Oracle® WebLogic Server

Release Notes

10*g* Release 3 (10.1.3)

July 2008

ORACLE®

Contents

What’s New in WebLogic Server

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

- "Web Services" on page 1-31
- "WLST" on page 1-37
- "WebLogic Tuxedo Connector" on page 1-37
- "Workshop" on page 1-37
- "Deprecated Functionality" on page 1-37
- "Standards Support" on page 1-48

Administration Console

This release of WebLogic Server introduces changes to the Administration Console, as described in the following sections:

- New Look and Feel
- New Console Configuration Features
- Configuration Search
- Improved Performance and Behavior
- New and Updated Console Pages
- Changes to Core Console Extensions
- Changes and Enhancements for Developing Console Extensions

New Look and Feel

The Console has been redesigned with new colors, borders, buttons, and such.

New Console Configuration Features

The following options have been added for configuring Console behavior.

Optional Domain Locking via the Change Center

The Administration Console Change Center provides a way to lock a domain configuration so you can make changes to the configuration while preventing other accounts from making changes during your edit session.

In previous releases, the Change Center domain locking feature was always enabled. It is now possible to enable or disable the feature in development domains. It is disabled by default when you create a new development domain.

See Enable and disable the domain configuration lock in *Administration Console Online Help*.

On-Demand Deployment

On the **Configuration: General** page for each domain, you can specify whether to deploy internal applications such as the Administration Console, UDDI, and the UDDI Explorer on demand (upon first access) instead of during server startup.

See Domains: Configuration: General in *Administration Console Online Help*.

Optional Confirmation Pages

By default, you are prompted for operation confirmations in a production domain but not in a development domain. A new option on the **Preferences: User Preferences** page lets you enable confirmation prompts in development domains.

See Preferences: User Preferences in *Administration Console Online Help.*

Configuration Search

You can use the new search feature in the banner toolbar region to find any WebLogic Server Configuration MBeans that contain the string you specified in their name.

See Search the configuration in *Administration Console Online Help.*

Improved Performance and Behavior

The following enhancements improve the performance, accessibility, and operational consistency of the Console.

- **Console performance enhancements**—A number of performance enhancements significantly improve the responsiveness and memory requirements for the Administration Console application.
- **Improved accessibility**—The semantic markup used to define Console pages helps assistive technologies understand the pages better.
- **Improved browser support**—Console pages are implemented to provide more consistent behavior between the Microsoft Internet Explorer and Mozilla Firefox browsers.

New and Updated Console Pages

The Administration Console has been updated with a number of new and changed pages. These are described in the following sections. For more information about these changes, see *Administration Console Online Help*.

Security Configuration

WebLogic Server now supports the use of SAML2 security providers. The Console has been updated to provide new pages for SAML2 identity asserters and credential mappers, and new server-specific security configuration pages.

New pages are also included to configure the new file-based and RDBMS security store features of WebLogic Server.

All pages that include an encrypted password field now also include confirmation fields for those passwords.

Console pages that export security configuration information to external files now check for the existence of those files before overwriting.

Credential mappings may now be edited from within the Console.

Server Control

The **Summary of Servers** page now contains two tabs: **Configuration** and **Control**.

- The **Configuration** page lists configured servers for a domain. From this page you can add new servers or access the configuration of existing ones. This page is similar to the single **Summary of Servers** pages in previous releases.
- The **Control** page lets you start, stop, suspend, and resume any server in a domain. It is similar to the existing **Control: Servers** page.

Migration

WebLogic Server now supports additional capabilities for automatically migrating failed servers and services from one server to another. The Console pages for configuring Migratable Targets have been updated to reflect those changes.

Web Services

New Console pages provide controls for configuring Reliable Messaging settings for Web Services.

SCA Deployments

The Console now supports the deployment and control of Service Component Architecture (SCA) deployments.

Console Extension Management

A Console Extension Preference page provides links for displaying information about Console extensions and provides options for enabling and disabling extensions and for displaying Console extension points. See Manage Console extensions and Display Console extension point labels in *Administration Console Online Help*.

Spring Applications

The Administration Console now includes the ability to inspect Spring applications. This feature is packaged as a Console extension named `spring-console.jar`. You must enable this extension in the Console, as described in Manage Console extensions in *Administration Console Online Help*. See also Spring Bean Task Overview in *Administration Console Online Help.*

Changes to Core Console Extensions

In this release, portions of the Administration Console have been refactored as Console extensions:

- WebLogic Tuxedo Connection pages are now located in an extension named `wtc.jar`.
- Jolt Connection Pool pages are now located in an extension named `jolt.jar`.
- Security pages to support the configuration of security settings in older (7.0) upgraded domains are now located in `compatibility-security.jar`.

Changes and Enhancements for Developing Console Extensions

The following sections describe changes of interest to Console extension developers. For more information on developing Console extensions, see *Extending the Administration Console*.

Portal Look and Feel Specifics

The Administration Console uses a new Portal look and feel, called `wlsconsole`, that is based on the WebLogic Portal 10.0 look and feel. The following list describes the main differences from WebLogic Portal 10.0:

- CSS selector prefix is renamed from `wlp-bighorn-` to `wlsc-`.
- `singlelevelmenu.jsp` and `abstractmenu.jsp` support extra Console functionality in tabs.
- The footer and header skeletons are Console specific.
- The msie classification and the borderless theme are removed.
- `book.jsp` and `window.jsp` contain conditional logic to generate extra markup needed for the gray round corner frame style.
- Two new layouts are added. `twocollayout.jsp` is a two-column layout where one column is fixed width and the other is dynamic. `nolayout.jsp` is a layout that adds no extra markup.

New Sample Look and Feel Files

Sample look and feel files are now packaged with WebLogic Server. You no longer have to copy the look and feel files from the MedRec sample application, as in previous releases. The new files are in the `$WL_HOME/server/lib/console-ext/templates` directory. The files are:

- build-new-laf.xml

  Use this ant script to create the initial set of look and feel files for your extensions. It copies the look and feel files from laftemplate.zip (see below) into a new directory and renames components, such as skeletons and skins, to match the name of the extension.

- laftemplate.zip

  This file contains the initial set of look and feel files that build-new-laf.xml (see above) expands. You should not open or use this file directly.

Miscellaneous Console Extension Updates

- A number of the Console tags (in `console-html.tld`) have been improved to remove hard coded styles and produce valid HTML.
- A number of JSP templates (in the layouts folder) have been deprecated.
- Cascading Style Sheets (CSS) are used more extensively to define visual styles and layout. This makes it easier to brand the Console. The new look and feel also now uses fewer files, simplifying development.

- The set of CSS files has changed. The CSS files are in the `framework\skins\`*`extension_name`*`\css` and `framework\skins\wlsconsole\css` folders. The CSS files are:
  - console.css – Styles for structural elements
  - general.css – General styles for the “wlsconsole” skin
  - menu.css – Menu-related styles for the “console” skin
  - window.css – Window-related styles for the “console” skin
- There is no longer a need to modify `LoginError.jsp`. It now forwards to `LoginForm.jsp`.
- Avoid using `skeletonUri` attributes in your portal files. Instead, use `presentationClass` and `presentationId` attributes, and specify styles with CSS.
- To add books or portlets that have the gray round corner frame similar to the rest of the Console, add the `presentationClass="wlsc-frame"` attribute to top level books or portlets that have a title bar.

Specifying an Alternate Help Link

The Console now supports an additional metadata tag, `helpurlpattern`, which can be used to specify how help will be sourced.

Core Server

The following features are new to WebLogic Core Server in this release:

Sun JDK 1.6 Support

This release supports Sun JDK 1.6, which delivers superior performance compared to previous versions and leverages JDK6 delivered features, such as:

- Thread synchronization improvements, thereby delivering increased scaling in the number of concurrent users supported and more reliable and stable code.
- Web container supports multiple scripting languages, such as PHP, Groovy, and Ruby. For more information, see JSR 223: Scripting for the Java Platform.
- Allows FastSwap of classes (without bouncing the class loaders in development mode), assuming that the class profile (new methods, method names) have not changed (only code

in the method changed). The benefit is that it reduces redeployment effort (for iterative development, redeployment and restart). See "FastSwap Deployment" on page 1-8.

- Clients for this release require JDK5 or later for client JVMs and a server JVM on JDK6. WLS applications built using development tools such as Eclipse, intelliJ, and Workshop, running in a separate JVM (such as JDK5) should work with this release.
- In this release, the JSP compiler uses the Java Compiler API instead of Javelin to generate byte code. (The JSP compiler will no longer depend on the Javelin framework Java class bytecode generation feature.) This replacement does not expose any new public features and should not impact JSP files in existing applications, while providing a highly-performing and reliable JSP compiler. For more information, see JSR 199: Java Compiler API.

WebLogic JarBuilder Enhancements

This release includes the following enhancements to the WebLogic JarBuilder tool:

- The `wljarbuilder.jar`, which is used to create a consolidated `wlfullclient.jar` for client applications, no longer includes the version number in the file name. The `wljarbuilder.jar` has also been changed to a manifest-only JAR that points to the latest `jarbuilder` module. For more information, see "Using the WebLogic Jar Builder Tool" in *Programming Stand-alone Clients*.
- The WebLogic JarBuilder tool supports both JDK 5-based and JDK 6-based client applications. See "Using the WebLogic JarBuilder Tool" in *Programming Stand-alone Clients.*

Deployment

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

FastSwap Deployment

With FastSwap Deployment in WLS, Java classes are redefined in-place without reloading the ClassLoader, thereby having the decided advantage of fast turnaround times. This means that you do not have to wait for an application to redeploy and then navigate back to wherever you were in the Web page flow. Instead, you can make your changes, auto compile, and then see the effects immediately.

FastSwap is only supported when WLS is running in development mode. It is automatically disabled in production mode. See Preparing Application and Modules for Deployment in *Deploying Applications to WebLogic Server*.

On-demand Deployment

There are many internal applications that are deployed during startup. These internal applications consume memory and require CPU time during deployment. This contributes to the WebLogic Server startup time and base memory footprint. Since many of these internal applications are not needed by every user, WebLogic Server has been modified to wait and deploy these applications on the first access (on-demand) instead of always deploying them during server startup. This reduces startup time and memory footprint.

See Managing Deployed Applications in *Deploying Applications to WebLogic Server*.

Generic Overrides

This feature lets you place application-specific files to be overridden into a new optional subdirectory (named `AppFileOverrides`) in the existing plan directory structure. The presence or absence of this new optional subdirectory controls whether file overrides are enabled for the deployment. If this subdirectory is present, an internal ClassFinder is added to the front of the application and module ClassLoaders for the deployment. As a result, the file override hierarchy rules follow the existing ClassLoader and resource loading rules and behaviors for applications.

Note: This mechanism is only for overriding resources and does not override classes.

See Configuring Applications for Production Deployment in *Deploying Applications to WebLogic Server*.

Diagnostics

The WebLogic Diagnostics Framework (WLDF) has new features.

Harvester Enhancements

This section describes Harvester capabilities enhancements.

Harvesting of nested complex attributes and collections

The WLDF Harvester now supports collecting and archiving data from MBean attributes that are nested bean structures or collections. This lets you collect data from nested complex data types or collections, such as lists, arrays, and maps. See Specifying Complex and Nested Harvester Attributes in *Configuring and Using the WebLogic Diagnostic Framework*.

ObjectName Pattern Matching

In prior releases, the Harvester instance specification had to be in the form of an ObjectName. This requirement could be cumbersome and error prone since the ObjectNames of WebLogic Server MBeans are complex.

To alleviate this, the WLDF now allows:

- Specification of an instance as an ObjectName in a non-canonical form.
- Specification of an instance as an ObjectName query pattern.
- The use of zero or more wildcard (*) characters in any of the values in the property list of an ObjectName, for example, Name=*.
- The use of zero or more wildcard (*) characters to replace any character sequence in a canonical ObjectName string. In this case, you must ensure that any properties of the ObjectName that do not contain wildcards are in canonical form.

See "Using Wildcards in Harvester Instance Names" in *Configuring and Using the WebLogic Diagnostic Framework*.

Harvesting from the DomainRuntime MBeanServer

The Harvester configuration now supports a `namespace` attribute, which provides the ability to harvest MBeans that reside in the DomainRuntime MBeanServer that is running on the Administrator Server. See "Harvesting from the DomainRuntime MBeanServer" in *Configuring and Using the WebLogic Diagnostic Framework*.

Watch Enhancements

This section describes Watch capabilities enhancements.

Complex Attribute Support

Similar to the support for nested complex attributes (such as a collection, an array type, or an Object with nested intrinsic attribute types) in the Harvester configuration, the Harvester Watch rules also fully support the complex drill-down syntax of the Harvester. While configuring instance based rules, object names for instances can be specified as patterns. See "Specifying

Complex Attributes in Harvester Watch Rules" in *Configuring and Using the WebLogic Diagnostic Framework*.

Wildcard Support

The wildcard support that is available for Harvester configuration is also being supported for the Watch rule specifications that have an instance name. See "Using Wildcards in Watch Rule Instance Names" in *Configuring and Using the WebLogic Diagnostic Framework.*

Namespace Support

With the integration of the DomainRuntime MBeanServer with the Harvester, it is now possible to reference run-time metrics from a Harvester Watch rule. In order to do so, the variable syntax for a Harvester Watch rule now allows the `namespace` specification, where the metric is registered. See "Creating Harvester Watch Rule Expressions" in *Configuring and Using the WebLogic Diagnostic Framework.*

Enterprise JavaBeans (EJB), Version 3.0

EJB 3.0 enhancements include new and changed features, particularly in the area of Java Persistence API (JPA), as described in the following sections:

- Management Configuration
- Persistence Configuration
- Integration of kodoc into appc
- WebLogic Kodo MBeans
- WebLogic Kodo Annotations
- Dynamic Descriptor Elements

Management Configuration

The `kodo.ManagementConfiguration` configuration property has been replaced with the following three configuration properties for configuring profiling, JMX, and the execution context name provider, respectively:

- `kodo.Profiling`
- `kodo.JMX`

- `kodo.ExecutionContextNameProvider`

For more information about the new configuration properties, see "12.1 Configuration" in the *Kodo Developer Guide*.

Persistence Configuration

The Kodo `persistence-configuration.xml` descriptor file is now parsed when running in a non-WLS environment. For more information about using the persistence-configuration.xml descriptor file, see "Using Kodo with WebLogic Server" in *Programming WebLogic Enterprise JavaBeans, Version 3.0*.

Integration of kodoc into appc

In order to provide optimal runtime performance, flexible lazy loading, and efficient, immediate dirty tracking, Kodo uses a class enhancer to add code to your persistent classes after you have written and compiled them. The enhancer post-processes the byte code generated by your Java compiler, adding the necessary fields and methods to implement the required persistence features. In WebLogic Server, this enhancement can be performed at compile time, using a standalone enhancer, or at run time when the classes are loaded. Running this process as a compile-time process can help to reduce deployment time and to catch any possible errors earlier in the compile-test cycle.

Since WebLogic Server 10.0, the `kodoc` command was used to run the Kodo class enhancer. In this release, the `kodoc` functionality is now integrated into `appc` enabling application classes to be run through the Kodo class enhancer when running `appc`. For more information, see "appc Reference" in *Programming WebLogic Enterprise JavaBeans*.

WebLogic Kodo MBeans

This release of WebLogic Server provides a set of MBeans for monitoring Kodo runtime status. These MBeans are registered automatically in the WebLogic Server Runtime MBean server.

Note: Kodo already provides a set of JMX MBeans, including MBeans that provide some of the same information made available by the new MBeans. The new MBeans have been revised to align with WebLogic Server MBean standards.

The following table describes the new WebLogic Kodo MBeans.

Table 1-1 WebLogic Kodo MBeans

MBean Name	Description
`weblogic.management.runtime.KodoPersistenceUnitRuntimeMBean`	Parent of the Kodo MBean tree. Kodo MBeans all have `KodoPersistenceUnitRuntimeMBean` as their parent.
`weblogic.management.runtime.KodoQueryCacheRuntimeBean`	Cache that stores the object IDs returned by the query executions. You can use this MBean to get the size of the cache, clear the cache, or view cache statistics.
`weblogic.management.runtime.KodoQueryCompilationCacheRuntimeBean`	Map used to cache parsed query strings. As a result of this MBean, most queries are parsed only once in Kodo, after which the cache is used. You can use this MBean to get the size of the cache or clear the cache.
`weblogic.management.runtime.KodoDataCache`	Data cache that provides significant performance increases over non-cached operation, while guaranteeing that behavior will be identical in both cached and non-cached operation. You can use this MBean to get the size of the cache, clear the cache, or view cache statistics.

WebLogic Kodo Annotations

The Session and message-driven bean implementation in WebLogic Server includes a number of proprietary specification extensions that you configure using the `weblogic-ejb-jar.xml`. In this release, a subset of these extensions can be configured using annotations.

The following WebLogic Kodo annotations are available in the `weblogic.javaee` package. For more information, see "WebLogic Kodo Annotations" in *Programming WebLogic Enterprise JavaBeans, Version 3.0*.

Table 1-2 WebLogic Kodo Annotations

Annotation	Description
`@AllowRemoveDuringTransaction`	Flag that specifies an instance can be removed during a transaction.
`@CallByReference`	Flag that specifies the parameters are passed by reference.

Table 1-2 WebLogic Kodo Annotations (Continued)

Annotation	Description
`@DisableWarnings`	Flag that specifies all warning messages are disabled.
`@Idempotent`	Number of times you want the EJB container to automatically retry a container-managed transaction method that has rolled back.
`@JMSClientID`	Client ID for the MDB when it connects to a JMS destination. Required for durable subscriptions to JMS topics.
`@JNDIName`	JNDI name of an actual EJB, resource, or reference available in WebLogic Server.
`@MessageDestinationConfiguration`	JNDI name of the JMS Connection Factory that a message-driven EJB looks up to create its queues and topics.
`@TransactionIsolation`	Method-level transaction isolation settings for an EJB.
`@TransactionTimeoutSeconds`	Timeout for transactions in seconds.

Dynamic Descriptor Elements

The following Kodo properties can be dynamically updated, without restarting WebLogic Server:

- `lockTimeout`
- `dataCacheTimeout`
- `fetchBatchSize`

Changes made to the values of these properties are saved in the `plan.xml` file can be activated without redeploying the application that contains the respective persistence units.

These settings can be modified using the Administration Console, WLST or the `weblogic.Deployer` command. For more information, see "Overview of WebLogic Server System Administration" in *Introduction to WebLogic Server and WebLogic Express*.

Guardian

The Guardian diagnostic tool is included with this release of WebLogic Server. Guardian is not enabled by default. You can enable it via the `GuardianEnabled` attribute of the `DomainMBean`, using WLST or the Administration Console. To enable Guardian from the Administration Console:

1. In the **Change Center** of the Administration Console, click **Lock & Edit**.
2. In the left pane of the Console, under **Domain Structure**, select the domain name.
3. In the right pane, select General → Configuration, then select the **Enable Oracle Guardian Agent** option.
4. To activate these changes, in the **Change Center** of the Administration Console, click **Activate Changes**.
5. Restart the server.

For more information about Guardian, see the *Guardian User Guide*.

JDBC

The following features are new to WebLogic JDBC in this release of WebLogic Server.

- JDBC 4.0 Support
- Updated WebLogic Type 4 JDBC Drivers
- Oracle 11g RAC Support

JDBC 4.0 Support

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

- SQLXML is fully supported on the server side. On the RMI client side, SQLXML is partially supported. You cannot use the `getSource` and `setResult` APIs on the client side.
- WebLogic Server JDBC supports standard Wrapper Pattern functionality and extends the functionality on the server side. The JDBC standard requires support for the Wrapper operation on the interface. WebLogic Server supports the Wrapper operation on both the interface and on the concrete class on the server side.
- WebLogic Server enhances statement pool management as follows.
  – For the `Statement` interface:

- Call `isPoolable()` always returns false
- Call `setPoolable()` does not change the poolable state.

– For the `PreparedStatement` and `CallableStatement` interfaces:

- Call `isPoolable()` returns the current poolable state, the default value is true
- Call `setPoolable()` modifies the poolable state.

– For the `PreparedStatement` or `CallableStatement` interface, the following occurs when you call the `close()` method:

- If the current poolable state is `false`, `PreparedStatement` or `CallableStatement` will be closed.
- If the current poolable state is `true`, `PreparedStatement` or `CallableStatement` reverts to the statement cache.

- Updated third-party JDBC drivers:
  - Oracle Thin driver updated from 10g to 11g
  - Pointbase database server and driver updated from 5.1 to 5.7

For more information, see the Java JDBC technology page on the Sun Web site at http://java.sun.com/javase/technologies/database/.

Updated WebLogic Type 4 JDBC Drivers

The WebLogic Type 4 JDBC drivers included with WebLogic Server are provided from DataDirect. In this release, the drivers have been updated to DataDirect Version 3.7. The following sections describe new features and changes for the WebLogic Server Type 4 JDBC Drivers:

Changes for All Drivers

- Support for the following JDBC 4.0 features:
  - Connection validation
  - Client information storage and retrieval
- IPv6 support.
- Enhanced support for returning auto-generated keys.
- Hibernate certification.

- New `ConvertNull` connection option to control how data conversions are handled for null values.
- New `QueryTimeout` connection option for setting a default query timeout.
- Performance enhancements.

For more information, refer to the documentation for each driver.

Changes for the DB2 Driver

- Support for DB2 V9.1 for Linux/UNIX/Windows, including support for the new XML data type.
- Support for DB2 v9.1 for z/OS, including support for:
  - New DB2 v9.1 for z/OS data types
  - Distributed transactions
  - SQL Procedures
  - IPv6 addresses
- Support for DB2 UDB V5R4 on iSeries including:
  - JTA support
  - 2-MB length for SQL statements
  - 128-byte column names
  - RAISE_ERROR function
  - New timestamp identifier format
  - SQLDA record changes
  - Blanks in hexadecimal literals
  - Instead of triggers
  - ENCRYPT_TDES function
- Enhanced Kerberos authentication support:
  - Support for DB2 UDB for z/OS
  - Support for MIT Kerberos for DB2 UDB for Linux/UNIX/Windows and DB2 UDB for z/OS.

- DB2-specific encryption support.
- New `EncryptionMethod` connection option for configuring data encryption across the network.
- New Database connection property synonym for the DatabaseName connection property
- Ability to use the DatabaseName connection property for all DB2 versions
- Support for new DB2 Graphic data types

For detailed information on these changes, see "The DB2 Driver" in *Type 4 JDBC Drivers*.

Changes for the Informix Driver

- Support for Informix 11
- New `FetchBufferSize` connection option for configuring the size (in bytes) of the fetch buffer.
- Support for multiple output parameters for stored procedures.
- New Database connection property synonym for the DatabaseName connection property

For detailed information on these changes, see "The Informix Driver" in *WebLogic Type 4 JDBC Drivers*.

Changes for the Oracle Driver

Note: The Oracle Type 4 JDBC driver has been deprecated as of this release of WebLogic Server. It will be removed in the next release of WebLogic Server. Instead of this deprecated driver, use the Oracle Thin Driver that is also provided with WebLogic Server. For details about the Oracle Thin Driver, see "Using Third-Party JDBC Drivers with WebLogic Server" in *Configuring and Managing WebLogic JDBC*.

- Support for Oracle 11g, including support for server result set caching and XQuery functions
- Kerberos authentication support, including support for Windows Active Directory and MIT Kerberos.
- SSL encryption support.
- Support for Insert/Update/Delete statements with a RETURNING clause.
- Asynchronous Commit support for Oracle 10g R2.

- Enhanced support for returning and inserting/updating data stored in XML columns.
- Support for UROWID data type.
- New `EncryptionMethod` connection option for configuring data encryption across the network.
- New connection options for configuring how the driver implements SSL encryption, including:
  - `ValidateServerCertificate` and `HostNameInCertificate` connection options for configuring certificate validation behavior.
  - `TrustStore` and `TrustStorePassword` connection options for configuring truststore information.
  - `KeyStore`, `KeyStorePassword`, and `KeyPassword` connection options for configuring keystore information for SSL client authentication.
- New `SysLoginRole` connection option for configuring whether the user is logged on the database with the Oracle system privilege SYSDBA or the Oracle system privilege SYSOPER.

For detailed information on these changes, see "The Oracle Driver" in *WebLogic Type 4 JDBC Drivers*.

Changes for the SQL Server Driver

- Kerberos authentication support for MIT Kerberos.
- SSL encryption support.
- New `EncryptionMethod` connection option for configuring data encryption across the network.
- New connection options for configuring how the driver implements SSL encryption, including:
  - `ValidateServerCertificate` and `HostNameInCertificate` connection options for configuring certificate validation behavior.
- `TrustStore` and `TrustStorePassword` connection options for configuring truststore information.

- New `DescribeParameters` connection option to control whether the driver will attempt to determine, at execute time, how to send String parameters to the server based on the database data type.
- New Database connection property synonym for the `DatabaseName` connection property.
- New `LongDataCacheSize` connection property controls whether the driver caches long data (images, pictures, long text, or binary data) in result sets.

For detailed information on these changes, see "The MS SQL Server Driver" in *WebLogic Type 4 JDBC Drivers*.

Changes for the Sybase Driver

- Support for Sybase 12.5.4.
  - SSL encryption support.
  - New `EncryptionMethod` connection option for configuring data encryption across the network.
- New connection options for configuring how the driver implements SSL encryption, including:
  - `ValidateServerCertificate` and `HostNameInCertificate` connection options for configuring certificate validation behavior.
  - `TrustStore` and `TrustStorePassword` connection options for configuring truststore information.
- New `LongDataCacheSize` connection option controls whether the driver caches long data (images, pictures, long text, or binary data) in results sets.
- New `PacketSize` connection option for configuring the database protocol packet size.
- New TransactionMode connection option to control how the driver delimits the start of a local transaction.
- New Database connection property synonym for the `DatabaseName` connection property.

For detailed information on these changes, see "The Sybase Driver" in *WebLogic Type 4 JDBC Drivers*.

Oracle 11g RAC Support

This release includes support for Oracle 11g RAC (Real Application Clusters) as a high-availability database solution for WebLogic Server. However, this release does not support the new Oracle Services features in Oracle 11g RAC that support XA with load balancing. Instead, WebLogic Server will continue to use its well-proven integration architecture using Multi Data Sources for XA with load balancing.

See "Using WebLogic Server with Oracle RAC" in *Configuring and Managing WebLogic JDBC*.

Lightweight WebLogic Server

This release of WebLogic Server includes the following lightweight features:

- Lightweight installer
- Net installer
- Lightweight runtime

Lightweight Installer

The WebLogic Server installer has been enhanced to provide fine-grained options for selecting software components during installation. For example, a fully functional Java EE5 (JEE5) compliant Core Application Server component is now available as a separate installable option.

This functionality provides flexibility to experienced users who want to install only the software components in which they are interested, thus reducing the disk footprint and improving run time.

Net Installer

The Net installer involves initially downloading a small software piece on the computer (in the range of 10 MB) and then running the Net installer to actually download and install WebLogic Server components. The Net installer eliminates the need to download a large single executable binaries file before actually installing the product.

Lightweight Runtime

Oracle has adopted an internal service-oriented architecture for products. Oracle refers to this initiative as microService Architecture or mSA. See http://www.bea.com/framework.jsp?CNT=msa.jsp&FP=/content/. WebLogic Server 10.0

included the first steps towards achieving the mSA vision. In this release of WebLogic Server, additional progress is achieved by allowing some services within WebLogic Server to be turned off. The benefit of excluding some services is reduced memory footprint and reduced startup time.

Logging

Logging in WebLogic Server now provides finer control of Logging Severity, down to the level of the logging source that is generating the message. This is provided via a set of Severities that are defined in the `weblogic.logging.Severitiies` class.

The Logging source can be:

- Generated Logger classes from the XML I18N catalog using `weblogic.i18ngen`.
- Instances of the Commons Logging APIs when the WebLogic Server implementation of the Commons `org.apache.commons.logging.LogFactory` interface is enabled.

The `LogMBean` interface offers two new attributes:

- `LoggerSeverity`
- `LoggerSeverityProperties`

See "Specifying Severity Level for Loggers" in *Configuring Log Files and Filtering Log Messages*.

Messaging

This release of WebLogic Server includes the following improvements in WebLogic Server JMS:

- Automatic Migration of Messaging/JMS-Related Services
- WebLogic JMS .NET Client

Automatic Migration of Messaging/JMS-Related Services

The WebLogic Server migration framework allows an administrator to specify a migratable target for JMS-related services, such as JMS servers and SAF agents. The WebLogic administrator can also configure migratable services so that will be *automatically* migrated from a failed server based on WebLogic Server health monitoring capabilities. This improves the availability of migratable JMS-related services in a cluster because those services can be quickly restarted on a redundant server should the host server fail.

See "Roadmap for Configuring Automatic Migration of JMS-Related Services" in *Using WebLogic Server Clusters*.

WebLogic JMS .NET Client

This release of WebLogic Server includes a WebLogic JMS .NET client, which is a fully-managed .NET runtime library and API that enables programmers to create .NET client applications, written in C#, that can access WebLogic JMS applications and resources.

See *Programming WebLogic JMS .NET Client Applications*.

Migration and Clustering

This release of WebLogic Server provides Asynchronous HTTP Session Replication (AsyncRep) to improve cluster performance.

AsyncRep gives you the option to choose asynchronous session replication to the secondary server. It also provides the ability to throttle the maximum size of the queue that batches up session objects before the batched replication takes place.

AsyncRep is used to specify asynchronous replication of data between a primary server and a secondary server. In addition, this option enables asynchronous replication of data between a primary server and a remote secondary server located in a different cluster according to a cluster topology of MAN.

Security

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

- SAML 2.0 and SAML Token Profile 1.1 Support
- RDBMS Security Store
- Password Validation Provider

SAML 2.0 and SAML Token Profile 1.1 Support

This release of WebLogic Server includes broad support for SAML 2.0, including support for the SAML 2.0 Web Single Sign-On (SSO) profile and the Web Services Security (WS-Security) SAML Token profile 1.1.

The SAML 2.0 Web SSO profile is part of the core set of SAML 2.0 standards, and specifies how SAML 2.0 assertions and protocols should be used to provide browser-based single sign-on between and among Identity Provider (IdP) sites and Service Provider (SP) sites.

The SAML Token profile is part of the core set of WS-Security standards, and specifies how SAML assertions can be used for Web Services security. This release of WebLogic Server supports SAML Token Profile 1.1, including support for SAML 2.0 and SAML 1.1 assertions. SAML Token Profile 1.1 is backwards compatible with SAML Token Profile 1.0.

SAML 2.0 Components

Support for SAML 2.0 is provided by several different components areas, including:

- New SAML 2.0 security providers, specifically the SAML 2.0 Credential Mapping provider and the SAML 2.0 Identity Assertion provider
- SAML 2.0 Single Sign-On Services
- Web Services support for SAML Token Profile 1.1

SAML 2.0 Security Providers

The new SAML2 Credential Mapping provider and SAML2 Identity Assertion provider generate and consume, respectively, SAML 2.0 assertions.

At least one of the providers must be configured in order to use of SAML 2.0 in this release of WebLogic Server.

Note: For some uses of SAML 2.0, the SAML authentication provider must also be configured. The SAML authentication provider enables "virtual user" functionality for both the SAML 2.0 and SAML 1.1 identity asserters.

SAML 2.0 Single Sign-On Services

This release of WebLogic Server can be configured to act as a SAML 2.0 Identity Provider, Service Provider, or both. When configuring a WebLogic Server instance as an Identity Provider, you must configure the SAML 2.0 Credential Mapper so that the Identity Provider can generate assertions. When configuring a WebLogic Server instance as a Service Provider, the SAML 2.0 Identity Assertion provider must be configured so that the Service Provider can consume assertions.

SAML 2.0 single sign-on services are configured on a per-server basis. To enable SAML 2.0 single sign-on services in two or more WebLogic Server instances in a domain, or among the Managed Servers in a cluster, you must do the following:

1. Create a domain that is configured to use the RDBMS security store.
2. Configure SAML 2.0 services identically and individually on each server instance.

For more information about configuring SAML 2.0 single sign-on service, see "Configuring Single Sign-On with Web Browsers and HTTP Clients" in *Securing WebLogic Server*.

Web Services Security SAML Token Profile 1.1

This release of WebLogic Server Web Services now supports SAML Token Profile 1.1. This feature includes support for SAML 2.0 and SAML 1.1 assertions, and is backwards compatible with SAML Token Profile 1.0 SAML tokens are configured for a Web Service through use of the appropriate WS-SecurityPolicy assertions.

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

When using SAML Token Profile, the appropriate SAML security providers must be configured (either the SAML 2.0 or SAML 1.1 Credential Mapping or IdentityAssertion providers), depending on the desired SAML version(s) and assertion usage. For more information, see "Configuring Message-Level Security" in *Securing WebLogic Web Services*.

RDBMS Security Store

This release of WebLogic Server provides the option of using an external RDBMS as a datastore that is used by the following security providers:

- XACML Authorization and Role Mapping providers
- WebLogic Credential Mapping provider
- PKI Credential Mapping provider
- The following providers for SAML 1.1:
  - SAML Identity Assertion provider V2
  - SAML Credential Mapping provider V2
- The following providers for SAML 2.0:
  - SAML 2.0 Identity Assertion provider
  - SAML 2.0 Credential Mapping provider
- Default Certificate Registry

The use of the RDBMS security store is required to use SAML 2.0 services in two or more WebLogic Server instances in a domain.

Supported RDBMS Systems

The following RDBMS systems can be used for the RDBMS security store:

- Oracle 9i Release 2 (9.2.0.5)
- Oracle 10g Release 1 (10.1.2)
- MS-SQL 2000 and 2005
- DB2 9.2 and 9.5
- PointBase RDBMS 5.1 included in WebLogic Server

RDBMS Security Store Configuration

The Configuration Wizard has been modified to allow you to create the RDBMS security store at the time you create a domain. When you boot the domain, you may set additional configuration options for the RDBMS security store from the WebLogic Administration Console.

WebLogic Server provides a set of SQL scripts for each supported RDBMS system that must be run prior to booting the domain. These scripts create the tables in the datastore that are used by the security providers.

Use of RDBMS Security Store with SAML 2.0 Services

If you are configuring SAML 2.0 services in two or more WebLogic Server instances in a domain, such as in a cluster, you must have a domain in which the RDBMS security store and a JMS topic have been configured. This enables the security information managed by the SAML 2.0 security providers to be synchronized among each server instance.

For more information about the RDBMS security store, see “Managing the RDBMS Security Store” in *Securing WebLogic Server*.

Password Validation Provider

WebLogic Server now includes a Password Validation provider, which can be configured with one of the following authentication providers to enforce a set of configurable password composition rules:

- WebLogic Authentication provider

- SQL Authenticator provider
- LDAP Authentication provider
- Active Directory Authentication provider
- iPlanet Authentication provider
- Novell Authentication provider
- Open LDAP Authentication provider

When a password is created or modified using an authentication provider that has been configured with the Password Validation provider, the password is automatically validated against a set of composition rules. The password composition rules are configurable and can govern the minimum length of passwords, minimum number of alphabetic or numeric characters that are required, the number of non-alphanumeric characters that are required, and more.

The Password Validation provider is configured via the WebLogic Scripting Tool (WLST). For more information, see "Configuring Authentication Providers" in *Securing WebLogic Server*.

Spring

This section describes Spring integration improvements for this release of WebLogic Server.

Spring extension DI and AOP for WLS

The Java EE specification added Dependency Injection (DI) to Web and EJB container, and interceptors (a form of AOP) to the EJB container. The Spring container is the leader in DI and AOP. WebLogic Server is using Pitchfork to bridge with Spring to provide DI and interceptors to the WLS Java EE container. This integration not only satisfies the standard specification requirement, it also creates the possibility for extension to the JavaEE5 specification as the Spring framework provides a richer set of features in terms of DI and AOP.

WLS takes advantage of these integration features, providing the extension to the standard JavaEE5 DI and AOP. This extension provides DI and AOP to an EJB instance and Web components that include the servlet listener and filter. In order to maintain server compliance, the standard out-of-box WebLogic Server installation only provides the standard JavaEE5 DI and AOP.

Spring Console

The Spring console for this release of WebLogic Server provides useful management functionality for Spring Beans that are deployed to a WebLogic Server instance. The Spring console is implemented as an extension to the standard WebLogic Server Administration Console.

WebLogic Spring Security Integration

The WLS security system supports and extends Java EE security while providing a rich set of security providers that you can customize to integrate with different security databases or security policies. Besides using standard Java EE security, application programmers can use a wide array of proprietary extensions that allow an application to tightly integrate with the security system. WLS packages several security provider offerings.

The Spring security (`acegi`) provides security to a Spring application while providing a rich set of security providers.

For a blended J2EE and Spring application, rather than require authentication with both security frameworks, WLS and Spring security will work together. WLS security handles the authentication, and converts WLS principals to Spring GrantedAuthority through a mapper class. Once authenticated by WLS security, a user is authenticated for Spring security. You can then decide how to secure the objects in the application.

Web Applications, Servlets, and JSPs

The following features and changes are new to Web applications, servlets and JSPs in this release.

HTTP Publish-Subscribe Server

An HTTP Publish-Subscribe Server (also called pub-sub server) is a channels-based publish/subscribe mechanism for Web clients to send and receive asynchronous messages over HTTP. One of the principal uses of the pub-sub server is to build event-driven or push-based Web 2.0 internet applications that are collaborative and capable of supporting multiple listening and publishing channels with thousands of users.

For more information, see Using the HTTP Publish-Subscribe Server in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

WLS Servlet Container Debugging Enhancements

The following sections describe debugging support enhancements that have been made in the servlet container for this release.

Disabling Access Logging

In cases where access logging is not required, you can improve server performance by disabling access logging. A new optional property `disable-access-logging` has been introduced into `container-descriptor` in `weblogic.xml` to indicate if access logging is disabled.

Debugging a Specific Session

A request flag named `wl_debug_session` as well as a *same-named* session attribute have been introduced to log the changes of the current request in the current session. If either flag is used, the container logs the modifications of the underlying session into the server log.

Tracking Request Handle Footprint

The WLS servlet container provides more detailed log messages during request handling to better describe each milestone in a request flow. No additional configuration changes are required other than enabling the `HttpDebug` logger. You can then find the footprint of a request handle in the server log.

For more information about these debugging enhancements, see Debugging Servlet Containers in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Application Container Enhancements

The following sections provide information on application container features for this release.

Version-independent Namespace

Prior to this release, the namespace included version information. The version-independent namespace in this release of WebLogic Server makes WLS-specific deployment descriptors easier to use and reduces the workload of schema upgrades. The namespace has been broken into multiple namespaces, one per descriptor. The version has also been removed from the namespaces to make them more stable. A `version` attribute has also been introduced for the root element of each descriptor.

For a listing of WLS deployment descriptors and their corresponding schemas, see XML Deployment Descriptors in *Developing Applications With WebLogic Server*.

Comprehensive Deployment Descriptor View

Additional descriptors are supported in AppMerge for the deployment view in this release of WebLogic Server.

External Diagnostics Descriptor

The path of the external diagnostics descriptors is defined in the `plan.xml` file as an external entry. This feature supports the deployment view and deployment of an application or a module, detecting the presence of an external diagnostic descriptor if the descriptor is declared in the plan.

WebLogic Descriptor Modularization

WebLogic-specific descriptor Bean files and schema files are now included in `com.bea.core.desciptor.wl_VERSION.jar` and `com.bea.core.descriptor.wl.binding_VERSION.jar`. These JAR files can be found in the `modules` directory.

Performance Enhancements

The following container performance-related enhancements have been introduced in this release:

Allowing Administration Channel Traffic

The `require-admin-trafffic` element can determine whether traffic should always go through the administration channel or only when the Web application is in administrative mode.

Disabling Access Logging

The `access-logging-disabled` element can eliminate access logging of the underlying Web application, which can improve server throughput by reducing the logging overhead.

Using HTML Template Compression

The `compress-html-template` element can compress the HTML in the JSP template blocks to improve runtime performance.

Optimizing JSP Expressions

The `optimize-java-expression` element can optimize Java expressions to improve runtime performance.

For more information about these new elements, see weblogic.xml Deployment Descriptor Elements in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Improved File Session Performance

File-based HTTP session performance has been dramatically improved by a factor of 10 in this release of WebLogic Server.

Removed Servlet Features

The `docHome` parameter for FileServlet, which was deprecated in release 9.2, has been removed in this release of WebLogic Server. Use virtual directories as an alternative.

Web Services

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

- Web Service Standards
- JAX-WS Enhancements
- WS-ReliableMessaging Enhancements
- Smart Policy Selection
- WS-Security Enhancements
- XML Catalog Support
- Improved Interoperability With Microsoft Windows Communication Foundation (WCF) 3.0
- WebLogic Web Services Documentation Set

Web Service Standards

WebLogic Web Services support updated versions of the following standards:

- Java API for XML-based Web Services (JAX-WS) 2.1
- Java Architecture for XML Binding (JAXB) 2.1
- Web Services Security (WS-Security) 1.1

- Web Services Addressing (WS-Addressing) 1.0
- Web Services Policy Framework (WS-Policy) 1.5
- Web Services Security Policy (WS-SecurityPolicy) 1.2
- Web Services Reliable Messaging (WS-ReliableMessaging) 1.1
- Web Services Trust Language (WS-Trust) 1.3
- SAML 2.0
- Web Services Secure Conversation Language (WS-SecureConversation) 1.3

For more information, see "Standards Supported by WebLogic Web Services" in *Introducing WebLogic Web Services*.

JAX-WS Enhancements

Java API for XML-based Web Services (JAX-WS) 2.1 is supported in this release, adding the following features to those found in JAX-WS 2.0:

- Support for the JAXB 2.1 (JSR 222) Data Binding API
- WS-Addressing support
- Dynamic publishing of endpoints
- APIs for `EndpointReference` creation and propagation
- Annotations and APIs to enabled/disable features, such as MTOM and Addressing

The WebLogic Server implementation of JAX-WS is based on the JAX-WS Reference Implementation (RI), Version 2.1.4, and includes enhancements to the tool layer to simplify the building and deployment of JAX-WS services and to ease the migration from JAX-RPC to JAX-WS. The following features and enhancements are available from the JAX-WS RI 2.1.4:

- .NET 2.0/WSF 3.0 MTOM interoperability support
- Significant performance improvements through the use of Woodstox StAX Parser
- SOAPAction- based dispatching
- Integration of JAXB RI 2.1.5
- JAXB type substitution support

- WS-Addressing support for both W3C (1.0) and Member Submission (2004/08)
- Asynchronous client/server support
- Dispatch and provider support:
  – `Dispatch<Message>` and `Provider<Message>` support
  – Development of non-WSDL or non-SOAP endpoints, such as REST

As with WebLogic Server 10.0, developers may begin development with either a Java source file or WSDL file. The WebLogic Server Ant tasks `<jwsc>` and `<clientgen>` automate the generation of portable data binding classes, creation of deployment descriptors, and packaging.

For more information about using JAX-WS, see:

- *Getting Started With WebLogic Web Services Using JAX-WS*
- *Programming Advanced Features of WebLogic Web Services Using JAX-WS*

WS-ReliableMessaging Enhancements

Web Services Reliable Messaging (WS-ReliableMessaging) 1.1 is supported in this release. This version includes several protocol-level enhancements, but does not significantly impact how clients interact with reliable services using WebLogic Service facilities (for example, `stubs`, `WsrmUtils`).

To use the WS-ReliableMessaging 1.1 protocol in your Web Services, attach a WS-ReliableMessaging 1.1 policy to your reliable JWS using the `@Policy` annotation. (The process is similar to the process of attaching a WS-ReliableMessaging 1.0 reliable service.). For your convenience, there are several pre-packaged policy files, such as the `DefaultReliability1.1.xml` policy file, that are provided to help you get started.

The following changes have been made to the WS-ReliableMessaging 1.1 protocol:

- Use the `WsrmUtils.closeSequence` method in your reliable client to close a reliable sequence and get a full and complete accounting of the acknowledged messages in that sequence, prior to terminating the sequence.
- The concept of a “last message” has been removed for WS-RM 1.1 and the following methods are no longer supported: `WsrmUtils.setLastMessage` and `WsrmUtils.sendEmptyLastMessage`.

For more information about WS-ReliableMessaging, see “Using Web Services Reliable Messaging” in *Programming Advanced Features of WebLogic Web Services Using JAX-RPC*.

Smart Policy Selection

You can configure multiple policy alternatives—also referred to as *smart policy alternatives*—for a single Web Service by creating a custom policy file. At run time, WebLogic Server selects which of the configured policies to apply. It excludes policies that are not supported or have conflicting assertions and selects the appropriate policy, based on your configured preferences, to verify incoming messages and build the response messages.

For more information about smart policy selection, see:

- "Smart Policy Selection" in "Configuring Message-Level Security" in *Securing WebLogic Web Services*.
- "Multiple Policy Alternatives" in "Using Web Services Reliable Messaging" in *Programming Advanced Features of WebLogic Web Services Using JAX-RPC*.

WS-Security Enhancements

Version Independent Policy

WebLogic Server now supports version-independent policy. That is, the WS-Policy or WS-SecurityPolicy policy can be based on either of the supported namespaces, and can come from different sources using different namespaces. At run time, the merged policy file then contains two or more different namespaces. See Version-Independent Policy Supported in *Securing WebLogic Web Services*.

Optional Policy Assertion

WebLogic Server now supports the `Optional` WS-Policy assertion. The following security policy assertions are now supported by the `Optional` policy assertion:

- Username Token
- SAML Token
- Signature parts or signature elements
- Encryption parts or encryption elements
- Derive Key Token

See Using the Optional Policy Assertion in *Security WebLogic Web Services* for more information.

Element-Level Security

WebLogic Server supports the element-level assertions defined in WS-SecurityPolicy 1.2. These assertions allow you to apply a signature or encryption to selected elements within the SOAP request or response message, enabling you to target only the specific data in the message that requires security and thereby reduce the computational requirements.

See Configuring Element-Level Security in *Security WebLogic Web Services* for more information.

SAML Token Profile

The SAML Token Profile 1.1 is part of the core set of WS-Security standards, and specifies how SAML assertions can be used for Web Services security. WebLogic Server supports SAML Token Profile 1.1, including support for SAML 2.0 and SAML 1.1 assertions. SAML Token Profile 1.1 is backwards compatible with SAML Token Profile 1.0.

See Using Security Assertion Markup Language (SAML) Tokens for Identity in *Security WebLogic Web Services* for more information.

XML Catalog Support

WebLogic Server now supports XML catalogs with JAX-WS Web Services. An XML catalog enables your application to reference imported XML resources, such as WSDLs and XSDs, from a source that is different from that which is part of the description of the Web Service. Redirecting the XML resources in this way may be required to improve performance or to ensure your application runs properly in your local environment.

For example, a WSDL may be accessible during client generation, but may no longer be accessible when the client is run. You may need to reference a resource that is local to or bundled with your application rather than a resource that is available over the network. Using an XML catalog file, you can specify the location of the WSDL that will be used by the Web Service at run time.

For more information about using XML catalogs, see "Using XML Catalogs" in *Programming Advanced Features of WebLogic Web Services Using JAX-WS*.

To support XML Catalog, a new Ant task, `wsdlget`, is supported that enables you to download a WSDL and its imported XML targets to the local directory, such as XSD and WSDL files. For more information, see "wsdlget" in *WebLogic Web Services Reference*.

Improved Interoperability With Microsoft Windows Communication Foundation (WCF) 3.0

In conjunction with Microsoft, Oracle has performed interoperability testing to ensure that the Web Services created using WebLogic Server can access and consume Web Services created using Microsoft Windows Communication Foundation (WCF)/.NET 3.0 Framework and vice versa. For complete details, see "Interoperability With Microsoft WCF/.NET" in *Introducing WebLogic Web Services*.

WebLogic Web Services Documentation Set

The WebLogic Web Services documentation set has been reorganized, as summarized in the following table.

Table 1-3 WebLogic Web Services Documentation Set

This document . . .	Describes . . .
Introducing WebLogic Web Services	An introduction to WebLogic Web Services, the standards that are supported, interoperability information, and relevant samples and documentation.
Getting Started With WebLogic Web Services Using JAX-WS	The basic knowledge and tasks required to program a simple WebLogic Web Service using JAX-WS. The guide includes use cases and examples, iterative development procedures, typical JWS programming steps, data type information, and how to invoke a Web Service.
Programming Advanced Features of WebLogic Web Services Using JAX-WS	How to program more advanced features using JAX-WS, such as callbacks, XML Catalog, and SOAP message handlers.
Getting Started With WebLogic Web Services Using JAX-RPC	The basic knowledge and tasks required to program a simple WebLogic Web Service using JAX-RPC. The guide includes use cases and examples, iterative development procedures, typical JWS programming steps, data type information, and how to invoke a Web Service.
Programming Advanced Features of WebLogic Web Services Using JAX-RPC	How to program more advanced features using JAX-RPC, such as Web Service reliable messaging, callbacks, conversational Web Services, use of JMS transport to invoke a Web Service, and SOAP message handlers.

Table 1-3 WebLogic Web Services Documentation Set (Continued)

This document . . .	Describes . . .
Securing WebLogic Web Services	How to program and configure message-level (digital signatures and encryption), transport-level, and access control security for a Web Service.
WebLogic Web Services Reference	Reference information on JWS annotations, Ant tasks, reliable messaging WS-Policy assertions, security WS-Policy assertions, and deployment descriptors.

WLST

A `listApplications()` command, which lists all applications that are currently deployed to the domain, has been added to WLST. For more information, see "Deployment Commands" in *WebLogic Scripting Tool*.

WebLogic Tuxedo Connector

This release of WebLogic Tuxedo Connector includes:

- SSL support, including support for trusted certificates and private keys, to provide full integration with Tuxedo 10.
- Password encryption using Advanced Encryption System (AES).

For more information, see WebLogic Tuxedo Connector Administration in the *WebLogic Tuxedo Connector Administration Guide*.

Workshop

With this release of WebLogic Server, the full Workshop IDE is available.

Deprecated Functionality

WebLogic Server Java Utilities

The command line tool **EarInit**, documented in the *WebLogic Server Command Reference*, 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**.

Oracle Type 4 JDBC Driver

The Oracle Type 4 JDBC driver has been deprecated in this release of WebLogic Server. It will be removed in the next release of WebLogic Server. Instead of this deprecated driver, you should use the Oracle Thin Driver that is also provided with WebLogic Server. For details about the Oracle Thin Driver, see "Using Third-Party JDBC Drivers with WebLogic Server" in *Configuring and Managing WebLogic JDBC.*

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 table for a complete list.

`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

`weblogic.deploy.api.model.WebLogicJ2eeApplicationObject`
Fields: ApplicationBean app **Methods:** String[] getModuleUris() void initEmbeddedModules() void addModule() File getModulePath
`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

`weblogic.deploy.api.tools.SessionHelper`
Methods: void setDebug() SessionHelper() LibrarySpec registerLibrary() LibrarySpec[] getLibraries() void enableLibraryMerge() void bumpVersion()

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, some method signatures on these interfaces were removed in non-backward compatible ways (see Table 1-4). Other methods and fields were deprecated in OpenJPA 0.97, making it likely that they will be removed in a future release of OpenJPA (see Table 1-5). Therefore, their use cannot be relied on.

Table 1-4 Removed methods

pre-1.0 method signature	Method signature for 1.0 and greater
`org.apache.openjpa.persistence.OpenJPAEntityManager`	
public int getConnectionRetainMode();	public ConnectionRetainMode getConnectionRetainMode();
public int getRestoreState();	public RestoreStateType getRestoreState();
public int getDetachState();	public DetachStateType getDetachState();
public intt getAutoClear();	public AutoClearType getAutoClear();
public int getAutoDetach();	public EnumSet<AutoDetachType> getAutoDetach();
`org.apache.openjpa.persistence.OpenJPAQuery`	
public int getOperation();	public QueryOperationType getOperation();

Table 1-4 Removed methods

pre-1.0 method signature	Method signature for 1.0 and greater
`org.apache.openjpa.persistence.jdbc.JDBCFetchPlan`	
public int getEagerFetchMode();	public FetchMode getEagerFetchMode();
public int getSubclassFetchMode();	public FetchMode getSubclassFetchMode();
public int getResultSetType();	public ResultSetType getResultSetType();
public int getFetchDirection();	public FetchDirection getFetchDirection();
public int getJoinSyntax();	public JoinSyntax getJoinSyntax();
`org.apache.openjpa.persistence.jdbc.EagerFetchMode`	
EagerFetchType value() default EagerFetchType.NONE;	FetchMode value() default FetchMode.NONE;
`org.apache.openjpa.persistence.jdbc.SubclassFetchMode`	
EagerFetchType value() default EagerFetchType.NONE;	FetchMode value() default FetchMode.NONE;

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
`org.apache.openjpa.persistence`	
OpenJPAPersistence.EntityManager	JPAFacadeHelper
OpenJPAPersistence.EntityManagerFactory	JPAFacadeHelper
OpenJPAPersistence.toEntityManagerFactory (BrokerFactory)	JPAFacadeHelper
OpenJPAPersistence.toBrokerFactory(EntityManager Factory)	JPAFacadeHelper
OpenJPAPersistence.toEntityManager(Broker)	JPAFacadeHelper
OpenJPAPersistence.toBroker(EntityManager)	JPAFacadeHelper
OpenJPAPersistence.getMetaData(Object)	JPAFacadeHelper

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
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
OpenJPAPersistence.fromOpenJPAObjectIdClass(Class)	JPAFacadeHelper
FetchPlan.getQueryResultCache()	FetchPlan.getQueryResultCacheEnabled()
FetchPlan.setQueryResultCache(boolean cache)	FetchPlan.setQueryResultCache()
FetchPlan.getDelegate()	FetchPlanImpl.getDelegate()
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)

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
OpenJPAEntityManagerFactory.addTransactionListener(Object)	OpenJPAEntityManagerFactorySPI.addTransactionListener(Object)
OpenJPAEntityManagerFactory.removeTransactionListener(Object)	OpenJPAEntityManagerFactorySPI.removeTransactionListener(Object)
QueryResultCache.getDelegate()	QueryResultCacheImpl.getDelegate()
Extent.getDelegate()	ExtentImpl.getDelegate()
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)	QueryImpl.AddFilterListener(FilterListener)
OpenJPAQuery.removeFilterListener(FilterListener)	QueryImpl.removeFilterListener(FilterListener)
OpenJPAQuery.addAggregateListener(AggregateListener)	QueryImpl.addAggregateListener(AggregateListener)
OpenJPAQuery.removeAggregateListener(AggregateListener)	QueryImpl.emoveAggregateListener(AggregateListener)
StoreCache.getDelegate()	StoreCacheImpl.getDelegate()
Generator.getDelegate()	GeneratorImpl.getDelegate()
OpenJPAEntityManager.CONN_RETAIN_DEMAND	ConnectionRetainMode enum
OpenJPAEntityManager.CONN_RETAIN_TRANS	ConnectionRetainMode enum
OpenJPAEntityManager.CONN_RETAIN_ALWAYS	ConnectionRetainMode enum

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
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
OpenJPAEntityManager.CALLBACK_ROLLBACK	CallBackMode enum
OpenJPAEntityManager.getConfiguration()	OpenJPAEntityManagerSPI.getConfiguration()
OpenJPAEntityManager.setRestoreState(int)	OpenJPAEntityManager.setRestoreState(RestoreStateType)
OpenJPAEntityManager.setDetachState(int)	OpenJPAEntityManager.setDetachState(DetachStateType)

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
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()

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
OpenJPAEntityManager.isActive()	EntityTransaction.isActive()
OpenJPAEntityManager.commitAndResume()	OpenJPAEntityTransaction.commitAndResume
OpenJPAEntityManager.rollbackAndResume()	OpenJPAEntityTransaction.rollbackAndResume
OpenJPAEntityManager.setRollbackOnly()	EntityTransaction.setRollbackOnly()
OpenJPAEntityManager.setRollbackOnly(Throwable)	OpenJPAEntityTransaction.setRollbackOnly()
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()

Table 1-5 Deprecated methods and fields

Deprecated	Use instead
JDBCFetchPlan.setLRSSize(int)	JDBCFetchPlan.setLRSSizeAlgorithm(LRSSizeAlgorithm)
JDBCFetchPlan.setJoinSyntax(int)	JDBCFetchPlan.setJoinSyntax(setJoinSyntax)

Standards Support

This release of WebLogic Server supports the following standards.

Table 1 Java Standards Support

Standard	Version
Java EE	5.0
JDKs	6.0, 5.0 (aka 1.5, 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)	2.1, 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)	4.0, 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

Table 1 Java Standards Support

Java EE JSP	2.1, 2.0, 1.2, and 1.1
RMI/IIOP	1.0
JMX	1.2, 1.0
JavaMail	1.2
JAAS	1.0 Full
Java EE CA	1.5, 1.0
Java EE JSF	1.2, 1.1
Java EE JSTL	1.2, 1.1
JCE	1.4
Java RMI	1.0
JAX-B	2.1, 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
JSR 77: Java EE Management	1.1

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)	2.1

Table 2 Web Services Standards Support

Standard	Version
SOAP	1.1, 1.2
WSDL	1.1
JAX-RPC	1.1
SOAP Attachments for Java (SAAJ)	1.3, 1.2
WS-Security	1.1, 1.0
WS-Policy	1.2, 1.5
WS-SecurityPolicy	1.2
WS-PolicyAttachment	1.0
WS-Addressing	1.0, 2004/08 member submission
WS-ReliableMessaging	1.1, 1.0
WS-Trust	1.3
WS-SecureConversation	1.3
UDDI	2.0
JAX-R	1.0
JAX-B	2.1, 2.0
SAML	2.0
SAML Token Profile	1.1

Table 3 Other Standards

Standard	Version
SSL	v3
X.509	v3

Table 3 Other Standards

Standard	Version
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 (RBAC) Profile of XACML	2.0
SAML	1.1, 2.0
Data Direct database drivers	3.7