BEA WebLogic Server® and WebLogic Express™

Release Notes

Version 10.0
Document Revised: March 30, 2007

Contents

What's New in WebLogic Server 10.0

Welcome to BEA WebLogic Server 10.0. The following sections describe new and changed functionality in this WebLogic Server™ release.

Java EE Metadata Annotations and Dependency Injection

The Java EE programming model uses the JDK 5.0 annotations feature for EJBs, Web services, and Web applications, such as Servlets and JSPs. Annotations simplify the application development process by allowing developers to specify within the Java class itself how the application component behaves in the container (such as lifecycle methods), requests for dependency injection, and so on. Annotations are an alternative to deployment descriptors that were required by older versions of Enterprise applications (J2EE 1.4 and earlier). In fact, with Java EE metadata annotations, the standard deployment descriptors are optional.

Dependency injection (DI) allows application components to declare dependencies on external resources and configuration parameters via annotations. The container reads these annotations and injects resources or environment entries into the application components. Dependency injection is simply an easier-to-program alternative to using the `javax` interfaces or JNDI APIs to look up resources.

See:

- "Enterprise JavaBeans (EJB), Version 3.0" on page 1-8
- "Web Services" on page 1-9
- "Web Applications, Servlets, and JSPs" on page 1-14
- Using Java EE Annotations and Dependency Injection in *Developing Applications with WebLogic Server*.

Core Server

Modularity and Client Support

Some aspects of WebLogic Server's file structure have been reorganized for greater flexibility. Prior to this release, the `weblogic.jar` file could be bundled with a client application to provide WebLogic Server specific value added features, such as:

- Enhanced JDBC and WLS specific JMX interfaces
- WLS T3 Client
- WLS-IIOP Client

Many WebLogic Server components that were formerly included in the `weblogic.jar` archive file are now included in separate modules. The `weblogic.jar` archive now refers to these components in the modules directory from its manifest classpath.

The `weblogic.jar` archive has never been completely self-contained and now it is decidely so. The `weblogic.jar` now includes relative manifest classpath references to the `WL_HOME/modules` directory. Because of this, `weblogic.jar` can no longer simply be moved to any new location. To use `weblogic.jar` in a different location, you have two choices:

1. Include not only `weblogic.jar` but also `WL_HOME/modules/weblogic.server.modules_10.0.0.0.jar` directly in the classpath.
2. Modify the manifest classpath of `weblogic.jar` to refer to this JAR in the correct relative location.

Note: Use the Jar Builder tool to create a consolidated `wlfullclient.jar` for client applications as described in Using the WebLogic Jar Builder Tool in *Programming Stand-alone Clients*.

Version Information about Subcomponents

The `java weblogic.version` command has a new `-verbose` optional argument that returns version information about numerous WebLogic Server components.

Security Consequences

As a consequence of the `weblogic.jar` reorganization, your existing J2SE security policies may need to be modified to control access to the new separate modules. The default J2SE security policy in WebLogic Server has been modified to grant access to the new modules.

Third-Party Libraries

The WebLogic Server classpath contains a number of 3rd party libraries that are used internally. However, the only APIs that are supported for customer use are those that are in BEA's published API Javadocs. Any use of the 3rd party classes on the WebLogic Server classpath is at your own risk; these classes are subject to change or removal at any time.

Enhanced Shared Library Support

The Java EE platform provides several mechanisms for applications to use optional packages and shared libraries. Libraries can be bundled with an application or may be installed separately for use by any application. An EAR file may contain a directory that contains libraries packaged in JAR files. The `library-directory` element of the EAR file's deployment descriptor contains the name of this directory. This feature is similar to the `APP-INF/lib` shared library feature supported in WebLogic Server. However, if both `APP-INF/lib` and `library-directory` exist, then the jars in the `library-directory` would take precedence

See Library Directories in *Developing Applications with WebLogic Server*.

JDBC and JTA

The following features are new to WebLogic JDBC and JTA in this release.

Improved JDBC Connection Monitoring and Testing

Prior to this release, WebLogic Server has relied on JDBC drivers to properly handle, or at least respond in a timely way to any DBMS connectivity failures. For certain network failure conditions, WebLogic Server was not able to determine a connection failure until the TCP/IP timeout expires. This feature provides additional connection health monitoring and testing for connections where connectivity is suspected to be broken. See Database Connection Testing Semantics in *Configuring and Managing WebLogic JDBC*.

Oracle Fast Connection Failover Support

WebLogic Server supports Oracle Fast Connection Failover. This feature offers a driver-independent way for your JDBC application to take advantage of the connection failover facilities offered by Oracle 10g. See Using WebLogic Server with Oracle RAC in *Configuring and Managing WebLogic JDBC*.

MySQL 5.0 Support

WebLogic Server bundles a driver for MySQL 5.0.x and supports MySQL 5.0.x. See Supported Database Configurations.

JTA 1.1 Support

WebLogic Server is JTA 1.1 compliant, including standard support for looking up the `TransactionSynchronizationRegistry` object in JNDI using `java:comp/TransactionSynchronizationRegistry`. BEA extends support by providing two additional global JNDI names:

`javax/transaction/TransactionSynchronizationRegistry` and `weblogic/transaction/TransactionSynchronizationRegistry`.

For more information on JTA, see Supported Programming Model and Java Transaction API and BEA WebLogic Extensions in *Programming WebLogic JTA*.

Automatic Migration of the Transaction Recovery Service

The WebLogic Server migration framework allows an administrator to configure the JTA Transaction Recovery Service (TRS) so that it is automatically migrated from the current unhealthy hosting server to a healthy active server with the help of WebLogic Server health monitoring capabilities. This capability improves the availability of the JTA TRS in a cluster because it can be quickly restarted on a redundant server should the host server fail.

For more information, see Service-Level Migration in *Using WebLogic Server Clusters*.

Cross-Domain Security for JTA

WebLogic Server's JTA implementation supports the new cross-domain security feature. See "Cross-Domain Security" on page 1-16 and Configuring Cross Domain Security in *Programming WebLogic JTA*.

Diagnostics

The WebLogic Diagnostics Framework (WLDF) and the WLDF Console Extension both have new features.

WebLogic Diagnostic Framework

- A new standard application-scoped monitor, HttpSessionDebug, enables you to inspect an HTTP session object.
- You can now perform numeric comparisons on a column of type String. If the data contained in the column is numeric, it will return the result of the comparison operation. If a column of a row contains non-numeric data and a numeric comparison operation is performed on the column, the row will not be included in the result set.

WebLogic Diagnostic Framework Console Extension

- **Custom Metrics.** You can define custom metrics on an MBean type. Once defined, a custom metric attribute for an MBean instance can be graphed just like other attributes. See

Working with Metrics Charts and Graphs in *Using the WebLogic Diagnostic Framework Console Extension*.

- **Cut, Copy, and Paste.** You can cut, copy, and paste graphs and charts. This gives you more flexibility in how you organize graphs and charts and allows you to copy and move graphs and charts from one view to another. See Working with All Charts and Graphs in *Using the WebLogic Diagnostic Framework Console Extension*.
- **Updated Visual and Interaction Design.** The WLDF Console Extension has new icons, plus various improvements to the visual tools, such as additional functionality available in the context menus.
- **Additional Global Property Settings.** On the Global Properties tab, you can now set a default viewport size (the time interval displayed in a chart) and a default zoom percentage for zooming in or out of a chart (thereby changing the viewport size). See Working with All Charts and Graphs in *Using the WebLogic Diagnostic Framework Console Extension*.

Messaging

WebLogic Server 10.0 includes the following improvements in WebLogic Server JMS:

- "Migration of JMS-Related Services" on page 1-7
- "JMS ${APPNAME} Parameter to Create Unique Runtime JNDI Names for JMS Resources" on page 1-8
- "Cross-Domain Security for JMS" on page 1-8

Migration of JMS-Related Services

The WebLogic Server migration framework allows an administrator to configure migratable targets so that certain JMS-related services can be migrated from the current unhealthy hosting server to a healthy active server. High availability is achieved by migrating a migratable target from one clustered server to another when a problem occurs on the original server. You can also manually migrate a migratable target for scheduled maintenance.

In addition to JMS servers and custom persistent stores, WebLogic Server supports service-level migration for the following JMS-related services:

- Store-and-Forward (SAF) agents
- Path services

For more information, see Service-Level Migration in *Using WebLogic Server Clusters.*

JMS ${APPNAME} Parameter to Create Unique Runtime JNDI Names for JMS Resources

In a clustered environment, JMS resources, such as connection factories, standalone destinations, and distributed destinations, can now substitute in a unique qualifier to a JNDI name at runtime to make the global JNDI names unique for those resources. The JMS `${APPNAME}` parameter is replaced at runtime with the application name of the host application being merged to the application library.

See Configuring JMS Application Modules for Deployment in *Configuring and Managing WebLogic Message Bridge*.

Cross-Domain Security for JMS

WebLogic Server's JMS implementation supports the new cross-domain security feature. See "Cross-Domain Security" on page 1-16 and Using Cross Domain Security in *Programming WebLogic JMS*.

Enterprise JavaBeans (EJB), Version 3.0

WebLogic Server 10.0 implements Version 3.0 of the Enterprise JavaBeans specification, also known as EJB 3.0.

One of the central goals of EJB 3.0 is to make it much easier to program an EJB, in particular by reducing the number of required programming artifacts and introducing a set of EJB-specific metadata annotations that make programming the bean file easier and more intuitive.

Another goal of the EJB 3.0 specification is to standardize the persistence framework and reduce the complexity of the entity bean programming model and object-relational (O/R) mapping model.

WebLogic Server 10.0 continues to support Version 2.1 of the EJB specification.

For information on programming session EJBs using the new 3.0 programming model, see Programming WebLogic Enterprise JavaBeans, Version 3.0.

WebLogic Server 10.0 also provides an implementation of the BEA Kodo product. This provides support for JPA and JDO as well as value-added functionality.

For information on programming entity beans, see Java Persistence API. This guide is part of the larger BEA Kodo documentation set.

Upgrading EJB 2 Beans to EJB 3

Note that in EJB 2, the default transaction attribute for session beans or Message-Driven Beans is `supports`. This attribute has changed to `required` in EJB 3. If you convert session beans or MDBs from EJB2 to EJB3, take care to ensure that your classes and methods have the proper transactional behavior.

There is no change for EJB 2 session beans, MDBs, or entity beans deployed in this release of WebLogic Server.

Message-Driven Bean Startup Behavior

By default, Message-Driven Beans (MDBs) that are not part of an EAR will not accept connections until the server enters RUNNING mode, unless the `start-mdbs-with-application` flag is set to false. This change causes MDBs to have the same startup behavior whether they are deployed in an EAR or not. This change is intended to prevent MDBs from firing prematurely.

Web Services

Web Services include new and changed features, as described in the following sections:

- "Implementation and Update of Java EE 5 Web Services Specifications" on page 1-9
- "New and Updated of WS-* Specification Implementations" on page 1-10
- "Standalone Client JAR File" on page 1-10
- "Support for MTOM/XOP in JAX-RPC" on page 1-10
- "Web Services Ant Task Changes" on page 1-11
- "Deprecated Web Services Features" on page 1-12

Implementation and Update of Java EE 5 Web Services Specifications

As part of the Java EE 5 compliance, WebLogic Server includes the following new Web Services specification implementations:

- **Java API for XML-Based Web Services (JAX-WS) 2.0**

- **Java Architecture for XML Binding (JAXB) 2.0**

WebLogic Server also includes the following specification updates:

- **Web Services for Java EE 1.2**
- **SOAP With Attachments API For Java (SAAJ) 1.3**

See Understanding WebLogic Web Services.

New and Updated of WS-* Specification Implementations

WebLogic Server includes the following additions and updates to its WS-* specification implementations:

- WS-SecureConversations 1.3
- WS-Security 1.1
- WS-SecurityPolicy 1.2
- WS-Trust 1.3

WebLogic Server includes a new set of security policy files that comply with WS-SecurityPolicy 1.2. See Configuring Message-Level Security.

Standalone Client JAR File

WebLogic Web Services now provide a standalone client JAR file that programmers can use to build and run Web Services client applications outside of the WebLogic Server container.

For details, see Invoking Web Services.

Support for MTOM/XOP in JAX-RPC

MTOM/XOP describes a method for optimizing the transmission of XML data of type `xs:base64Binary` using MIME attachments over HTTP to carry that data while at the same time allowing both the sender and the receiver direct access to the XML data without having to be aware that any MIME artifacts were used to marshal the `xs:base64Binary` data.

MTOM/XOP support is standard in JAX-WS. In this release of WebLogic Server, JAX-RPC-style Web Services also support it.

See Sending Binary Data Using MTOM/XOP.

Web Services Ant Task Changes

All three of the Web Services Ant tasks have a new attribute (`type`) that specifies whether to generate a JAX-RPC 1.1 or JAX-WS 2.0 style Web Service (`jwsc`/`wsdlc`) or client artifacts (`clientgen`).

Depending on the type of Web Service you are generating or for which you are generating client artifacts, only some of the Ant task attributes apply. The Ant task reference documentation has been updated to specify which attributes apply to which type of Web Service. See Ant Task Reference.

In addition, the WebLogic Web Services Ant tasks have been further updated as described in the following sections.

jwsc

The `jwsc` Ant task, used to compile a JWS file into a deployable Web Service, has changed as follows:

- The name of the `<xsdConfig>` child element has been changed to `<binding>`. Use of the `<xsdConfig>` element has been deprecated.

See jwsc in *WebLogic Web Services: Reference.*

clientgen

The `clientgen` Ant task has changed as follows:

- The name of the `<xsdConfig>` child element has been changed to `<binding>`. Use of the `<xsdConfig>` element has been deprecated.
- The task has a two new attributes: `sysProperty` and `catalog`.

See clientgen in *WebLogic Web Services: Reference.*

wsdlc

The `wsdlc` Ant task has the following new attributes:

- The name of the `<xsdConfig>` child element has been changed to `<binding>`. Use of the `<xsdConfig>` element has been deprecated.
- The task has a two new attributes: `typeFamily` and `wlw81CallbackGen`.

See wsdlc in *WebLogic Web Services: Reference.*

Deprecated Web Services Features

The following classes and methods in the `weblogic.xml.crypto.wss` and `weblogic.xml.crypto.wss.provider` packages are deprecated:

- `weblogic.xml.crypto.wss.SecurityTokenContextHandler` class
- `java.lang.Object getCredential()` method of `weblogic.xml.crypto.wss.provider.SecurityToken`
- `java.security.Key getSecretKey()` method of `weblogic.xml.crypto.wss.provider.SecurityToken`
- `java.security.PrivateKey getPrivateKey()` method of `weblogic.xml.crypto.wss.provider.SecurityToken`
- `java.security.PublicKey getPublicKey()` method of `weblogic.xml.crypto.wss.provider.SecurityToken`
- `void setId(java.lang.String)` method of `weblogic.xml.crypto.wss.provider.SecurityToken`

Deployment

The following deployment features were added in this release:

- Production redeployment is now fully supported for Web Services, both stateless and stateful services that use more advanced features such as conversations and reliable messaging. See Updating Applications in a Production Environment in *Deploying Applications to WebLogic Server.*
- You can specify a grace period (in seconds) for processing of RMI client requests when retiring or gracefully shutting down an application. The work manager of a server instance accepts and schedules RMI calls until the grace period expires without a receiving new RMI client request. See Graceful Shut Down of RMI Client Request Processing in *Deploying Applications to WebLogic Server*.

Filtering Classloader

The WebLogic FilteringClassLoader enables users to configure deployment descriptors to explicitly specify packages that are always loaded from the application, rather than being loaded using the system classloader. See Using a Filtering Class Loader in *Developing Applications with WebLogic Server*.

Migration and Clustering

The following features related to migration and clustering were added in this release:

Migratable Targets Can Host Other JMS-related Services

In addition to JMS servers and custom persistent stores, WebLogic Server migratable targets also supports manual service-level migration for the following JMS-related services:

- Store-and-Forward (SAF) agents
- Path services

For more information, see Service-Level Migration in *Using WebLogic Server Clusters.*

Automatic Singleton Service Migration

Automatic singleton service migration allows the automatic health monitoring and migration of singleton services. A singleton service is a user-defined service operating within a cluster that is available on only one server at any given time.

When a migratable service fails or become unavailable for any reason (for example, because of a bug in the service code, server failure, or network failure), it is deactivated at its current location and activated on a new server.

For more information, see Automatic Singleton Service Migration in *Using WebLogic Server Clusters.*

Independent Support for Leasing in a Cluster

A database is no longer required to store leasing information that is used during server migration. For more information, see Server Migration in *Using WebLogic Server Clusters*.

Job Scheduler

The Job Scheduler functionality is an implementation of the `commonj.timer` API that can be used within a clustered environment. Job Schedulers allow cluster-wide timers to be aware of the other JVMs containing each server within the cluster and is therefor able to perform load balancing and failover. The Job Scheduler now supports MySQL, as well as Informix, DB2, Microsoft SQL Server, Oracle, and Sybase databases. For information on using the Job Scheduler, see Using the Timer API Within a Cluster in *The Timer and Work Manager API (CommonJ)*.

Using Unicast for Communications within a Cluster

WebLogic Server provides an alternative to multicast communication within a cluster. Both multicast and unicast are used to facilitate communications between cluster members. Unicast is much easier to configure and requires fewer network resources. For information on configuring unicast, see One-to-Many Communications Using Unicast in *Using WebLogic Server Clusters*.

weblogic.PurgeConfigCache Utility Removed

The `weblogic.PurgeConfigCache` utility has been removed from this release, since it is no longer needed for migration.

Web Applications, Servlets, and JSPs

Web applications, such as servlets and JSPs, include new and changed features, as described in the following sections:

- "Support for Servlet 2.5" on page 1-14
- "Support for JSF 1.2 and JSTL 1.2" on page 1-15
- "Support for JSP 2.1" on page 1-15
- "Weblogic Annotation for Web Components" on page 1-15
- "Java EE Application Client Utility" on page 1-16

Support for Servlet 2.5

As part of the Java EE 5 compliance, WebLogic Server 10.0 implements the Servlet 2.5 specification. One of the new important features introduced in Servlet 2.5 is the support for annotations and resource injection on servlets, filters, and listeners. The annotations are used to declare dependencies on external resources. The container will detect annotations on such components and inject necessary dependencies. The annotations are also used to declare configuration data in Java code without the need to define that data in a deployment descriptor file. With annotations, the standard `web.xml` deployment descriptor is now optional.

See WebLogic Annotations for Web Components in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Support for JSF 1.2 and JSTL 1.2

JSF 1.2 (JavaServer™ Faces) and JSTL 1.2 (JSP™ Standard Tag Library) packages are bundled with WebLogic Server as shared Web application libraries. These libraries can be referenced by standard Web applications that use JSF or JSTL functionality. Weblogic Server 10.0 continues to support existing web applications that use JSF 1.1 and JSTL 1.1 functionality and bundle JSF 1.1 and JSTL 1.1 packages as shared web application libraries.

See Configuring JSF and JSTL Libraries in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Support for JSP 2.1

As part of the Java EE 5 compliance, Weblogic Server 10.0 implements the JSP 2.1 specification. One of the new important features introduced in JSP 2.1 is the support for deferred expressions. Deferred expressions allow the evaluation of a JSP 2.1 expression to be deferred so that it can be processed by the underlying mechanism at the appropriate moment within its lifecycle. Another important new feature is that tag handlers and event listeners can be annotated for resource injection.

Consistent with the JSP 2.1 specification, JSPs now implicitly import only the following five packages: `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, and `javax.servlet.http.*`. If you have existing JSPs that implicitly imported other packages, you must now import them explicitly.

See JSP Expression Language in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Weblogic Annotation for Web Components

The WebLogic Server 10.0 provides several Weblogic specific annotations to declare configuration data in Java code for servlets and filters. The annotations include the `WLServlet`, `WLFilter`, `WLInitParam` annotations. With these annotations, servlets and filters can be developed in a web application without having to declare them in a `web.xml` deployment descriptor.

See WebLogic Annotations for Web Components in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Java EE Application Client Utility

The `weblogic.j2eeclient.Main` utility, which you can use to execute Java EE application client code, has a new option, `-clientName`. Use this option to specify the name of the client JAR to be invoked when passing in an exploded EAR. For example:

```
java weblogic.j2eeclient.Main appclient.jar t3://localhost:7001
```

For more information about `weblogic.j2eeclient.Main`, see Developing a Java EE Application Client (Thin Client) in *Programming Stand-alone Clients*.

Spring Framework Support

WebLogic Server supports the deployment and use of applications that use the 2.0.2 version of the Spring Framework. For details, see Spring Applications Reference.

WebLogic Tuxedo Connector

This release of WebLogic Tuxedo Connector includes support for:

- Using embedded Views when using FML. See Using FML with WebLogic Tuxedo Connector in *WebLogic Tuxedo Connector Programmer's Guide.*
- Translating XML buffers to and from FML/FML32. See Using the XmlFmlCnv Class for XML to and From FML/FML32 in *WebLogic Tuxedo Connector Programmer's Guide*.
- Translating XML to View(32) or View(32) to XML. See Using the XmlViewCnv Class for XML to and From View(32) Translation in *WebLogic Tuxedo Connector Programmer's Guide*.

Security

The following features are new to WebLogic security in this release:

- “Cross-Domain Security” on page 1-16
- “Windows NT Authorization Provider Deprecated” on page 1-17

Cross-Domain Security

This release includes a new method of establishing security across two or more WebLogic Server domains. WebLogic Server establishes a security role for cross-domain users and uses the

WebLogic Credential Mapping security provider in each domain to store the credentials to be used by the cross-domain users. See Enabling Trust Between WebLogic Server Domains.

Windows NT Authorization Provider Deprecated

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

SNMP

This release introduces several important changes to WebLogic Server SNMP services:

- WebLogic Server now supports the SNMPv3 protocol in addition to the SNMPv1 and SNMPv2 protocols.

  In the SNMPv3 protocol, both SNMP agent and manager must encode identical credentials in their PDUs for the communication to succeed. For additional security and flexibility, WebLogic Server SNMP agents map SNMP credentials to a WebLogic Server user. The WebLogic Server security realm authenticates the user and authorizes access to monitoring data in the domain.
- You can now create multiple SNMP agents in a domain and target the agents to WebLogic Server instances.
- WebLogic Server SNMP agents now communicate through a TCP port in addition to a UDP port.
- You can now use SNMP to monitor MBeans that you create and register (custom MBeans).
- The SNMP command-line utility that is documented in the WebLogic SNMP Agent Command-Line Reference chapter of *WebLogic Server Command Reference* is deprecated as of WebLogic Server 10.0. Instead, use the SNMP command-line utility that is documented in the WebLogic SNMP Command-Line Utility chapter of *WebLogic SNMP Management Guide*.
- You can now change configuration options for SNMP agents—including port numbers—without needing to restart WebLogic Server. Before this release, most changes required a restart.

For more information, see Understanding the WebLogic Server SNMP Agents and MIB in *SNMP Management Guide*.

WebLogic Scripting Tool (WLST)

WebLogic Server release 10.0 introduces only minor changes to WLST:

- In the `storeUserConfig` command, *`userConfigFile`* and *`userKeyFile`* are now optional arguments. If you do not specify these arguments, WLST stores the configuration and key files in a default location. See `storeUserConfig` in *WebLogic Scripting Tool*.
- You can now use the `migrate` command to migrate only JMS-related services. See `migrate` in *WebLogic Scripting Tool*.
- The `connect` command adds the `timeout` argument to specify the number of milliseconds that WLST online commands wait to complete.

  When you invoke a WLST online command, WLST connects to an MBean server, invokes an MBean server method, and returns the results of the invocation. By default, WLST will wait up to 5 minutes for the command to return the results of the invocation. If the MBean server method does not complete (return) within the timeout period, WLST abandons its invocation attempt and notifies you. You can now change this default timeout. See `connect` in *WebLogic Scripting Tool*.
- The `deploy` command now includes the `remote` argument, which is a Boolean value specifying whether the operation will be remote from the file system that contains the source. Use this option when you are on a different machine from the Administration Server and the deployment files are already at the specified location where the Administration Server is located. This option defaults to false.
- The WLST Ant task now supports a classpath nested element that you can use if your script requires classes that are not already on the classpath. This element is the standard Ant `classpath` element. See Running WLST from Ant in *WebLogic Scripting Tool*.

Administration Console

WebLogic Server release 10.0 introduces the following changes to the Administration Console:

- You can now record your configuration actions in the Administration Console as a series of WebLogic Scripting Tool (WLST) commands and then use WLST to run the commands. For more information, see Record WLST Scripts in *WebLogic Server Administration Console Help*.
- The Administration Console adds a preference that specifies the number of seconds that the Administration Server waits for a management operation to complete.

Almost all actions in the Administration Console — including logging in — require the Administration Console to connect to one or more MBean servers and invoke MBean server methods. This timeout preference specifies how long the Administration Console will wait for the method invocation to complete. If the invocation does not complete (return) within the timeout period, the Administration Console abandons its invocation attempt.

- Administration Console extensions must be packaged as a WAR file instead of a JAR file. Many console-extension features are unsupported unless you follow the packaging and archiving conventions of Web applications. For example, if include a properties bundle in your console extension, the Administration Console will not deploy the bundle properly unless it is located within your console extension's `root-dir/WEB-INF/classes` directory and unless the extension is archived as a WAR file. For more information, see Archiving and Deploying Console Extensions in *Extending the Administration Console*.

JMX

As of WebLogic Server 10.0, the JMX implementation supports the `jmx.remote.x.request.waiting.timeout` environment parameter. (The JMX Remote API 1.0 specification states that support for this parameter is optional.) Use this option to specify the number of milliseconds that your JMX client waits for the invocation of an MBean-server method to return. If a method does not return by the end of the timeout period, the client moves to its next set of instructions. By default, a client waits indefinitely for a method to return; if the MBean server is unable to complete an invocation, the JMX client will hang indefinitely.

For more information, see Make Remote Connections to an MBean Server in *Developing Custom Management Utilities with JMX*.

weblogic.apache Classes

The `weblogic.apache.xerces.*` classes, which have been deprecated since WebLogic Server release 9.1, have been removed from WebLogic Server as of release 10.0.

Also note that all other classes in the `weblogic.apache.*` packages have been deprecated since WebLogic Server release 9.2 and will be removed in a future release.

Instead of using these proprietary BEA classes, use the equivalent `org.apache.*` classes which you can download from `http://xerces.apache.org/xerces-j/`, or which are included in the JDK in the `com.sun.org.apache.*` packages.

Deprecated Beehive Functionality

The additional Beehive functionality that is available only in WebLogic Server, as described in Beehive Integration, is deprecated as of version 10.0 of WebLogic Server. This deprecation warning applies *only* to the functionality that is described in the Beehive Integration document, and not to Beehive functionlity described in other BEA documents or to open-source Beehive.

Deprecated EJB Functionality

In the next release, OpenJPA will provide a set of APIs for which compatibility is guaranteed. As a result of this effort, the following OpenJPA method signatures are deprecated and will change in the next release..

Table 1-1 Deprecated methods

Class	Method Signature
`org.apache.openjpa.persistence .OpenJPAEntityManager`	`public int getConnectionRetainMode();`
	`public int getRestoreState();`
	`public int getDetachState();`
	`public int getAutoClear();`
	`public int getAutoDetach();`
`org.apache.openjpa.persistence .OpenJPAQuery`	`public int getOperation();`
`org.apache.openjpa.persistence .jdbc.JDBCFetchPlan`	`public int getEagerFetchMode();`
	`public int getSubclassFetchMode();`
	`public int getResultSetType();`
	`public int getFetchDirection();`
	`public int getJoinSyntax();`
`org.apache.openjpa.persistence .jdbc.EagerFetchMode`	`EagerFetchType value() default EagerFetchType.NONE;`
`org.apache.openjpa.persistence .jdbc.SubclassFetchMode`	`EagerFetchType value() default EagerFetchType.NONE;`

Standards Support

This release of WebLogic Server supports the following standards.

Table 1 Java Standards Support

Standard	Version
Java EE	5.0
JDKs	5.0 (aka 1.5), 1.4 (clients only)
Java EE Enterprise Web Services	1.2, 1.1
Web Services Metadata for the Java Platform	2.0, 1.1
Java API for XML-Based Web Services (JAX-WS)	2.0
Java EE EJB	3.0, 2.1, 2.0, and 1.1
Java EE JMS	1.1, 1.0.2b
Java EE JDBC (with third-party drivers)	3.0
MS SQL jDriver	1.0
Oracle OCI jDriver	1.0 and some 2.0 features (batching)
Java EE JNDI	1.2
OTS/JTA	1.2 and 1.1
Java EE Servlet	2.5, 2.4, 2.3, and 2.2
Java EE Application Deployment	1.2
Java Authorization Contract for Containers (JACC)	1.1
Java EE JSP	2.1, 2.0, 1.2, and 1.1
RMI/IIOP	1.0

Table 1 Java Standards Support

JMX	1.2, 1.0
JavaMail	1.2
JAAS	1.0 Full
Java EE CA	1.5, 1.0
JCE	1.4
Java RMI	1.0
JAX-B	2.0
JAX-P	1.2, 1.1
JAX-RPC	1.1, 1.0
JAX-R	1.0
SOAP Attachments for Java (SAAJ)	1.3, 1.2
Streaming API for XML (StAX)	1.0

Table 2 Web Services Standards Support

Standard	Version
Java EE Enterprise Web Services	1.2, 1.1
Web Services Metadata for the Java Platform (JWS)	2.0, 1.0
Java API for XML-Based Web Services (JAX-WS)	2.0
SOAP	1.1, 1.2
WSDL	1.1
JAX-RPC	1.1
SOAP Attachments for Java (SAAJ)	1.3, 1.2

Table 2 Web Services Standards Support

Standard	Version
WS-Security	1.1, 1.0
WS-Policy	1.0
WS-SecurityPolicy	1.2, 1.1
WS-PolicyAttachment	1.0
WS-Addressing	1.0
WS-ReliableMessaging	1.0
WS-Trust	1.0
WS-SecureConversation	1.3, 1.0
UDDI	2.0
JAX-R	1.0
JAX-B	2.0

Table 3 Other Standards

Standard	Version
SSL	v3
X.509	v3
Security Assertion Markup Language (SAML)	1.0, 1.1
LDAP	v3
TLS	v1
HTTP	1.1
SNMP	SNMPv1, SNMPv2, SNMPv3

Table 3 Other Standards

Standard	Version
xTensible Access Control Markup Language (XACML)	2.0
Partial implementation of Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML	2.0
Internet Protocol (IP)	Versions: • v6 • v4 **Note:** WLS supports IPV6 as long as the system is dual-hosted with IPV4. WLS licenses use IPV4 addresses so dual-hosted IP V4/V6 servers are required. Web server plug-ins, delivered with WLS, do not support IPv6.