Oracle® Fusion Middleware

What's New in Oracle WebLogic Server

11*g* Release 1 (10.3.3)

E13852-04

April 2010

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

Note: Oracle Fusion Middleware 11*g* contains Oracle WebLogic Server 11*g*. The version number of Oracle WebLogic Server is 10.3.3.

- Section 1, "Administration Console"
- Section 2, "Core Server"
- Section 3, "Deployment"
- Section 4, "Diagnostics"
- Section 5, "Configuration Wizard"
- Section 6, "Enterprise Java Beans (EJBs)"
- Section 7, "Installation and Upgrade"
- Section 8, "JDBC"
- Section 9, "JTA"
- Section 10, "JMX"
- Section 11, "Logging"
- Section 12, "Messaging"
- Section 13, "Plug-Ins"
- Section 14, "Security"
- Section 15, "WebLogic Service Component Architecture (WebLogic SCA)"
- Section 16, "Web Applications, Servlets, and JSPs"
- Section 17, "Web Services"
- Section 18, "WLST"
- Section 19, "Deprecated Functionality (WebLogic Server 11g Release 1)"
- Section 20, "Deprecated Functionality (WebLogic Server 10.3)"
- Section 21, "Standards Support"
- Section 22, "Supported Configurations"
- Section 23, "Documentation Accessibility"

1 Administration Console

This release of WebLogic Sever includes a diagnostics monitoring dashboard. See Section 4.5, "Monitoring Dashboard and Request Performance Pages."

2 Core Server

In this release of WebLogic Server, the WebLogic Server Runtime MBean Server is configured by default to contain platform MXBeans for the corresponding server. The Domain Runtime MBean Server will contain the platform MXBeans for all of the servers in the domain.

Using the platform MBean server for the Runtime MBean Server is controlled by the `PlatformMBeanServerUsed` attribute in the JMX MBean. In previous releases, the default value for the `PlatformMBeanServerUsed` attribute was false so the platform MBean server was not used unless explicitly enabled. In this release of WebLogic Server, the default value for the `PlatformMBeanServerUsed` attribute is true for domains that are at version 10.3.3.0 or higher. For more information, see "Using the Platform MBean Server" in *Developing Custom Management Utilities With JMX for Oracle WebLogic Server*.

2.1 WebLogic Thin T3 Client

The WebLogic Thin T3 java client provides a light-weight alternative to the WebLogic Install, Full, and Thin IIOP clients. This client provides the same performance that you would see with the full client, but leverages a much smaller JAR file. The Thin T3 client supports most of the use cases in which the full client can be used.

The Thin T3 client can be used in stand-alone applications, and is also designed for applications running on foreign (non-WebLogic) servers. One common use case is integration with WebLogic JMS destinations.

For more information, see "Developing a WebLogic Thin T3 Client" in *Programming Stand-alone Clients for Oracle WebLogic Server*.

2.2 WebLogic Persistent Store

WebLogic File Store behavior and tuning have changed for default file stores and custom file stores. File stores may be used by JTA, JMS, and WS applications, among others, but the changes should be transparent to most users. The following enhancements were made in this release:

- A new synchronous write policy has been added: `Direct-Write-With-Cache`. This new policy provides the same data integrity as `Direct-Write`, but it reduces boot time and increases runtime performance in some common use cases. It also has some different behavior for backing files. In addition to the file store's primary files, the new `Direct-Write-With-Cache` write policy creates cache files in a configurable location. The location is logged in Info message 280103.

> **Note:** The optional new file store synchronous write policy `Direct-Write-With-Cache` creates new cache files in the OS user's temp directory. This new behavior may have implications related to disk space, locking, security, migration, and performance. See "Tuning the WebLogic Persistent Store" in *Performance and Tuning for Oracle WebLogic Server*.

- New configuration and tuning attributes were added that apply to all synchronous write policies, most notably an option to disable file locking (useful for some NFS environments) and options to tune native memory usage.

See "Guidelines for Configuring a Synchronous Write Policy" in *Configuring Server Environments for Oracle WebLogic Server*.

3 Deployment

In this release of WebLogic Server, you can now use a JMX API to start and stop application deployments on specified target servers. This JMX API uses open MBean data types so that no WebLogic Server classes are required on the client side.

Supporting the JMX API for starting and stopping applications are three new runtime MBeans:

- `DeploymentManagerMBean`—a run-time MBean singleton that provides access to the `AppDeploymentRuntime` MBeans for each application deployed to the domain.
- `AppDeploymentRuntimeMBean`—contains the application start and stop operations.
- `DeploymentProgressObjectMBean`—a run-time MBean that is returned from the start and stop operations; this MBean allows the client to monitor the status of the deployment operation.

In this model, you must initiate the deployment operations on the Administration Server. Consequently, these new MBeans are located in the Domain Runtime MBeanServer. For more information, refer to the *Oracle WebLogic Server MBean Reference*.

4 Diagnostics

In this release of WebLogic Server, the WebLogic Server Diagnostic Framework (WLDF) introduces the following new features:

- Section 4.1, "Oracle JRockit Flight Recorder Integration"
- Section 4.2, "WLDF Diagnostic Volume"
- Section 4.3, "Diagnostic Actions"
- Section 4.4, "WLST Commands for Downloading WLDF Diagnostic Image Capture Files"
- Section 4.5, "Monitoring Dashboard and Request Performance Pages"

4.1 Oracle JRockit Flight Recorder Integration

WebLogic Server provides specific integration points with Oracle JRockit Flight Recorder. WebLogic Server events are propagated to the Flight Recorder for inclusion in a common data set for runtime or post-incident analysis. The flight recording data is also included in WLDF diagnostic image captures, enabling you to capture flight recording snapshots based on WLDF watch rules. This full set of functionality enables you to capture and analyze runtime system information for both the JVM and the Fusion Middleware components running on it, in a single view.

For information about WLDF integration features with JRockit Flight Recorder, see "Using WLDF with Oracle JRockit Flight Recorder" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

4.2 WLDF Diagnostic Volume

This release of WebLogic Server includes a WLDF diagnostic volume setting, which controls the amount of data that is automatically produced by WebLogic Server at run time and captured in the JRockit Flight Recorder file. For general use, Oracle recommends a setting of `Low`. However, you can increase the volume of diagnostic data that is generated, as appropriate.

By default, the WLDF diagnostic volume setting is set to `Off`, but this may change in a future WebLogic Server release. For more information, see "Configuring Diagnostic Image Capture for JRockit Flight Recorder" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

4.3 Diagnostic Actions

Note the following diagnostic action changes and additions introduced in this release of WebLogic Server:

4.3.1 Change to DisplayArgumentsAction Behavior

The behavior of the DisplayArgumentsAction, which is used with custom diagnostic monitors, has been modified in this release of WebLogic Server to prevent sensitive data in your application from being inadvertently transmitted when an instrumentation event captures input arguments to, or return values from, a joinpoint.

If you need to override this behavior change, WLDF adds a new operator, the percent sign (`%`), which can be specified in pointcut expressions to designate the value of a non-static class instantiation, parameter, or return specification as not containing nor exposing sensitive information.

For more information, see "Defining Pointcuts for Custom Monitors" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

4.3.2 New Actions for Obtaining Memory Usage Statistics

WLDF provides the two new diagnostic actions that can be used in custom monitors to obtain memory usage statistics about method invocations. Both actions use the JRockit API as follows:

- `TraceMemoryAllocationAction`—Traces the amount of memory allocated by a thread during a method call. Functions similar to `TraceElapsedTimeAction`.
- `MethodMemoryAllocationStatisticsAction`—Gathers statistics about memory allocated by a thread during a method call. Functions similar to `MethodInvocationStatisticsAction`.

For more information, see "Diagnostic Action Library" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

4.4 WLST Commands for Downloading WLDF Diagnostic Image Capture Files

In this release of WebLogic Server, WLST includes the following new commands you can use for downloading the WLDF diagnostic image capture file:

- `getAvailableCapturedImages`—Returns a list of diagnostic images that have been created in the image destination directory configured on the server.
- `saveDiagnosticImageCaptureFile`—Downloads a specified diagnostic image capture file.
- `saveDiagnosticImageCaptureEntryFile`—Downloads a specific entry within a diagnostic image capture.

Note that JRockit Flight Recorder (JFR) files included in the diagnostic image capture may only be viewed using the JFR graphical user interface to Oracle JRockit Mission Control. For more information, see "Using WLDF with Oracle JRockit Flight Recorder" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

For details about and examples of each command, see *WebLogic Scripting Tool Command Reference*.

4.5 Monitoring Dashboard and Request Performance Pages

The WLDF Console Extension has been removed from the WebLogic Server Administration Console and has been replaced by the following:

- Monitoring Dashboard—Provides views and tools for graphically presenting diagnostic data about servers and applications running on them. The underlying functionality for generating, retrieving, and persisting diagnostic data is provided by the WebLogic Diagnostic Framework. The Monitoring Dashboard provides additional tools for presenting that data in a wide range of built-in and custom views.

  For more information, see "Using the Monitoring Dashboard" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.
- Diagnostics Request Performance page—Displays information about the real-time and historical views of method performance that has been captured by WLDF instrumentation capabilities.

  For more information, see "Creating Request Performance Data" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

5 Configuration Wizard

This section describes changes and new features for the Oracle Fusion Middleware Configuration Wizard.

5.1 Configuring JMS Distributed Destination Types in Configuration Wizard

The ability to change the distributed destination type for JMS system resources from the default, Weighted Distributed Destinations (WDD), to Uniform Distributed Destinations (UDD) has been added to the Oracle Fusion Middleware Configuration Wizard. To do so, select the JMS Distributed Destination option on the Select Optional Configuration screen, which subsequently displays the Select JMS Distributed Destination Type screen in the wizard.

For more information, see "Select JMS Distributed Destination Type" in *Creating Domains Using the Configuration Wizard*.

6 Enterprise Java Beans (EJBs)

This section describes changes and new features for the WebLogic Server Enterprise Java Beans (EJBs).

6.1 MDB Configuration using Activation Properties

This release extends the available Activation configuration properties to support most configurations available in the `weblogic-ejb-jar.xml` file. See "Deployment Elements for MDBs" in *Programming Message-Driven Beans for Oracle WebLogic Server*.

7 Installation and Upgrade

This section describes changes and new features for the WebLogic Server installation and upgrade.

7.1 Development-Only Installer

Oracle provides a complete WebLogic Server installation in a ZIP file for development use only. This installation is supported on Windows, Linux, and Mac OS X systems. The extracted installation contains all the necessary artifacts you need to develop applications on WebLogic Server, but uses less disk space than a WebLogic Server installation performed in Typical mode.

For more information, see "Development-Only Installer" in the *Oracle WebLogic Server Installation Guide*.

7.2 SIP Server Examples

The WebLogic Server installation includes additional server examples for SIP server. The SIP server examples are installed if you select the Server Examples option during installation.

7.3 Coherence Installation

You can install Oracle Coherence directly from the WebLogic Server installation program by selecting Coherence Product Files. Oracle Coherence is installed by default if you select a Typical installation. If you select a Custom installation, you also have the option to install Oracle Coherence code examples.

7.4 Server Examples

The WebLogic Server Medrec and Medrec-Spring server examples have been modified to use the evaluation Derby database that is included with WebLogic Server (see the next section). They have also been modified to use Oracle TopLink as the Java Persistence Architecture (JPA) persistence provider, where such a provider is used.

7.5 Evaluation Database

The WebLogic Server installation program includes an **Evaluation Database** option on the Choose Products and Components screen. If selected, an evaluation Derby database is installed with WebLogic Server in the `*WL_HOME*\common\derby` directory. If you select a Typical installation, this component is installed by default. If you choose to install the Server Examples component, the **Evaluation Database** option cannot be deselected, as some of the server examples use the evaluation database.

8 JDBC

This section describes some of the new connection properties provided by Oracle's Type 4 JDBC Drivers in this release of WebLogic Server.

8.1 Bulk Load

Bulk Load improves and expands upon current methods for inserting mass amounts of data into a database as quickly as possible. Previously available for Oracle in the 4.0 SP2 release, support for DataDirect Bulk Load has been expanded to DB2, SQL Server and Sybase.

8.2 Freeze/Unfreeze the Statement Pool

You can now "freeze" the state of the statement pool. Once frozen, important statements in the statement pool remain in the pool and are not replaced until the connection is closed or the application "unfreezes" the state of the statement pool. By freezing the statement pool, you can ensure that your most important statements are not removed from the pool by less important statements, thereby optimizing the performance of your application.

9 JTA

This release provides a new attribute, `completion-timeout-seconds`, that tunes the maximum amount of time that can be spent in the completion (rollback or second phase of a two-phase commit) of a transaction. See "Tuning Transaction Processing" in *Programming JTA for Oracle WebLogic Server*.

10 JMX

For domains that are at version 10.3.3.0 or higher, WebLogic Server registers its runtime MBeans in the JVM's platform MBean server. If you want to change the default and use a separate MBean Server, set the `PlatformMBeanServerUsed` attribute in the JMX MBean to be `false` using either the Administration Console or WLST. For more information, see "Using the Platform MBean Server" in *Developing Custom Management Utilities With JMX for Oracle WebLogic Server*.

11 Logging

WebLogic Server introduces the Server Logging Bridge, which provides a lightweight mechanism for applications that currently use Java Logging or Log4J Logging to have their log messages redirected to WebLogic logging services. Applications can use the Server Logging Bridge with their existing configuration; no code changes or programmatic use of the WebLogic Logging APIs is required.

Two versions of the Server Logging Bridge are available:

- For applications that use Java Logging, the Server Logging Bridge exists as the `weblogic.logging.ServerLoggingHandler` object, which is an instance of the `java.util.logging.Handler` class. You configure the Server Logging

Bridge handler in a `logging.properties` file that is passed in the `weblogic.Server` startup command.

- For applications that use Log4J Logging, the Server Logging Bridge exists as the `weblogic.logging.log4j.ServerLoggingAppender` object, which is an implementation of the `org.apache.log4j.Appender` interface. You configure the Server Logging Bridge appender in a `log4j.properties` file that is placed in the application classpath.

WebLogic Server also adds the `LogMBean.ServerLoggingBridgeUseParentLoggersEnabled` attribute. When you enable this attribute, application log messages are propagated to the application's root logger and use of the Server Logging Bridge is suppressed. This attribute is disabled by default.

For more information, see "Server Logging Bridge" in *Using Logging Services for Application Logging for Oracle WebLogic Server*.

12 Messaging

This release provides the following new features:

12.1 Changes to weblogic.jms.extension API

The following internal methods of `weblogic.jms.extensions.WLMessage` have been included in Oracle's public documentation, but have been removed:

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

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

12.2 JMSDestinationAvailabilityHelper API

JMSDestinationAvailabilityHelper API provides a means for getting notifications when destinations become available or unavailable. These APIs are for advanced use cases only. Use this helper only when standard approaches for solving WebLogic distributed consumer problems have been exhausted. See "Using the JMS Destination Availability Helper APIs with Distributed Queues" in *Programming JMS for Oracle WebLogic Server*.

12.3 Persistent Store Updates

WebLogic File Store behavior and tuning have changed for default file stores and custom file stores. See Section 2.2, "WebLogic Persistent Store."

13 Plug-Ins

This section describes the new features of the version 1.1 plug-ins. The following topics are described:

- Section 13.1, "Apache Plug-In Now Supports Oracle HTTP Server"

- Section 13.2, "Standard Encryption Strength Allows Simplified Naming"
- Section 13.3, "Plug-Ins Now Use Oracle SSL Toolkit"
- Section 13.4, "Version 1.1 Plug-Ins Use Oracle Security Framework"
- Section 13.5, "Version 1.1 Plug-Ins Support IPv6"
- Section 13.6, "Version 1.1 Plug-Ins Support Two-Way SSL"

See *Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server* for information about the version 1.1 plug-ins.

13.1 Apache Plug-In Now Supports Oracle HTTP Server

In previous releases of Oracle WebLogic Server, Oracle HTTP Server required the use of the `mod_wl_ohs.so` plug-in included with Oracle HTTP Server. This plug-in is documented in *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*.

As of this release of Oracle WebLogic Server, Oracle HTTP Server is now supported by the same version 1.1 plug-in as is used for the Apache Server: `mod_wl.so`.

13.2 Standard Encryption Strength Allows Simplified Naming

Because the version 1.0 plug-ins supported both 40- and 128-bit encryption standards, the plug-in file names needed to identify which standard was supported. For example, `mod_wl_22.so` indicated 40-bit encryption and `mod_wl128_22.so` indicated 128-bit encryption.

However, the version 1.1 plug-ins support only 128-bit encryption, and the plug-in names are now simplified. For example, `mod_wl.so` is the only file name required.

13.3 Plug-Ins Now Use Oracle SSL Toolkit

The plug-ins now use the Oracle SSL toolkit for enhanced SSL support.

13.4 Version 1.1 Plug-Ins Use Oracle Security Framework

The version 1.1 plug-ins use the Oracle certified security framework, and can therefore use Oracle wallets to store SSL configuration information.

13.5 Version 1.1 Plug-Ins Support IPv6

The version 1.1 plug-ins support IPv6. The `WebLogicHost` and `WebLogicCluster` configuration parameters now support IPv6 addresses.

13.6 Version 1.1 Plug-Ins Support Two-Way SSL

The version 1.1 plug-ins provide two-way SSL support for verifying client identity. Two-way SSL is automatically enforced when WebLogic Server requests the client certificate during the handshake process.

14 Security

This section describes the following changes and new features in the WebLogic Security Service in this release of WebLogic Server:

- Section 14.1, "JDBC Connection Security Service API"
- Section 14.2, "SSL Support"
- Section 14.3, "Performance Enhancements for Security Policy Deployment"

14.1 JDBC Connection Security Service API

[The WebLogic Security Service adds a new API that can be used in custom security providers for obtaining a JDBC connection. The `JDBCConnectionService` SSPI, which is used in the provider initialization, accesses the JDBC data sources that are configured for your WebLogic domain. This capability enables your custom security providers to take advantage of full database access and database connection management capabilities provided through JDBC data sources, including multi data sources.

For more information, see "Best Practice: Use the JDBC Connection Security Service API to Obtain Database Connections" in *Developing Security Providers for Oracle WebLogic Server*.]

14.2 SSL Support

This release of WebLogic Server replaces the Certicom SSL implementation in Weblogic Server with an SSL implementation based on Java Secure Socket Extension (JSSE). JSSE is the Java standard framework for SSL and TLS and includes both blocking-IO and non-blocking-IO APIs, and a reference implementation including several commonly-trusted CAs.

Additional SSL support changes include the following:

- Support for the Certicom SSL implementation is deprecated as of this release and will eventually be removed. For this purpose, this release of WebLogic Server continues to support the Certicom SSLPlus Java version 4.0 SSL implementation, as well as RSA Cert-J version 2.1.1 and Crypto-J version 3.5.
- The `SSLMBean` has been modified in this release to support additional SSL configuration capabilities, including the ability to enable or disable the JSSE adapter. See "SSL" in *Command Reference for Oracle WebLogic Server*.

For more information, see "Secure Sockets Layer (SSL)" in *Understanding Security for Oracle WebLogic Server*.

14.3 Performance Enhancements for Security Policy Deployment

This release of WebLogic Server includes a deployment performance enhancement for Deployable Authorization providers and Role Mapping providers that are thread safe.

By default, Weblogic Server now supports thread-safe parallel modification to security policy and roles during application and module deployment. For this reason, deployable Authorization and Role Mapping providers configured in the security realm should support parallel calls. The WebLogic deployable XACML Authorization and Role Mapping providers meet this requirement.

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

Console, and via the `DeployableProviderSynchronizationEnabled` and `DeployableProviderSynchronizationTimeout` attributes of the RealmMBean.

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

15 WebLogic Service Component Architecture (WebLogic SCA)

In this release of WebLogic Server, WebLogic SCA supports the following new features:

- Enhanced data binding support in Web Service bindings:
  - SOAP attachments in TopLink/EclipseLink JAXB bindings. Both SOAP Message Transmission Optimization Mechanism (MTOM) and SOAP Messages with Attachments (SwA) are supported.
  - Java Collection Objects in TopLink/EclipseLink JAXB bindings.
- Dispatch policies for EJB service bindings

For more information about WebLogic SCA, see *Developing WebLogic SCA Applications for Oracle WebLogic Server*.

16 Web Applications, Servlets, and JSPs

This section describes changes and new Web application, servlet, and JSP features in this release of WebLogic Server.

16.1 ActiveCache

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

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

- Provides storage and replication of important application data such as session state.
- Manages the life cycle of stored objects.
- Provides serialization options which reduce heap requirements and the computational cost of deserializing session state each time it is accessed.
- Breaks large objects into smaller segments to enable more efficient access to data.
- Provides near caching, keeping a small amount of data immediately available.

For more information, see *Using ActiveCache*.

16.2 Class Caching

WebLogic Server now allows you to enable class caching. The advantages of using class caching are:

- Reduces server startup time.

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

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

17 Web Services

This section describes new and changed WebLogic Web services features in this release of WebLogic Server.

17.1 Support for Web Services Atomic Transactions

WebLogic Web services enable interoperability with other external transaction processing systems, such as WebSphere, JBoss, Microsoft .NET, and so on, through the support of the following specifications:

- Web Services Atomic Transaction (WS-AtomicTransaction) Versions 1.0, 1.1, and 1.2:
  `http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-cs-01/wstx-wsat-1.2-spec-cs-01.html`
- Web Services Coordination (WS-Coordination) Versions 1.0, 1.1, and 1.2:
  `http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01/wstx-wscoor-1.2-spec-cs-01.html`

These specifications define an extensible framework for coordinating distributed activities among a set of participants. For more information, see "Using Web Services Atomic Transactions" in *Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*.

17.2 Enhanced Support for Web Services in a Clustered Environment

WebLogic Server provides enhanced routing performance of Web service requests and responses in a clustered environment. For more information, see "Managing Web Services in a Cluster" in *Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server*.

17.3 Enhanced Monitoring of Web Services and Clients

The monitoring pages available from the WebLogic Server Administration Console to monitor runtime information for Web service and clients have been enhanced. For example, you can monitor information specific to features such as Web services atomic transactions or cluster routing for JAX-WS Web services, and Web services reliable messaging for JAX-RPC Web services. For more information, see "Monitoring Web Services and Clients" in *Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.

17.4 Attach Oracle WSM Policies to WebLogic Web Services Using Fusion Middleware Control

You can now attach Oracle Web Services Manager (WSM) policies to WebLogic Web services using Oracle Fusion Middleware Enterprise Manager Fusion Middleware Control. For more information, see:

- "Using Oracle Web Services Manager Security Policies" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
- "Attaching Policies to Web Services" in *Security and Administrator's Guide for Web Services*.

17.5 Build Database Web Services Using the EclipseLink DBWS Component

The EclipseLink DBWS component provides Java developers with a declarative Web service solution for accessing relational databases. The DBWS Builder generates the necessary configuration files based on the provided database artifacts so that EclipseLink's relational and persistence services can be combined to handle the requests. For more information, see `http://www.eclipse.org/eclipselink/dbws.php`.

17.6 Method-Level Policy Attachment Behavior Has Changed

Prior to WebLogic Server 10.3.3, if a policy was attached, via the Administration Console, to a method of one Web service, the policy was also attached to all methods of the same name for all Web services in that module.

In WebLogic Server 10.3.3, the policy is attached only to the method of the appropriate Web service.

17.7 policy: Prefix Has Been Removed From OWSM Policy Names

Prior to WebLogic Server 10.3.3, OWSM policy names included a `policy:` prefix in the available policies list displayed in the Administration Console. In WebLogic Server 10.3.3, the `policy:` prefix has been removed from OWSM policy names that are displayed in the Administration Console.

As a result, OWSM policies that were attached in WebLogic Server 10.3.1 or 10.3.2 are listed in the available policies list, even though they are attached. See "Web Services and XML Issues and Workarounds" in the Oracle WebLogic Server release notes for more information about this issue.

17.8 Web Services WSDL Tab Has Been Removed

Prior to WebLogic Server 10.3.3, you could view the WSDL for the current Web service by selecting the **Configuration > WSDL** tab. The **WSDL** tab has been removed as of WebLogic Server 10.3.3. To view the WSDL for the current Web service, select the **Testing** tab, expand the name of the Web service to view its test points, and click **?WSDL**.

For more information, see "View the WSDL of a Web Service" in the *Oracle WebLogic Server Administration Console Help*

18 WLST

The section describes new and changed WLST features in this release of WebLogic Server.

18.1 SIP Server Domain Scripts

The following WLST offline sample scripts have been added to the `*WL_HOME*\common\templates\scripts\wlst` directory:

- `basicWLSSDomain.py`
- `geo1domain.py`
- `geo2domain.py`
- `replicatedDomain.py`

These scripts create simple WebLogic SIP server domains using various SIP server domain templates, which are included with your WebLogic Server installation in the `*WL_HOME*\common\templates\domains` directory.

For more information, see "WLST Sample Offline Scripts" in *Oracle WebLogic Scripting Tool*.

19 Deprecated Functionality (WebLogic Server 11*g* Release 1)

Information about deprecated functionality for WebLogic Server 11*g* Release 1 can be found on My Oracle Support at `https://support.oracle.com/`. Enter the following document ID in the **Search Knowledge Base** field:

888028.1

20 Deprecated Functionality (WebLogic Server 10.3)

This section lists all functionality that was deprecated in WebLogic Server 10.3.

- Section 20.1, "WebLogic Server Java Utilities"
- Section 20.2, "Oracle Type 4 JDBC Driver"
- Section 20.3, "Deployment"
- Section 20.4, "OpenJPA"
- Section 20.5, "Apache Beehive Support"

20.1 WebLogic Server Java Utilities

The command line tool *EarInit*, documented in the *Command Reference for Oracle WebLogic Server*, has been deprecated in this release of WebLogic Server. As a result, you should no longer:

- Use the **DDInit** utility to generate deployment descriptors for Enterprise applications.
- Use the **ddcreate** ant task, which calls **EarInit**.

20.2 Oracle Type 4 JDBC Driver

The Oracle Type 4 JDBC driver has been deprecated in WebLogic Server.10.3. It has been removed in WebLogic Server 10.3.1. Instead of this driver, you should use the Oracle Thin Driver that is provided with WebLogic Server. For details about the Oracle Thin Driver, see "Using JDBC Drivers with WebLogic Server" in *Configuring and Managing JDBC for Oracle WebLogic Server*.

20.3 Deployment

Internal fields and methods in the following classes have been deprecated in this release of WebLogic Server, and are no longer documented.

- weblogic.deploy.api.model.WebLogicDeployableObject
- weblogic.deploy.api.model.WebLogicJ2eeApplicationObject
- weblogic.deploy.api.shared.WebLogicModuleType
- weblogic.deploy.api.tools.SessionHelper

See the following sections for a complete list.

20.3.1 weblogic.deploy.api.model.WebLogicDeployableObject

This section lists the deprecated fields, methods, and classes for weblogic.deploy.api.model.WebLogicDeployableObject.

Fields

String uri

Boolean haveAppRoot

DDRootFields ddRoot

ClassLoaderControl clf

File Plan

File plandir

DeploymentPlanBean planBean

LibrarySpec[] libraries

boolean deleteOnClose

ClassFinder resourceFinder

InputStream getDDStream()

void setDDBeanRoot()

InputStream getSteamFromParent()

Methods

LibrarySpec[] getLibraries()

WebLogicJ2EEApplicationObject getParent()

void closeGCL()

void closeResourceFinder()

void closeVJF()

Class

DDRootFields

20.3.2 weblogic.deploy.api.model.WebLogicJ2eeApplicationObject

This section lists the deprecated fields and methods for weblogic.deploy.api.model.WebLogicJ2eeApplicationObject.

Fields

ApplicationBean app

Methods

String[] getModuleUris()

void initEmbeddedModules()

void addModule()

File getModulePath

20.3.3 weblogic.deploy.api.shared.WebLogicModuleType

This section lists deprecated fields for weblogic.deploy.api.shared.WebLogicModuleType.

Fields

WebLogicModuleType CONFIG

WebLogicModuleType SUBMODULE

String MODULETYPE_EAR

String MODULETYPE_WAR

String MODULETYPE_EJB

String MODULETYPE_RAR

String MODULETYPE_CAR

String MODULETYPE_UNKNOWN

String MODULETYPE_JMS

String MODULETYPE_JDBC

String MODULETYPE_JDBC

String MODULETYPE_INTERCEPT

String MODULETYPE_CONFIG

20.3.4 weblogic.deploy.api.tools.SessionHelper

This section lists deprecated methods for weblogic.deploy.api.tools.SessionHelper.

Methods

void setDebug()

SessionHelper()

LibrarySpec registerLibrary()

LibrarySpec[] getLibraries()

void enableLibraryMerge()

void bumpVersion()

20.4 OpenJPA

OpenJPA now has a set of APIs for which compatibility is guaranteed. These are the public interfaces and annotations in the `org.apache.openjpa.persistence` and `org.apache.openjpa.persistence.jdbc` packages. To ensure this compatibility, the return type for some method signatures on these interfaces were changed in non-backward compatible ways (see Section 20.4.1, "OpenJPA Changed Method Signatures"). Other methods and fields were deprecated in OpenJPA 1.0, making it likely that they will be removed in a future release of OpenJPA (see Section 20.4.2, "OpenJPA Deprecated Methods and Fields"). Therefore, their use cannot be relied on.

Note: Only the OpenJPA interfaces and classes marked `@published` have compatibility guarantees. The OpenJPA project strives to maintain compatibility for the SPI interfaces, but does not provide any guarantees on them. Additionally, classes and interfaces navigable from the SPI interfaces may change in the future.

20.4.1 OpenJPA Changed Method Signatures

This section lists the OpenJPA changed method signatures.

Table 1 org.apache.openjpa.persistence.OpenJPAEntityManager Changed Method Signatures

Pre-1.0 method signature	Method signature for 1.0 and greater
public int getConnectionRetainMode();	public ConnectionRetainMode getConnectionRetainMode();
public int getRestoreState();	public RestoreStateType getRestoreState();
public int getDetachState();	public DetachStateType getDetachState();
public int getAutoClear();	public AutoClearType getAutoClear();
public int getAutoDetach();	public EnumSet<AutoDetachType> getAutoDetach();

Table 2 org.apache.openjpa.persistence.OpenJPAQuery Changed Method Signatures

Pre-1.0 method signature	Method signature for 1.0 and greater
public int getOperation();	public QueryOperationType getOperation();

Table 3 org.apache.openjpa.persistence.jdbc.JDBCFetchPlan Changed Method Signatures

Pre-1.0 method signature	Method signature for 1.0 and greater
public int getEagerFetchMode();	public FetchMode getEagerFetchMode();
public int getSubclassFetchMode();	public FetchMode getSubclassFetchMode();

Table 3 (Cont.) org.apache.openjpa.persistence.jdbc.JDBCFetchPlan Changed Method Signatures

Pre-1.0 method signature	Method signature for 1.0 and greater
public int getResultSetType();	public ResultSetType getResultSetType();
public int getFetchDirection();	public FetchDirection getFetchDirection();
public int getJoinSyntax();	public JoinSyntax getJoinSyntax();

Table 4 org.apache.openjpa.persistence.jdbc.EagerFetchMode

Pre-1.0 method signature	Method signature for 1.0 and greater
EagerFetchType value() default EagerFetchType.NONE;	FetchMode value() default FetchMode.NONE;

Table 5 org.apache.openjpa.persistence.jdbc.SubclassFetchMode

Pre-1.0 method signature	Method signature for 1.0 and greater
EagerFetchType value() default EagerFetchType.NONE;	FetchMode value() default FetchMode.NONE;

20.4.2 OpenJPA Deprecated Methods and Fields

This section lists the OpenJPA deprecated methods and fields.

Table 6 org.apache.openjpa.persistence

Deprecated	Use Instead
OpenJPAPersistence.EntityManager	JPAFacadeHelper
OpenJPAPersistence.EntityManagerFactory	JPAFacadeHelper
OpenJPAPersistence.toEntityManagerFactory (BrokerFactory)	JPAFacadeHelper
OpenJPAPersistence.toBrokerFactory(EntityManagerFactory)	JPAFacadeHelper
OpenJPAPersistence.toEntityManager(Broker)	JPAFacadeHelper
OpenJPAPersistence.toBroker(EntityManager)	JPAFacadeHelper
OpenJPAPersistence.getMetaData(Object)	JPAFacadeHelper
OpenJPAPersistence.getMetaData(EntityManager, Class)	JPAFacadeHelper
OpenJPAPersistence.getMetaData(EntityManagerFactory, Class)	JPAFacadeHelper
OpenJPAPersistence.fromOpenJPAObjectId(Object)	JPAFacadeHelper
OpenJPAPersistence.toOpenJPAObjectId(ClassMetaData, Object)	JPAFacadeHelper
OpenJPAPersistence.toOpenJPAObjectId(ClassMetaData, Object[])	JPAFacadeHelper
OpenJPAPersistence.toOpenJPAObjectId(ClassMetaData, Collection)	JPAFacadeHelper

Table 6 (Cont.) org.apache.openjpa.persistence

Deprecated	Use Instead
OpenJPAPersistence.fromOpenJPAObjectIdClass(Class)	JPAFacadeHelper
FetchPlan.getQueryResultCache()	FetchPlan.getQueryResultCacheEnabled()
FetchPlan.setQueryResultCache(boolean cache)	FetchPlan.setQueryResultCache()
FetchPlan.getDelegate()	FetchPlan.getDelegate() **Note:** Cast to ExtentImpl. This method pierces the published-API boundary, as does the SPI cast.
OpenJPAEntityManagerFactory.CONN_RETAIN_DEMAND	ConnectionRetainMode enum
OpenJPAEntityManagerFactory.CONN_RETAIN_TRANS	ConnectionRetainMode enum
OpenJPAEntityManagerFactory.CONN_RETAIN_ALWAYS	ConnectionRetainMode enum
OpenJPAEntityManagerFactory.getConfiguration()	OpenJPAEntityManagerFactorySPI.getConfiguration()
OpenJPAEntityManagerFactory.addLifecycleListener(Object, Class[])	OpenJPAEntityManagerFactorySPI.addLifecycleListener(Object, Class[])
OpenJPAEntityManagerFactory.removeLifecycleListener(Object)	OpenJPAEntityManagerFactorySPI.removeLifecycleListener(Object)
OpenJPAEntityManagerFactory.addTransactionListener(Object)	OpenJPAEntityManagerFactorySPI.addTransactionListener(Object)
OpenJPAEntityManagerFactory.removeTransactionListener(Object)	OpenJPAEntityManagerFactorySPI.removeTransactionListener(Object)
QueryResultCache.getDelegate()	QueryResultCache.getDelegate() **Note:** Cast to ExtentImpl. This method pierces the published-API boundary, as does the SPI cast.
Extent.getDelegate()	Extent.getDelegate() **Note:** Cast to ExtentImpl. This method pierces the published-API boundary, as does the SPI cast.
OpenJPAQuery.OP_SELECT	QueryOperationType enum
OpenJPAQuery.OP_DELETE	QueryOperationType enum
OpenJPAQuery.OP_UPDATE	QueryOperationType enum
OpenJPAQuery.FLUSH_TRUE	FlushModeType enum
OpenJPAQuery.FLUSH_FALSE	FlushModeType enum
OpenJPAQuery.FLUSH_WITH_CONNECTIONS	FlushModeType enum
OpenJPAQuery.addFilterListener(FilterListener)	OpenJPAQuerySPI.AddFilterListener(FilterListener)
OpenJPAQuery.removeFilterListener(FilterListener)	OpenJPAQuerySPI.removeFilterListener(FilterListener)

Table 6 (Cont.) org.apache.openjpa.persistence

Deprecated	Use Instead
OpenJPAQuery.addAggregateListener(AggregateListener)	OpenJPAQuerySPI.addAggregateListener(AggregateListener)
OpenJPAQuery.removeAggregateListener(AggregateListener)	OpenJPAQuerySPI.removeAggregateListener(AggregateListener)
StoreCache.getDelegate()	StoreCache.getDelegate() **Note:** Cast to ExtentImpl. This method pierces the published-API boundary, as does the SPI cast.
Generator.getDelegate()	Generator.getDelegate() **Note:** Cast to ExtentImpl. This method pierces the published-API boundary, as does the SPI cast.
OpenJPAEntityManager.CONN_RETAIN_DEMAND	ConnectionRetainMode enum
OpenJPAEntityManager.CONN_RETAIN_TRANS	ConnectionRetainMode enum
OpenJPAEntityManager.CONN_RETAIN_ALWAYS	ConnectionRetainMode enum
OpenJPAEntityManager.DETACH_FETCH_GROUPS	DetachStateType enum
OpenJPAEntityManager.DETACH_FGS	DetachStateType enum
OpenJPAEntityManager.DETACH_LOADED	DetachStateType enum
OpenJPAEntityManager.DETACH_ALL	DetachStateType enum
OpenJPAEntityManager.RESTORE_ALL	RestoreStateType enum
OpenJPAEntityManager.RESTORE_NONE	RestoreStateType enum
OpenJPAEntityManager.RESTORE_IMMUTABLE	RestoreStateType enum
OpenJPAEntityManager.DETACH_CLOSE	AutoDetachType enum
OpenJPAEntityManager.DETACH_COMMIT	AutoDetachType enum
OpenJPAEntityManager.DETACH_NONTXREAD	AutoDetachType enum
OpenJPAEntityManager.DETACH_ROLLBACK	AutoDetachType enum
OpenJPAEntityManager.CLEAR_DATASTORE	AutoCleartType enum
OpenJPAEntityManager.CLEAR_ALL	AutoCleartType enum
OpenJPAEntityManager.CALLBACK_FAIL_FAST	CallBackMode enum
OpenJPAEntityManager.CALLBACK_IGNORE	CallBackMode enum
OpenJPAEntityManager.CALLBACK_LOG	CallBackMode enum
OpenJPAEntityManager.CALLBACK_RETHROW	CallBackMode enum

Table 6 (Cont.) org.apache.openjpa.persistence

Deprecated	Use Instead
OpenJPAEntityManager.CALLBACK_ ROLLBACK	CallBackMode enum
OpenJPAEntityManager.getConfiguration()	OpenJPAEntityManagerSPI.getConfiguration()
OpenJPAEntityManager.setRestoreState(int)	OpenJPAEntityManager.setRestoreState(RestoreStateType)
OpenJPAEntityManager.setDetachState(int)	OpenJPAEntityManager.setDetachState(DetachStateType)
OpenJPAEntityManager.setAutoClear(int)	OpenJPAEntityManager.setAutoClear(AutoClearType)
OpenJPAEntityManager.setAutoDetach(int)	OpenJPAEntityManager.setAutoDetach(AutoDetachType)
OpenJPAEntityManager.setAutoDetach(int, boolean)	OpenJPAEntityManager.setAutoDetach(AutoDetachType, boolean)
OpenJPAEntityManager.isLargeTransaction()	OpenJPAEntityManager.isTrackChangesByType()
OpenJPAEntityManager.setLargeTransaction(boolean)	OpenJPAEntityManager.setTrackChangesByType(boolean)
OpenJPAEntityManager.addTransactionListener(Object)	OpenJPAEntityManagerSPI.addTransactionListener(Object)
OpenJPAEntityManager.removeTransactionListener(Object)	OpenJPAEntityManagerSPI.removeTransactionListener(Object)
OpenJPAEntityManager.getTransactionListenerCallbackMode()	OpenJPAEntityManagerSPI.getTransactionListenerCallbackMode()
OpenJPAEntityManager.setTransactionListenerCallbackMode(int)	OpenJPAEntityManagerSPI.setTransactionListenerCallbackMode(int)
OpenJPAEntityManager.addLifecycleListener(Object, Class[])	OpenJPAEntityManagerSPI.addLifecycleListener(Object, Class[])
OpenJPAEntityManager.removeLifecycleListener(Object)	OpenJPAEntityManagerSPI.removeLifecycleListener(Object)
OpenJPAEntityManager.getLifecycleListenerCallbackMode()	OpenJPAEntityManagerSPI.getLifecycleListenerCallbackMode()
OpenJPAEntityManager.setLifecycleListenerCallbackMode(int)	OpenJPAEntityManagerSPI.setLifecycleListenerCallbackMode(int)
OpenJPAEntityManager.begin()	EntityTransaction.begin()
OpenJPAEntityManager.commit()	EntityTransaction.commit()
OpenJPAEntityManager.rollback()	EntityTransaction.rollback()
OpenJPAEntityManager.isActive()	EntityTransaction.isActive()
OpenJPAEntityManager.commitAndResume()	OpenJPAEntityTransaction.commitAndResume
OpenJPAEntityManager.rollbackAndResume()	OpenJPAEntityTransaction.rollbackAndResume
OpenJPAEntityManager.setRollbackOnly()	EntityTransaction.setRollbackOnly()
OpenJPAEntityManager.setRollbackOnly(Throwable)	OpenJPAEntityTransaction.setRollbackOnly()

Table 6 (Cont.) org.apache.openjpa.persistence

Deprecated	Use Instead
OpenJPAEntityManager.getRollbackCause()	OpenJPAEntityTransaction.getRollbackCause()
OpenJPAEntityManager.getRollbackOnly()	EntityTransaction.getRollbackOnly()
JDBCFetchPlan.EAGER_MODE	FetchMode enum
JDBCFetchPlan.EAGER_JOIN	FetchMode enum
JDBCFetchPlan.EAGER_PARALLEL	FetchMode enum
JDBCFetchPlan.SIZE_UNKNOWN	LRSSizeAlgorithm enum
JDBCFetchPlan.SIZE_LAST	LRSSizeAlgorithm enum
JDBCFetchPlan.SIZE_QUERY	LRSSizeAlgorithm enum
JDBCFetchPlan.SYNTAX_SQL92	JoinSyntax enum
JDBCFetchPlan.SYNTAX_TRADITIONAL	JoinSyntax enum
JDBCFetchPlan.SYNTAX_DATABASE	JoinSyntax enum
JDBCFetchPlan.setEagerFetchMode(int)	JDBCFetchPlan.setEagerFetchMode(FetchMode)
JDBCFetchPlan.setSubclassFetchMode(int)	JDBCFetchPlan.setSubclassFetchMode(FetchMode)
JDBCFetchPlan.setResultSetType(int)	JDBCFetchPlan.setResultSetType(ResultSetType)
JDBCFetchPlan.setFetchDirection(int)	JDBCFetchPlan.setFetchDirection(FetchDirection)
JDBCFetchPlan.getLRSSize()	JDBCFetchPlan.getLRSSizeAlgorithm()
JDBCFetchPlan.setLRSSize(int)	JDBCFetchPlan.setLRSSizeAlgorithm(LRSSizeAlgorithm)
JDBCFetchPlan.setJoinSyntax(int)	JDBCFetchPlan.setJoinSyntax(setJoinSyntax)

20.4.3 OpenJPAEntityManager

In WebLogic Server 10*g* Release 3 (10.3), the `org.apache.openjpa.persistence.OpenJPAEntityManager` interface extends `EntityTransaction`. This relationship is deprecated; in future releases, `OpenJPAEntityManager` will not extend `EntityTransaction`.

The following provides an example of how this might impact your code:

Pre-10.3

```
OpenJPAEntityManager em = ...
EntityTransaction t = em;
```

10.3

```
OpenJPAEntityManager em = ...;
EntityTransaction t = em.getTransaction();
```

20.5 Apache Beehive Support

Apache Beehive has been deprecated as of WebLogic Server 10.3. Oracle intends to remove Apache Beehive APIs in a future WebLogic Server Version release. In preparation, we recommend that you migrate your Beehive applications and infrastructure to other frameworks such as Oracle's ADF or Java Server Faces at your earliest convenience. Note, Beehive will still be available and supported for use within WebLogic Integration and WebLogic Portal.

21 Standards Support

This release of WebLogic Server supports the following standards and versions.

21.1 Java Standards

Table 7 lists currently supported Java standards.

Table 7 Java Standards Support

Standard	Version
JAAS	1.0 Full
Java API for XML-Based Web Services (JAX-WS)	2.1, 2.0
Java Authorization Contract for Containers (JACC)	1.1
Java EE	5.0
Java EE Application Deployment	1.2
Java EE CA	1.5, 1.0
Java EE EJB	3.0, 2.1, 2.0, and 1.1
Java EE Enterprise Web Services	1.2, 1.1
Java EE JDBC	4.0, 3.0
Java EE JMS	1.1, 1.0.2b
Java EE JNDI	1.2
Java EE JSF	2.0, 1.2, 1.1
Java EE JSP	2.1, 2.0, 1.2, and 1.1
Java EE Servlet	2.5, 2.4, 2.3, and 2.2
Java RMI	1.0
JavaMail	1.4
JAX-B	2.1, 2.0
JAX-P	1.2, 1.1
JAX-R	1.0
JAX-RPC	1.1, 1.0 (deprecated)
JCE	1.4
JDKs	6.0 (aka 1.6), 5.0 (aka 1.5, clients only)
JMX	1.2, 1.0
JPA	1.0
JSR 77: Java EE Management	1.1
JSTL	1.2
OTS/JTA	1.2 and 1.1

Table 7 (Cont.) Java Standards Support

Standard	Version
RMI/IIOP	1.0
SOAP Attachments for Java (SAAJ)	1.3, 1.2
Streaming API for XML (StAX)	1.0
Web Services Metadata for the Java Platform	2.0, 1.1

21.2 Web Services Standards

Table 8 lists currently supported Web Services standards.

Table 8 Web Services Standards Support

Standard	Version
Web Services Java EE	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)	2.1, 2.0
Simple Object Access Protocol (SOAP)	1.1, 1.2
Web Services Description Language (WSDL)	1.1
Java API for XML-based RPC (JAX-RPC)	1.1, 1.0 (deprecated)
SOAP with Attachments for Java (SAAJ)	1.3, 1.2
Web Services Security (WS-Security)	1.1, 1.0
Web Services Policy Framework (WS-Policy)	1.5, 1.2
Web Services Security Policy (WS-SecurityPolicy)	1.2
Web Services Policy Attachment (WS-PolicyAttachment)	1.5, 1.2
Web Services Addressing (WS-Addressing)	1.0, 2004/2008 member submission
Web Services Reliable Messaging (WS-ReliableMessaging)	1.1, 1.0
Web Services Trust Language (WS-Trust)	1.3
Web Services Secure Conversation Language (WS-SecureConversation)	1.3
Universal Description, Discovery, and Integration (UDDI)	2.0 (deprecated in WebLogic Server 10.3.1)
Java API for XML Registries (JAX-R)	1.0
Java Architecture for XML Binding (JAX-B)	2.1, 2.0
Security Assertion Markup Language (SAML)	2.0, 1.1
SAML Token Profile	1.1, 1.0
Web Services Atomic Transaction (WS-AtomicTransactions)	1.2, 1.1, 1.0
Web Services Coordination (WS-Coordination)	1.2, 1.1, 1.0

21.3 Other Standards

Table 9 lists other standards that are supported in this release of WebLogic Server.

Table 9 Other Standards

Standard	Version
SSL	v3
X.509	v3
LDAP	v3
TLS	v1
HTTP	1.1
SNMP	SNMPv1, SNMPv2, SNMPv3
xTensible Access Control Markup Language (XACML)	2.0
Partial implementation of Core and Hierarchical Role Based Access Control (RABC) Profile of XACML	2.0
Internet Protocol (IP)	Versions: ▪ v6 ▪ v4

For more information about IPv6 support for all Fusion Middleware products, refer to the IPv6 Certification worksheet in the *Oracle Fusion Middleware 11g Release 1 (11.1.1.x) Certification Matrix* at `http://www.oracle.com/technology/software/products/ias/files/oracle%20fusion%20middleware%2011gR1%20(11.1.1.x)%20certification%20matrix.xls`.

22 Supported Configurations

For the most current information on supported configurations, refer to the Oracle Fusion Middleware Supported Configurations Central Hub at `http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html`.

23 Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Oracle Fusion Middleware What's New in Oracle WebLogic Server, 11*g* Release 1 (10.3.3)
E13852-04