplier.ear`, and `tour.war`, likewise replace the WebLogic Server 7.1 `WL_HOME` path with the `c:\petstorefrom70to81` path.

c. Change the path to the Java compiler, if necessary.

4. Open the `startpetstore.cmd` (or .sh) script in `C:\petstorefrom70to81\server\config\petstore` and make the following changes:

   a. Change %JAVA_HOME% from the JDK your WebLogic Server 7.0 installation uses to the JDK your WebLogic Server 8.1 installation uses. For example:

      set JAVA_HOME=WL_HOME\jdk131_03

      becomes:

      set JAVA_HOME=WL_HOME\jdk141

   b. Change %SAMPLES_HOME% from the WebLogic Server 7.0 \samples directory to the WebLogic Server 8.1 \samples directory. For example:

      ```
      set SAMPLES_HOME=WL_HOME\samples
      ```

      becomes:

      ```
      set SAMPLES_HOME=C:\petstorefrom70to81
      ```

   c. Change the `startWLS.cmd` (or `.sh`) path from the WebLogic Server 7.0 installation to the WebLogic Server 8.1 installation:

      ```
      call "C:\bea70sp1\weblogic700\server\bin\startWLS.cmd"
      ```

   d. Under `JAVA_OPTIONS`, change the `cacerts` path from the WebLogic Server 7.1 installation to the WebLogic Server 8.1 installation.

      ```
      -Dweblogic.security.SSL.trustedCAKeyStore=WL_HOME\server\lib\cacerts
      ```

5. Test the upgrade by starting Pet Store using the `startpetstore.cmd` or `.sh` command, and then browse to http://localhost:7001/petstore to run the application.

# Upgrading the Pet Store Application from WebLogic Server 6.1 Service Pack 4 to WebLogic Server 8.1

The procedures described below assume that WebLogic Server 6.1 and WebLogic Server 8.1 are both installed.

The procedure involves moving your application domain to a new directory and then updating paths in the config.xml and start scripts.

You may not need to move your domain. BEA Systems recommends that application domains reside outside the WebLogic Server installation directory.

1. Create a new directory in which to upgrade the WebLogic Server 6.1 domain to a WebLogic Server 8.1 domain. In this walkthrough, the new directory is called `C:\petstorefrom61to81`.
2. Copy these three directories and their contents from the WebLogic Server 6.1 `WL_HOME` directory to `C:\petstorefrom61to81`:

   ```
   WL_HOME\config
   WL_HOME\pstore
   WL_HOME\samples
   ```

3. Open the config.xml file in `C:\petstorefrom61to81\server\config\petstore`. Make the following edits.

   a. Edit the application paths to point to `C:\petstorefrom61to81`. Replace:

      ```
      Path="WL_HOME\config\petstore\applications">
      ```

      with:

      ```
      Path="c:\petstorefrom61to81\config\petstore\applications">
      ```

      Perform this substitution for the `petstore.ear` and `petstoreadmin.ear` paths as well.

b. For the server's `RootDirectory` setting, likewise replace the WebLogic Server 7.1 `WL_HOME` path with the `c:\petstorefrom70to81` path. That is, replace:

```
Name="petstoreServer" RootDirectory="WL_HOME"
```

with

```
Name="petstoreServer" RootDirectory="C:\petstore61to81"
```

c. Change the path to the Java compiler, if necessary.

4. Open the `startpetstore.cmd` (or .sh) script in `C:\petstorefrom61to81\config\petstore` and make the following changes:

a. Remove the line:

cd ..\..

a. Change %JAVA_HOME% from the JDK your WebLogic Server 7.0 installation uses to the JDK your WebLogic Server 8.1 installation uses. For example:

```
set JAVA_HOME=WL_HOME\jdk131
```

becomes:

```
set JAVA_HOME=WL_HOME\jdk141
```

b. Change `%BEA_HOME%` from the WebLogic Server 6.1 home directory to the WebLogic Server 8.1 home directory.

a. Edit the line that checks which directory contains the script from:

```
if not exist lib\weblogic.jar goto wrongplace
```

to read:

```
if not exist
C:\bea81Dec4beta\weblogic81b\server\lib\weblogic.jar goto
wrongplace
```

b. Change the classpath from

```
set
CLASSPATH=.;.\lib\weblogic_sp.jar;.\lib\weblogic.jar;.WL_HOM
E61\samples\eval\cloudscape\lib\cloudscape.jar;.\config\petS
tore\serverclasses
```

to

```
set
CLASSPATH=.;WL_HOME81\server\lib\weblogic.jar;.C:\petstorefr
```

```
om61to81\samples\eval\cloudscape\lib\cloudscape.jar;C:\petst
orefrom61to81\config\petstore\serverclasses
```

5. Test the upgrade by starting Pet Store using the `startpetstore.cmd` or `.sh` command, and then browse to http://localhost:7001/estore to run the application.

# Upgrading the Banking Application from WebLogic Server 5.1 to WebLogic Server 8.1

This example shows a successful upgrade from WebLogic Server 5.1 to WebLogic Server 8.1. The following example assumes an application running on WebLogic Server 5.1 SP12. The example is a simple banking application.

The following two main steps will migrate the banking application from WebLogic Server 5.1 to WebLogic Server 8.1:

- Convert the weblogic.properties File
- Configure the Banking Application for WebLogic Server 8.1

## Convert the weblogic.properties File

Use the WebLogic 8.1 Administration Console to convert the WebLogic Server 5.1 application's weblogic.properties file.

1. Launch the WebLogic Server 8.1 Examples Server from Start | Programs | BEA WebLogic Platform 8.1 Server | WebLogic Server 8.1 | Server Tour and Examples | Launch Examples Server.
2. Go to the Administration Console page. On the WebLogic Server Examples page that the Examples Server launches, click the Administration Console link. You will have to login with a username and password.
3. In the Administration Console, click "Convert weblogic.properties" to go to the converter. This conversion will write the properties in your WebLogic Server 5.1 weblogic.properties file to XML files for WebLogic Server 8.1 to use in a

domain. (For more information about Domains in WebLogic Server, see WebLogic Server Configuration Reference at http://e-docs.bea.com/wls/docs70/config_xml/overview.html.)

4. In the first page of the "Convert weblogic.properties" path ("Step 1 - Locate weblogic root"), browse to select the directory that contains the weblogic.properties file.

   The second page of the "Convert weblogic.properties" path appears.

5. From a list of available application directories, select the directories that contain your application. The `weblogic.properties` converter will convert these directories into a WebLogic Server 8.1 domain.

6. Fill in the remaining text fields.

   a. Admin Server Name (migrationserver)

   b. Output Directory (c:\banco)

   c. WebLogic Home

   d. Name for New Domain (migrationdomain)

7. Click Convert.

If the conversion is successful, a page will appear with a message something like the following:

```
New Domain name is migrationdomain
**********************************
Server Name is migrationserver
This server doesn't belong to any cluster
**********************************
Converting Server properties
Converting Server Debug Properties
Converting WebServer properties
Converting WebApp Component Properties
Converting JDBC Specific properties
Converting CORBA IIOP properties
```

```
Converting EJB Specific Properties

--- Warning Source File D:\510sp12\migrationserver\app_banking.jar
does not exist copy the correct file manually after conversion to
C:\banco\applications

Converting StartupClass properties

Converting Shutdown Class properties

Converting MailSession Properties

Converting FileT3 properties

Converting JMS properties

Converting Security Properties

Converting the PasswordPolicy properties

Converting User Group and ACL properties

Creating webApp for the servlets registerd in the properties file

Startup Scripts for the Server are created in the ResultDir C:\banco


Conversion successful.
```

# Configure the Banking Application for WebLogic Server 8.1

To deploy and run the banking application on WebLogic Server 8.1:

- Edit the startmigration Script in the banking application's directory
- Copy Banking Application Files to the Output Directory specified in the weblogic.properties converter

## Edit the startmigration Script

The `weblogic.properties` converter generated a script for starting up the banking application's domain. Specify some additional variables to run the banking application in this domain.

1. Edit the `startmigrationdomain` script, adding the following variables:

```
set
APPLICATIONS=%WL51_HOME%\config\migrationdomain\applications

set
CLIENT_CLASSES=%WL51_HOME%\config\migrationdomain\clientclasses

set
SERVER_CLASSES=%WL51_HOME%\config\migrationdomain\serverclasses

set
BANKING_WEBAPP_CLASSES=D:\banking\510sp12\migrationserver\serve
rclasses\examples\tutorials\migration\banking

set
CLOUDSCAPE_CLASSES=%WL51_HOME%\samples\eval\cloudscape\lib\clou
dscape.jar
```

2. Append these variables to the `startmigrationdomain` CLASSPATH:

```
CLASSPATH=...%APPLICATIONS%;%CLIENT_CLASSES%;%SERVER_CLASSES%;%
BANKING_WEBAPP_CLASSES%;%CLOUDSCAPE_CLASSES%
```

3. Add the following setting to the `startmigrationdomain` script, making sure to add it before the final `weblogic.Server`:

```
-Dcloudscape.system.home=WL51_HOME\eval\cloudscape\data
```

## Copy Banking Application Files to the Output Directory

Copy the application jar file and the web application classes and files to the banking directory.

Copy `AccountDetail.jsp`, `error.jsp`, `login.html` to `C:\banco\applications\DefaultWebApp_migrationserver`.

1. Copy `app_banking.jar` to `C:\banco\applications\`.
2. Copy `AccountDetail.jsp`, `error.jsp`, `login.html` to `C:\banco\applications\DefaultWebApp_migrationserver`.

3. Copy `BankAppServlet.class` into `C:\banco\applications\DefaultWebApp_migrationserver\WEB-INF\classes`.

# Deploy and Run the Banking Application

Start the application by navigating to `c:\banco\` in a command console and entering the command `startmigration`.

See the application at http://localhost:7001/banking.

Use:

username:system

password:password

account 1000