Oracle® Fusion Middleware

What's New in Oracle WebLogic Server

12*c* Release 1 (12.1.1)

E24494-07

February 2014

Welcome to Oracle WebLogic Server. This is most significant release of WebLogic Server to date, as it is the first release of WebLogic Server that is compliant with Java Enterprise Edition 6 (Java EE 6).

The following sections describe new and changed functionality in this WebLogic Server release:

- Section 1, "JDK 7 Certification"
- Section 2, "Java EE 6 Support"
- Section 3, "Administration Console"
- Section 4, "Core Server"
- Section 5, "Enterprise Java Beans (EJBs)"
- Section 6, "JDBC Data Sources"
- Section 7, "Node Manager"
- Section 8, "Resource Adapters"
- Section 9, "Security"
- Section 10, "Stand-alone Clients"
- Section 11, "Web Services"
- Section 12, "Web Server Plug-ins"
- Section 13, "Standards Support"
- Section 14, "Supported Configurations"
- Section 15, "WebLogic Server Compatibility"
- Section 16, "Documentation Accessibility"

1 JDK 7 Certification

On March 15, 2012, Oracle made available an updated Oracle WebLogic Server 12.1.1 distribution. This distribution includes patches that enable Java SE Development Kit (JDK) 7 certification and provide other product optimizations. Oracle recommends use of WebLogic Server 12.1.1 with these patches applied, on both JDK 6 and JDK 7. You can obtain these patches in either of the following ways:

- The preferred approach is to use the updated WebLogic Server 12.1.1 distribution. If you obtain the updated product distribution for WebLogic Server 12.1.1 on or after March 15, 2012, the patches are included. After you install WebLogic Server,

the JDK 7 certification patches are automatically applied and put in effect when using standard WebLogic Server start scripts.

For more information about obtaining the Oracle WebLogic Server distribution for your platform, see "Product Distribution" in the *Oracle WebLogic Server Installation Guide*.

- If you obtained a product distribution for WebLogic Server 12.1.1 prior to March 15, 2012, and you do not wish to re-install with the updated WebLogic Server distribution, you can obtain the patches from My Oracle Support at `http://support.oracle.com/` and apply them to your existing installation. For information about how to obtain and apply these patches, see Section 1.1, "About the Patches Available for WebLogic Server 12.1.1".

If you are unsure whether your WebLogic Server 12.1.1 distribution contains the patches, launch Smart Update to view the patches applied your installation, or inspect the contents of your *MW_HOME*`/patch_wls1211/patch_jars` directory, where *MW_HOME* represents the Middleware home directory.

> **Note:** Before you install the updated WebLogic Server 12.1.1 distribution or the JDK 7 certification patches, Oracle recommends first installing JDK 7. For information, see "Using WebLogic Server with JDK 7" in the *Oracle WebLogic Server Installation Guide*.

1.1 About the Patches Available for WebLogic Server 12.1.1

Table 1 lists the categories of patches that are included in the updated distribution of WebLogic Server 12.1.1. This table also provides links to the sections in the *Oracle WebLogic Server Release Notes* where the specific issues corrected by those patches are described.

Table 1 Issues Corrected by Patches Available for WebLogic Server 12.1.1

For information about issues corrected by the following . . .	. . . see the following section in the *Release Notes*
JDK 7 certification patches	"JDK 7 Certification"
Contexts and Dependency Injection patches	"Contexts and Dependency Injection"
Deployment performance optimization patches	"Deployment Performance"
Developer experience patch (applies to the developer-only distribution)	"Developer Experience"

If you use an updated WebLogic Server 12.1.1 distribution that is available on March 15, 2012:

- The patches are placed in the *MW_HOME*`/patch_wls1211/patch_jars` directory on your computer, where *MW_HOME* represents your Middleware home directory.
- The patches contained in this patch directory will be included in the system classpath using standard WebLogic Server start scripts so that those patches automatically go into effect when WebLogic Server is started.

- If you do not use standard WebLogic Server start scripts, you must ensure that your start scripts will pick up patches that are stored in the preceding patch directory location.

If you have an installation of WebLogic Server 12.1.1 obtained prior to March 15, 2012, you may prefer to download and apply the patches yourself rather than re-installing WebLogic Server 12.1.1. If you choose to apply the patches yourself, you can refer to the bug descriptions in the sections listed in Table 1 for links to My Oracle Support where the patches can be obtained.

If you choose to download and apply the patches yourself, note the following:

- To apply these patches to an installation created with a package installer (see "Types of Installers" in *Oracle WebLogic Server Installation Guide*), follow the instructions provided in *Oracle Smart Update Installing Patches and Maintenance Packs*.
- WebLogic Server Smart Update is not included in the WebLogic Server developer-only ZIP file distribution. To apply these patches to an installation created with this ZIP file distribution, you must:

  1. Download the patch ZIP files from My Oracle Support using the Oracle patch numbers provided in the *Oracle WebLogic Server Release Notes* sections referenced in Table 1.
  2. From each patch ZIP file you download, extract the patch jar file and copy it into a well-known patch directory, such as *MW_HOME*`/patch_wls1211/patch_jars`.

     Note that the particular patch jar file you need to copy is typically embedded within a high-level jar file in the patch ZIP file.

     For example, a downloaded patch ZIP named `p13019800_12110_Generic.zip` might contain the jar file `8PE3.jar`. Unjarring `8PE3.jar` yields, in turn, in inner jar file named something like `BUG`*nnnn*`_1211.jar`. (This inner jar file is typically located within a `dirs` sub-folder.) This inner jar file—that is, `BUG`*nnnn*`_1211.jar`—is the patch jar that must be copied into the well-known patch directory.

  The patches contained in this well-known patch directory are included in the system classpath using standard WebLogic Server start scripts so that those patches automatically go into effect when WebLogic Server is started.

  If you do not use standard WebLogic Server start scripts, you must ensure that your start scripts pick up patches that are stored in the well-known location where you have placed your patches.

2 Java EE 6 Support

This release of WebLogic Server supports Java EE 6. This section describes the main features of Java EE 6 that can be leveraged by application developers in a WebLogic Server 12.1.1 environment. It includes the following sections:

- Section 2.1, "Java EE 6 Platform Highlights"
- Section 2.2, "Enterprise JavaBeans (EJB) 3.1"
- Section 2.3, "Java Servlet 3.0 Technology"
- Section 2.4, "JavaServer Faces (JSF) 2.x and JavaServer Pages Standard Tag Library (JSTL) 1.2"

- Section 2.5, "Java Persistence API (JPA) 2.0"
- Section 2.6, "Java Transaction API (JTA)"
- Section 2.7, "Java API for RESTful Web Services 1.0"
- Section 2.8, "Managed Beans 1.0"
- Section 2.9, "Contexts and Dependency Injection for the Java EE Platform 1.0"
- Section 2.10, "Dependency Injection for Java 1.0"
- Section 2.11, "Bean Validation"
- Section 2.12, "Java EE Connector Architecture (JCA) 1.6"
- Section 2.13, "Java Authorization Contract for Containers (JACC) 1.3"
- Section 2.14, "Java Authentication Service Provider Interface for Containers (JASPIC) 1.0"
- Section 2.15, "Common Annotations for Java Platform 1.1"
- Section 2.16, "Java Architecture for XML Binding (JAXB) 2.2"
- Section 2.17, "Java API for XML Web Services (JAX-WS) 2.2"
- Section 2.18, "Interceptors 1.1"

2.1 Java EE 6 Platform Highlights

The most important goal of the Java EE 6 platform is to simplify development by providing a common foundation for the various kinds of components in the Java EE platform. Developers benefit from productivity improvements with more annotations and less XML configuration, more Plain Old Java Objects (POJOs), and simplified packaging. The Java EE 6 platform includes the following new features:

- New technologies, including the following:
  - Java API for RESTful Web Services (JAX-RS)
  - Managed Beans
  - Contexts and Dependency Injection for the Java EE Platform (JSR 299)
  - Dependency Injection for Java (JSR 330)
  - Bean Validation (JSR 303)
  - Java Authentication Service Provider Interface for Containers (JASPIC)
- New features for Enterprise JavaBeans (EJB) components
- New features for servlets
- New features for JavaServer Faces components

2.2 Enterprise JavaBeans (EJB) 3.1

An Enterprise JavaBeans (EJB) component, or enterprise bean, is a body of code having fields and methods to implement modules of business logic. You can think of an enterprise bean as a building block that can be used alone or with other enterprise beans to execute business logic on the Java EE server.

Enterprise beans are either session beans or message-driven beans:

- A session bean represents a transient conversation with a client. When the client finishes executing, the session bean and its data are gone.
- A message-driven bean combines features of a session bean and a message listener, allowing a business component to receive messages asynchronously. Commonly, these are Java Message Service (JMS) messages.

In the Java EE 6 platform, new enterprise bean features including:

- The ability to package local enterprise beans in a WAR file
- Singleton session beans, which provide easy access to shared state
- Simplified no-interface client view, which provides session bean access without a separate local business interface
- Asynchronous session bean invocations
- Portable global JNDI name syntax for looking up EJB components
- Automatically created EJB timers and calendar-based EJB timer expressions
- An embeddable API for executing EJB components within a Java SE environment

For more information on these features, see Section 5, "Enterprise Java Beans (EJBs)".

The Interceptors specification, which is part of the EJB 3.1 specification, makes the interceptor facility that is originally defined as part of the EJB 3.0 specification more generally available.

WebLogic Server features that are provided in this release to support EJB 3.1 include:

- EJB 3.1 annotation support
- Administration Console support for EJBs in a WAR.

2.3 Java Servlet 3.0 Technology

Java Servlet technology lets you define HTTP-specific servlet classes. A servlet class extends the capabilities of servers that host applications accessed by way of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers.

In the Java EE 6 platform, new Java servlet technology features include:

- Annotation support
- Asynchronous support
- Ease of configuration
- Enhancements to existing APIs
- Pluggability

2.4 JavaServer Faces (JSF) 2.x and JavaServer Pages Standard Tag Library (JSTL) 1.2

This release of WebLogic Server supports JSF 2.x and JSTL 1.2. JSF 2.x and JSTL 1.2 have been incorporated directly into the server's classpath. Applications deployed to WebLogic Server can seamlessly make use of JSF 2.x and JSTL 1.2 without requiring developers to deploy and reference separate shared libraries.

JSF technology is a user interface framework for building web applications. The main components of JSF technology are:

- A GUI component framework.
- A flexible model for rendering components in different kinds of HTML or different markup languages and technologies. A Renderer object generates the markup to render the component and converts the data stored in a model object to types that can be represented in a view.
- A standard RenderKit for generating HTML/4.01 markup.

The following features support the GUI component:

- Input validation
- Event handling
- Data conversion between model objects and components
- Managed model object creation
- Page navigation configuration
- Expression Language (EL)

All of this functionality is available using standard Java APIs and XML-based configuration files.

In the Java EE 6 platform, new JSF features include:

- The ability to use annotations instead of a configuration file to specify managed beans
- Facelets, a display technology that replaces JavaServer Pages (JSP) technology using XHTML files
- Ajax support
- Composite components
- Implicit navigation

JSTL encapsulates core functionality common to many JSP applications. Instead of mixing tags from numerous vendors in your JSP applications, you use a single, standard set of tags. This standardization allows you to deploy your applications on any JSP container that supports JSTL and makes it more likely that the implementation of the tags is optimized.

JSTL has an iterator and conditional tags for handling flow control, tags for manipulating XML documents, internationalization tags, tags for accessing databases using SQL, and commonly used functions.

2.5 Java Persistence API (JPA) 2.0

JPA is a Java standards-based solution for persistence. Persistence uses an object/relational mapping approach to bridge the gap between an object-oriented model and a relational database. The Java Persistence API can also be used in Java SE applications, outside of the Java EE environment. Java Persistence consists of the following areas:

- The Java Persistence API
- The query language

- Object/relational mapping metadata

2.6 Java Transaction API (JTA)

JPA provides a standard interface for demarcating transactions. The Java EE architecture provides a default auto commit to handle transaction commits and rollbacks. An auto commit means that any other applications that are viewing data will see the updated data after each database read or write operation. However, if your application performs two separate database access operations that depend on each other, you will want to use the JTA API to demarcate where the entire transaction, including both operations, begins, rolls back, and commits.

2.7 Java API for RESTful Web Services 1.0

This release of WebLogic Server provides full support for the Java API for RESTful Web Services (JAX-RS).

JAX-RS for the development of web services is built according to the Representational State Transfer (REST) architectural style. A JAX-RS application is a web application that consists of classes that are packaged as a servlet in a WAR file along with required libraries.

The JAX-RS API is new to the Java EE 6 platform.

2.8 Managed Beans 1.0

Managed Beans, lightweight container-managed objects (POJOs) with minimal requirements, support a small set of basic services, such as resource injection, lifecycle callbacks, and interceptors. Managed Beans represent a generalization of the managed beans specified by JavaServer Faces technology and can be used anywhere in a Java EE application, not just in web modules.

The Managed Beans specification is part of the Java EE 6 platform specification (JSR 316).

Managed Beans are new to the Java EE 6 platform.

2.9 Contexts and Dependency Injection for the Java EE Platform 1.0

Contexts and Dependency Injection (CDI) for the Java EE platform defines a set of services for using injection to specify dependencies in an application. CDI provides contextual life cycle management of beans, type-safe injection points, a loosely coupled event framework, loosely coupled interceptors and decorators, alternative implementations of beans, bean navigation through the Unified Expression Language (EL), and a service provider interface (SPI) that enables CDI extensions to support third-party frameworks or future Java EE components. CDI is integrated with the major component technologies in Java EE.

CDI is new to the Java EE 6 platform

2.10 Dependency Injection for Java 1.0

Dependency Injection (DI) for Java defines a standard set of annotations (and one interface) for use on injectable classes.

In the Java EE platform, CDI provides support for DI. Specifically, you can use DI injection points only in a CDI-enabled application.

DI for Java is new to the Java EE 6 platform.

2.11 Bean Validation

The Bean Validation specification (JSR 316) defines a metadata model and API for validating data in JavaBeans components. It is supported on both the server and Java EE 6 client; therefore, instead of distributing validation of data over several layers, such as the browser and the server side, you can define the validation constraints in one place and share them across the different layers. Further, bean validation is not only for validating beans. In fact, it can also be used to validate any Java object.

Bean Validation is new to the Java EE 6 platform.

2.12 Java EE Connector Architecture (JCA) 1.6

JCA is used by tools vendors and system integrators to create resource adapters that support access to enterprise information systems that can be plugged in to any Java EE product. A resource adapter is a software component that allows Java EE application components to access and interact with the underlying resource manager of the EIS. Because a resource adapter is specific to its resource manager, a different resource adapter typically exists for each type of database or enterprise information system.

JCA also provides a performance-oriented, secure, scalable, and message-based transactional integration of Java EE based web services with existing EISs that can be either synchronous or asynchronous. Existing applications and EISs integrated through the Java EE Connector architecture into the Java EE platform can be exposed as XML-based web services by using JAX-WS and Java EE component models. Thus JAX-WS and the Java EE Connector architecture are complementary technologies for enterprise application integration (EAI) and end-to-end business integration.

2.13 Java Authorization Contract for Containers (JACC) 1.3

The JACC specification defines a contract between a Java EE application server and an authorization policy provider. All Java EE containers support this contract.

The JACC specification defines `java.security.Permission` classes that satisfy the Java EE authorization model. The specification defines the binding of container access decisions to operations on instances of these permission classes. It defines the semantics of policy providers that use the new permission classes to address the authorization requirements of the Java EE platform, including the definition and use of roles.

2.14 Java Authentication Service Provider Interface for Containers (JASPIC) 1.0

This release of WebLogic Server provides support for the JASPIC specification.

The JASPIC specification defines a service provider interface (SPI) by which authentication providers that implement message authentication mechanisms may be integrated in client or server message-processing containers or runtimes. Authentication providers integrated through this interface operate on network messages provided to them by their calling container. The authentication providers transform outgoing messages so that the source of the message can be authenticated by the receiving container, and the recipient of the message can be authenticated by the message sender. Authentication providers authenticate incoming messages and

return to their calling container the identity established as a result of the message authentication.

JASPIC is new to the Java EE 6 platform.

For more information, see Section 9.1, "Java Authentication Service Provider Interface for Containers (JASPIC) Support".

2.15 Common Annotations for Java Platform 1.1

Java EE 6 supports naming of applications and modules. In WebLogic Server 12.1.1, application containers have been modified to include support for the Java EE 6 `<application_name>` element in `application.xml`. For more detailed information, see "Understanding Default Deployment Names" in *Deploying Applications to Oracle WebLogic Server*.

2.16 Java Architecture for XML Binding (JAXB) 2.2

JAXB provides a convenient way to bind an XML schema to a representation in Java language programs. JAXB can be used independently or in combination with JAX-WS, where it provides a standard data binding for web service messages. All Java EE application client containers, web containers, and EJB containers support the JAXB API.

2.17 Java API for XML Web Services (JAX-WS) 2.2

The JAX-WS specification provides support for web services that use the JAXB API for binding XML data to Java objects. The JAX-WS specification defines client APIs for accessing web services as well as techniques for implementing web service endpoints. The Implementing Enterprise Web Services specification describes the deployment of JAX-WS-based services and clients. The EJB and Java Servlet specifications also describe aspects of such deployment. It must be possible to deploy JAX-WS-based applications using any of these deployment models.

The JAX-WS specification describes the support for message handlers that can process message requests and responses. In general, these message handlers execute in the same container and with the same privileges and execution context as the JAX-WS client or endpoint component with which they are associated. These message handlers have access to the same JNDI java:comp/env namespace as their associated component. Custom serializers and deserializers, if supported, are treated in the same way as message handlers.

2.18 Interceptors 1.1

This release of WebLogic Server provides generic support for interceptors. Support for a generic interceptor layer that can be used by the EJB container has been added.

Interceptors are used in conjunction with Java EE managed classes to allow developers to invoke interceptor methods in conjunction with method invocations or lifecycle events on an associated target class. Common uses of interceptors are logging, auditing, or profiling.

Interceptors can be defined within a target class as an interceptor method, or in an associated class called an interceptor class. Interceptor classes contain methods that are invoked in conjunction with the methods or lifecycle events of the target class.

Interceptor classes and methods are defined using metadata annotations, or in the deployment descriptor of the application containing the interceptors and target classes.

3 Administration Console

This section describes the new Administration Console features in this release of WebLogic Server.

3.1 Support for EJB Modules in a WAR File

Support for managing EJB modules that are directly inside of Web application archive (WAR) files via the Administration Console has been added. For more information, see Section 5.1, "Packaging and Deploying EJBs Directly in WAR Files".

3.2 Console Changes to Support Java EE 6

Various Administration Console changes have been made to support the implementation of Java EE 6, including changes to:

- Deployment
- Application container
- SCA container
- Split source
- Application and module naming

4 Core Server

WebLogic Server 12.1.1 includes a new Maven plug-in for WebLogic Server (`wls-maven-plugin`) with enhanced functionality to install, start and stop servers, create domains, execute WLST scripts, and compile and deploy applications from within your Maven environment. For more information, see "Using the WebLogic Development Maven Plug-In" in *Developing Applications for Oracle WebLogic Server*.

5 Enterprise Java Beans (EJBs)

This section describes the new EJB 3.1 features in this release of WebLogic Server:

- Section 5.1, "Packaging and Deploying EJBs Directly in WAR Files"
- Section 5.2, "Singleton Session Beans"
- Section 5.3, "EJB Timer Enhancements"
- Section 5.4, "Portable Global JNDI Names"
- Section 5.5, "Asynchronous Session Bean Invocations"
- Section 5.6, "Simplified No Interface Client View"
- Section 5.7, "Embeddable EJB API"
- Section 5.8, "JPA 2.0 Support Using the Default TopLink Persistence Provider"
- Section 5.9, "Applications That Use Kodo as the Persistence Provider"

5.1 Packaging and Deploying EJBs Directly in WAR Files

EJB 3.1 provides the ability to place EJB modules directly inside of Web application archive (WAR) files, removing the need to produce archives to store the Web and EJB components and combine them together in an enterprise application archive (EAR) file. For more information see, "Packaging an EJB In a WAR" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

5.2 Singleton Session Beans

Singleton session beans provide a formal programming construct that guarantees a session bean will be instantiated once per application in a particular Java Virtual Machine (JVM), and that it will exist for the life cycle of the application. With singletons, you can easily share state between multiple instances of an enterprise bean component or between multiple enterprise bean components in the application. For more information see, "Singleton Session Beans" in *Understanding Oracle WebLogic Server.*

5.3 EJB Timer Enhancements

EJB 3.1 provides the following TimerService features:

- **Calendar-based EJB Timers** – EJB 3.1 supports calendar-based EJB TimerService expressions. The scheduling functionality takes the form of CRON-styled schedule definitions that can be placed on EJB methods, in order to have the methods be automatically invoked according to the defined schedule. For more information see, "Calendar-based Timers" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.
- **Automatically-created EJB Timers** – EJB 3.1 supports the automatic creation of a timer based on metadata in the bean class or deployment descriptor, which allows the bean developer to schedule a timer without relying on a bean invocation to programmatically invoke one of the Timer Service timer creation methods. Automatically created timers are created by the container as a result of application deployment. For more information see, "Automatically-created EJB Timers" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

5.4 Portable Global JNDI Names

The Portable Global JNDI naming option in EJB 3.1 provides a number of common, well-known namespaces in which EJB components can be registered and looked up from using the pattern `java:global[/<app-name>]/<module-name>/<bean-name>`. This standardizes how and where EJB components are registered in JNDI, and how they can be looked up and used by applications. For more information see, "Programming Access to EJB Clients" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server.*

5.5 Asynchronous Session Bean Invocations

An EJB 3.1 session bean can expose methods with asynchronous client invocation semantics. Using the `@Asynchronous` annotation in an EJB class or specific method will direct the EJB container to return control immediately to the client when the method is invoked. The method may return a future object to allow the client to check on the status of the method invocation, and retrieve result values that are asynchronously

produced. For more information see, "Programming Asynchronous Business Methods" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

5.6 Simplified No Interface Client View

The No-interface local client view type simplifies EJB development by providing local session bean access without requiring a separate local business interface, allowing components to have EJB bean class instances directly injected. For more information see, "Accessing EJBs Using the No-Interface Client View" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

5.7 Embeddable EJB API

EJB 3.1 supports an embeddable API for executing EJB components within a Java SE environment. Unlike traditional Java EE server-based execution, embeddable usage allows client code and its corresponding enterprise beans to run within the same virtual machine and class loader. This provides better support for testing, offline processing (for example, batch jobs), and the use of the EJB programming model in desktop applications. For more information, see "Using an Embedded EJB Container in Oracle WebLogic Server" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

5.8 JPA 2.0 Support Using the Default TopLink Persistence Provider

Oracle TopLink, a JPA 2.0 persistence provider, is now the default JPA provider, replacing Kodo, which was the default provider in previous releases. Any application that does not specify a JPA provider in `persistence.xml` will now use TopLink by default. Applications can continue to use Kodo (a JPA 1.0 provider) by explicitly specifying Kodo/OpenJPA as their persistence provider in `persistence.xml`. In addition, a Weblogic Server domain can be configured to use Kodo by default, if desired.

For more information, see "Configuring the Persistence Provider in WebLogic Server" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server.*

5.9 Applications That Use Kodo as the Persistence Provider

WebLogic Server runs with the JPA 2.0 JAR in the server's classpath. Although JPA 2.0 is upwardly compatible with JPA 1.0, JPA 2.0 introduced some methods to existing JPA interfaces that conflict with existing signatures in OpenJPA interfaces. As a result, applications that continue to use Kodo/JPA as the persistence provider with WebLogic Server 12.1.1 must be recompiled. For more information, see "Updating Applications to Overcome Conflicts" in *Programming WebLogic Enterprise JavaBeans, Version 3.0 for Oracle WebLogic Server*.

6 JDBC Data Sources

This release of WebLogic Server supports data sources per the Java EE 6 specifications.

WebLogic Server 12.1.1 also provides an extended set of WebLogic data source configuration attributes. See "Using DataSource Resource Definitions" in *Programming JDBC for Oracle WebLogic Server*.

7 Node Manager

The default value for `startScriptEnabled` has been changed to `true` as of this release. In previous releases, the default was `false`.

8 Resource Adapters

WebLogic Server now fully supports the Java EE Connector Architecture Specification, Version 1.6 Final Release, as well as resource adapters based on versions 1.0 and 1.5 of the Java EE Connector Architecture. Except where noted, the following sections describe new functionality for version 1.6 resource adapters:

- Section 8.1, "Ease of Development and Configuration"
- Section 8.2, "Generic Work Context"
- Section 8.3, "Security"
- Section 8.4, "Schema Changes"
- Section 8.5, "Contexts and Dependency Injection (CDI) Support"

8.1 Ease of Development and Configuration

WebLogic Server supports several ease-of-development and ease-of-configuration features introduced in the Java EE Connector Architecture version 1.6, including the following:

- Metadata annotations — Developer complexity is reduced by supporting the use of annotations in resource adapter class files. Annotations can specify deployment information, eliminating the need to create the `ra.xml` deployment descriptor file manually.
- Dynamic configuration properties that can be defined on ResourceAdapter, ManagedConnectionFactory, and Admin Object beans.
- The ability to specify, at run time, the level of transaction support a resource adapter can provide.

8.2 Generic Work Context

The Generic Work Context, new in Connector Architecture version 1.6, is a mechanism for resource adapters to propagate contextual information, in addition to transaction inflow, from Enterprise Information Systems (EISes) during message delivery and Work submission. The Generic Work Context has the following impact on developing, configuring, and running resource adapters in the WebLogic Server environment:

- Transaction support—The new TransactionContext replaces the transaction inflow contract.
- Security— The new SecurityContext defines the security context for work submitted by the resource adapter on behalf of the EIS to WebLogic Server.
- Management of long-running work—A resource adapter can define constants to represent long-running Work instances in WebLogic Server, including the threads in which they execute, to facilitate better thread monitoring and management.

8.3 Security

The WebLogic Server adds supplemental support for the security context in the Administration Console by providing a means to create inbound EIS-to-WebLogic principal mappings, which map EIS principals, such as users or groups defined in the EIS security domain, to corresponding principals in the WebLogic domain. For more information, see "Security" in *Programming Resource Adapters for Oracle WebLogic Server*.

8.4 Schema Changes

As a result of the features added to Connector Architecture 1.6 to simplify development and configuration, the schemas for the `ra.xml` and `weblogic-ra.xml` deployment descriptor files have changed. For more information, see "weblogic-ra.xml Schema" in *Programming Resource Adapters for Oracle WebLogic Server*.

8.5 Contexts and Dependency Injection (CDI) Support

The WebLogic Server Connector container adds support for JSR 299: Contexts and Dependency Injection for the Java EE Platform (CDI) in embedded and global resource adapters. CDI defines a set of services for using injection to specify dependencies in an application. WebLogic Server supports the following CDI features in 1.6 resource adapters:

- Bean discovery — The resource adapter RAR, all JARs packaged inside the RAR, and every bean archive referenced by the RAR is searched, regardless of whether a `beans.xml` bean archive descriptor file exists in those JARs.
- Bean Manager — The ExtendedBootstrapContext class now includes the `getBeanManager` method. This method can be used to return a resource adapter's Bean Manager, which provides the means to retrieve bean instances from that adapter.
- Bean injection — WebLogic Server includes a number of built-in beans in its Connector Architecture 1.6 implementation that permit injection points for references to a variety of artifacts for the current resource adapter bean, such as the BootstrapContext, WorkManager, XATerminator, and JTA TransactionSynchronizationRegistry objects.
- Support for the `javax.annotation.PostConstruct` and `javax.annotation.PreDestroy` annotation types on ResourceAdapter, ManagedConnectionFactory, ActivationSpec, and Admin Object beans.

Note that the use of the `javax.annotation.Resource` annotation type used for declaring a reference to a resource adapter's managed beans is not supported. For more information, see "Using Contexts and Dependency Injection in Resource Adapters" in *Programming Resource Adapters for Oracle WebLogic Server*.

9 Security

This section describes the following security changes in WebLogic Server 12.1.1:

- Section 9.1, "Java Authentication Service Provider Interface for Containers (JASPIC) Support"
- Section 9.2, "Java Authorization Contract for Containers (JACC) 1.4 Support"
- Section 9.3, "RSA JSSE Provider"
- Section 9.4, "SSL Implementation"

- Section 9.5, "Enhancements to Support for Single Sign-On with Microsoft Clients"

9.1 Java Authentication Service Provider Interface for Containers (JASPIC) Support

This release of WebLogic Server supports JSR 196: Java Authentication Service Provider Interface for Containers (JASPIC) Version 1.0. The JASPIC specification defines a service provider interface (SPI) by which authentication providers that implement message authentication mechanisms can be integrated in server Web application message processing containers or runtimes.

The message processing run time uses this SPI at the identified message processing points to delegate the corresponding message security processing to the authentication providers.

The JASPIC authentication provider assumes responsibility for authenticating the user credentials and returning a Subject. WebLogic Server then treats this Subject as it would all others.

Authentication providers integrated through this interface operate on network messages provided to them by the WLS container. They authenticate incoming Web application messages and return to their calling container (WLS) the identity (the expected Subject) established as a result of the message authentication.

9.2 Java Authorization Contract for Containers (JACC) 1.4 Support

In this release, WebLogic Server adds support for JACC 1.4. For information about using JACC, see "Using the Java Authorization Contract for Containers" in *Programming Security for Oracle WebLogic Server*.

9.3 RSA JSSE Provider

WebLogic Server 12.1.1 includes support for the RSA JSSE provider. RSA JSSE is a third-party JSSE provider that can be statically registered in the JVM if you wish to use it. For information about how to install and configure the RSA JSSE provider, see "Using the RSA JSSE Provider in WebLogic Server" in *Securing Oracle WebLogic Server.*

9.4 SSL Implementation

Certicom has been removed from WebLogic Server 12.1.1 and is no longer supported.

JSSE is the only SSL implementation that is supported in WebLogic Server 12.1.1. The following configuration changes have been made to be consistent with this support:

- The default for `JSSEEnabled` has been changed to `true`. Oracle recommends that you keep this value set to `true`.
- If `JSSEEnabled` is set to `false`, it will be ignored. That is, the MBean value will not be changed either in memory or the persisted `config.xml` file. WebLogic Server will continue to use JSSE, but will issue a warning.

9.5 Enhancements to Support for Single Sign-On with Microsoft Clients

The release of WebLogic Server adds the following enhancements to its support for single sign-on (SSO) with Microsoft clients using Windows Integrated Authentication

based on the Simple and Protected Negotiate (SPNEGO) mechanism and the Kerberos protocol:

- Use of the AES-128, AES-256, and RC4 encryption algorithms for encrypting the user accounts that are mapped to Kerberos services on the WebLogic Server host
- Support for Java SE clients

10 Stand-alone Clients

The WebLogic Thin T3 Client jar (`wlthint3client.jar`) supports the GlassFish application server version 3.1 and higher. For more information, see "Developing a WebLogic Thin T3 Client" in *Oracle Fusion Middleware Programming Stand-alone Clients for Oracle WebLogic Server*.

11 Web Services

This section describes new and changed WebLogic Web Services features in this release of WebLogic Server, including:

- Section 11.1, "WebLogic Web Service Compliance With Java EE 6"
- Section 11.2, "Enhanced Support of Web Services in EJB 3.1"
- Section 11.3, "Improved Support for Jersey JAX-RS RI Version 1.9"
- Section 11.4, "New RESTful Web Service (JAX-RS) Sample"
- Section 11.5, "Support for EclipseLink MOXy (JAXB)"
- Section 11.6, "UDDI v2.0 Registry and UDDIExplorer Removed in this Release"
- Section 11.7, "WebLogic Web Services 8.1 Application Environment Removed"

11.1 WebLogic Web Service Compliance With Java EE 6

WebLogic Web service features and standards are compliant with Java EE 6. For the current list of standards supported for WebLogic Web services, see "Features and Standards Supported by WebLogic Web Services" in *Introducing WebLogic Web Services for Oracle WebLogic Server*.

11.2 Enhanced Support of Web Services in EJB 3.1

WebLogic Web services include enhancements to support EJB 3.1. Specifically, WebLogic Web services can be packaged as follows:

- EJB in a WAR file
- Singleton EJB

11.3 Improved Support for Jersey JAX-RS RI Version 1.9

WebLogic Server supports Jersey Java API for RESTful Web Services (JAX-RS) Reference Implementation (RI) Version 1.9, which is a production quality implementation of the JSR-311 JAX-RS specification, defined at: `http://jcp.org/en/jsr/summary?id=311`. In this release, the JAX-RS application environment is no longer dependent on WebLogic Server shared libraries and provides WebLogic Server Runtime MBeans for post-deployment manageability. As required, you can use a more recent version of the Jersey JAX-RS RI.

For information about developing WebLogic Web services that conform to the Representational State Transfer (REST) architectural style using JAX-RS, see *Developing RESTful Web Services for Oracle WebLogic Server*.

11.4 New RESTful Web Service (JAX-RS) Sample

A new sample, RESTful Web Services (JAX-RS), has been added to the WebLogic Server Sample Applications and Code Examples. For information about accessing the JAX-RS sample, see "Sample Application and Code Examples" in *Understanding Oracle WebLogic Server*.

11.5 Support for EclipseLink MOXy (JAXB)

In this release, EclipseLink MOXy (JAXB) is the default data binding and JAXB provider. For more information about EclipseLink MOXy (JAXB), see:

- "Using JAXB Data Binding" in *Developing JAX-WS Web Services for Oracle WebLogic Server*
- *EclipseLink MOXy (JAXB) User's Guide* at http://wiki.eclipse.org/EclipseLink/UserGuide/MOXy

Alternatively, you can enable the Glassfish RI JAXB as the data binding and JAXB provider at the server or application level. For more information, see "Using the Glassfish RI JAXB Data Binding and JAXB Providers" in *Getting Started With JAX-WS Web Services for Oracle WebLogic Server*.

11.6 UDDI v2.0 Registry and UDDIExplorer Removed in this Release

The UDDI v2.0 registry and UDDIExplorer applications have been removed in this release. Customers are encouraged to consider upgrading to Oracle Enterprise Repository or Oracle Service Registry, which provide SOA visibility and governance. For more information, see http://www.oracle.com/us/technologies/soa/soa-governance/index.html. From the list of products at the bottom of the page, select **Oracle Service Registry**. On the lower right, expand the **Brochures and Data Sheets** section to access the *Oracle Enterprise Repository and Oracle Service Registry for SOA Governance* data sheet.

11.7 WebLogic Web Services 8.1 Application Environment Removed

The WebLogic Web Services 8.1 application environment has been removed in this release. As a result, WebLogic Web service applications built on WebLogic Server 8.1 are no longer supported in this release. You must upgrade you 8.1 Web services to a more recent version.

12 Web Server Plug-ins

The WebLogic Web Server plug-ins are no longer included with WebLogic Server. The plug-ins are now available as a separate download. You can download the plug-ins from http://www.oracle.com/technetwork/middleware/weblogic/downloads/wls-main-097127.html. For more information, please see the My Oracle Support Doc ID 1111903.1 within Oracle Support.

13 Standards Support

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

13.1 Java Standards

Table 2 lists currently supported Java standards.

Table 2 Java Standards Support

Standard	Version
Contexts and Dependency Injection for Java EE	1.0
Dependency Injection for Java EE	1.0
Expression Language (EL)	2.2, 2.1, 2.0 Only JSP 2.0 and greater supports Expression Language 2.x.
JAAS	1.0 Full
JASPIC	1.0
Java API for XML-Based Web Services (JAX-WS)	2.2, 2.1, 2.0
Java API for RESTful Web Services (JAX-RS)	1.1
Java Authorization Contract for Containers (JACC)	1.4
Java EE	6.0
Java EE Application Deployment	1.2
Java EE Bean Validation	1.1
Jave EE Common Annotations	1.0
Java EE Connector Architecture	1.6
Java EE EJB	3.1
Java EE Enterprise Web Services	1.3, 1.2, 1.1
Jave EE Interceptors	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	2.1, 2.0, 1.2, 1.1
Java EE JSP	2.2, 2.1, 2.0, 1.2, and 1.1 JSP 1.2. and 1.1 include Expression Language (EL), but do not support EL 2.x or greater.
Java EE Managed Beans	1.0
Java EE Servlet	3.0, 2.5, 2.4, 2.3, and 2.2
Java RMI	1.0
JavaMail	1.4
JAX-B	2.2, 2.1, 2.0
JAX-P	1.3, 1.2, 1.1

Table 2 (Cont.) Java Standards Support

Standard	Version
JAX-R	1.0
JAX-RPC	1.1
JCE	1.4
JDKs	7.0 (1.7), 6.0 (1.6)
JMX	1.2, 1.0
JPA	2.0, 1.0
JSR 77: Java EE Management	1.1
JSTL	1.2
Managed Beans	1.0
OTS/JTA	OTS 1.2 and JTA 1.1
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

13.2 Web Services Standards

For the current list of standards supported for WebLogic Web services, see "Features and Standards Supported by WebLogic Web Services" in *Introducing WebLogic Web Services for Oracle WebLogic Server*.

13.3 Other Standards

Table 3 lists other standards that are supported in this release of WebLogic Server.

Table 3 Other Standards

Standard	Version
SSL	v3
X.509	v3
LDAP	v3
TLS	v1.1, v1.2
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/technetwork/middleware/ias/downloads/fusion-certification-100350.html.

14 Supported Configurations

For the most current information on supported configurations, refer to the Oracle Fusion Middleware Supported Configurations Central Hub.

15 WebLogic Server Compatibility

For the most current information on compatibility between current version of WebLogic Server and previous releases, see "WebLogic Server Compatibility" in *Understanding Oracle WebLogic Server*.

16 Documentation Accessibility

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

Access to Oracle Support

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

Oracle Fusion Middleware What's New in Oracle WebLogic Server, 12*c* Release 1 (12.1.1)
E24494-07