BEA WebLogic Server® and WebLogic Express™

Release Notes

Version 9.0
Document Revised: April 20, 2006

Contents

What’s New in WebLogic Server 9.0

Welcome to BEA WebLogic Server 9.0, the most significant WebLogic Server® release to date. Fully compliant with J2EE 1.4, WebLogic Server 9.0 tackles the biggest challenge facing enterprise networks today: reducing overall cost of management and operations while delivering high reliability, continuous uptime, scalability, and mission-critical integration solutions.

Note: For information on current known problems and problems that are now resolved, see WebLogic Server Known and Resolved Issues.

The following sections describe new and changed functionality in WebLogic Server 9.0:

What's New: The Big Picture

WebLogic Server 9.0 is fully compliant with the J2EE 1.4 Specification. This release implements and extends the latest J2EE standards -- Enterprise Web Services 1.1, JMS 1.1, JMX 1.2, JDBC 3.0, Connector Architecture 1.5, EJB 2.1, and many more -- to deliver unparalleled quality-of-service across the enterprise. The following sections summarize innovations in key areas of functionality. For detailed standards support information, see "Standards Support" on page 63.

Systems Management

WebLogic Server 9.0 focuses on simplifying and streamlining the day-to-day management of production systems with minimal disruption to live production environments. A comprehensive, centralized diagnostics service lets administrators identify and resolve issues in real time, and accommodates the integration of third-party analytic tools. This release of WebLogic Server introduces a more extensible, "portalized" Administration Console as well as a tightly controlled, predictable process for managing changes to a domain's configuration regardless of the configuration tool you use. A new scripting tool, a simplified domain directory structure, and modular deployment capabilities automate and facilitate application configuration and deployment.

See "System Administration and Management" on page 4.

Performance and Availability

Out-of-the-box support for WAN and MAN failover address catastrophic data center failures. Overall server performance improves greatly in WebLogic Server 9.0, with automated server migration and provisioning of services, policy-driven processing, self-tuning capabilities, increased overload protection, and cross-cluster failover.

See "Server Reliability, Availability, Scalability, and Performance" on page 15.

Enterprise Web Services

Now a J2EE standard, Web Services increase developer flexibility and choice by providing a common runtime environment and industry-standard support for Java annotations and Web Services extensions. The WebLogic Server implementation of Web Services 1.1 helps developers respond more effectively to changing business requirements, with true asynchronous messaging support for conversational applications; interoperability features designed for enterprise service-oriented architecture (SOA); and improved XML processing. Developers can leverage WebLogic Web Services to rapidly create, deploy, and adapt secure and fully interoperable Enterprise Web Services.

See "J2EE Standard Web Services" on page 19.

Enterprise-Ready Messaging Infrastructure

WebLogic Server 9.0 implements cross-domain communication; bi-directional transactions from Enterprise Information Systems; automatic destination migration for high availability; message store-and-forward for improved reliability; and tightly controlled order-of-message delivery. In addition, this release improves management and performance of enterprise and messaging-intensive applications.

See "Resource Adapters" on page 28.

System Administration and Management

The following sections describe new and changed features that affect overall server administration and management:

- "Improved Server Operations" on page 4
- "New Look, Feel, and Functionality in the Administration Console" on page 6
- "Portal Application Support in the Administration Console" on page 7
- "Predictable Distribution of Domain Configuration Changes" on page 7
- "Automation of Domain Configuration Tasks with WLST" on page 8
- "Centralized Diagnostic Service with More Visibility and Runtime Control" on page 9
- "WebLogic Logging Services" on page 12
- "Web Application and Resource Adapter Logging" on page 14
- "Changes to Data Storage That Affect Core Functionality" on page 14
- "Configuration Data Changes After WebLogic Server 9.0 BETA" on page 14
- "Restricted and Deprecated Utilities: wlconfig and weblogic.Admin" on page 14

Improved Server Operations

The following sections describe key changes and improvements to WebLogic Server core operations:

- "Server Life Cycle State Isolates Administration Operations" on page 4
- "Work Contexts Ease Implementation and Maintenance" on page 5
- "Network Channels Can Manage Traffic Between Server Instances" on page 5
- "Default Persistent Store Can Be Used Simultaneously by Subsystems" on page 5
- "Store-and-Forward Service Enables Highly Available Messaging" on page 6

Server Life Cycle State Isolates Administration Operations

A new life cycle state, `ADMIN`, facilitates application redeployment, maintenance, and troubleshooting. In the `ADMIN` state, WebLogic Server is running, but available only for administration operations, so you can perform server and application-level administration tasks without risk to running applications.

See "Understanding Server Lifecycle" in *Managing Server Startup and Shutdown*.

Work Contexts Ease Implementation and Maintenance

Developers can define properties as work contexts and can pass properties without explicitly including them in a remote call. A work context is propagated with each remote call, which allows the called component to add or modify properties defined in the work context. Similarly, the calling component can access the work context to obtain new or updated properties.

Work contexts facilitate diagnostics monitoring, application transactions, and application load-balancing, all of which require communication with remote components. They also provide information to third-party components.

Work contexts can be used by both clients and servers, and are enabled separately for each.

See *Developing WebLogic Server Applications*.

Network Channels Can Manage Traffic Between Server Instances

In addition to managing external network traffic, network channels can now manage network traffic between server instances. Other new and improved network channel capabilities include:

- SSL behaviors configurable on a per-channel basis
- Dynamic configuration and starting of channels without re-booting the server instance
- New capabilities for closing and restarting a channel
- Replication channel for replication traffic among server instances in a WebLogic Server cluster

See *Configuring WebLogic Server Environments*.

Default Persistent Store Can Be Used Simultaneously by Subsystems

The WebLogic Persistent Store is a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence, especially subsytems that require the creation and deletion of short-lived data objects, such as transactional messages for JMS Servers. Each server instance in a domain has a default persistent store that requires no configuration and that can be used simultaneously by subsystems that do not require explicit selection of a particular store, but can use the system's default storage. These subsystems include JMS Servers, Web

Services, EJB Timer services, Store-and-Forward services, and the JTA Transaction Log (TLOG). Optionally, administrators can configure dedicated file-based stores or JDBC-accessible stores to suit their environment.

See Using The WebLogic Persistent Store in *Configuring WebLogic Server Environments*.

Store-and-Forward Service Enables Highly Available Messaging

The WebLogic Store-and-Forward (SAF) service enables WebLogic Server to deliver messages reliably between applications that are distributed across WebLogic Server instances. For example, with the SAF service, an application that runs on or connects to a local WebLogic Server instance can reliably send messages to a destination that resides on a remote server. If the destination is not available at the moment the messages are sent, then the messages are durably saved on a local server instance, and are forwarded to the remote destination once it becomes available.

WebLogic JMS utilizes the SAF service to enable local JMS message producers to reliably send messages to remote JMS queues or topics. WebLogic Web Services build the implementation of the Web Services Reliable Messaging (WSRM) standard on top of the SAF service.

See *Configuring and Managing WebLogic Store-and-Forward*.

New Look, Feel, and Functionality in the Administration Console

The WebLogic Server Administration Console has been completely redesigned in the current release. The Administration Console is now built on the WebLogic Portal Framework, which makes it more open and more readily extensible, as described in "Portal Application Support in the Administration Console" on page 7. Other new features include:

- Improved navigation and user interface design.
- New application deployment and configuration tools, including assistants for installing applications, more configuration screens, and new deployment and redeployment controls for production applications. Additional Console updates enable you to assign values more easily to deployment plan variables when you deploy an exported application. See "Application Deployment" on page 46.
- The WebLogic Diagnostic Service, with many new features for configuring, collecting, and viewing diagnostic information in a runtime environment. You access the service through the Console. See "Centralized Diagnostic Service with More Visibility and Runtime Control" on page 9.

- Domain change management features, including a domain lock and reliable batch change capabilities. See "Predictable Distribution of Domain Configuration Changes" on page 7.

Portal Application Support in the Administration Console

The WebLogic Server Administration Console is completely redesigned in this release. Prior to 9.0, the Administration Console used JSPs and its own framework to render its user interface. Now the Console uses JSPs, the WebLogic Portal framework, Apache Struts, Apache Beehive, and other open technologies.

You can extend the Administration Console as you can extend portal applications generally, except that the Administration Console does not support JSR 168 or Web Services for Remote Portlets (WSRP). In addition to adding and replacing content, you can add a node to the navigation tree and change colors, branding images, and other aspects of the Console's appearance.

Note: Because the new architecture is so different, WebLogic Administration Console extensions built for prior releases of WebLogic Server will not function in 9.0. Although you might be able to reuse the content and some logic in the JSPs that you created in previous Administration Console extensions, BEA no longer supports the APIs or JSP tag libraries that it provided for extensions prior to 9.0. In addition, you must now define portlets as the container for your JSPs.

Predictable Distribution of Domain Configuration Changes

Improvements in change management enable you to distribute configuration changes throughout a domain securely, consistently, and predictably, regardless of whether you are using the Administration Console, WebLogic Scripting Tool (WLST), JMX, or other APIs.

To protect your changes and to prevent others from making changes, a new region in the Administration Console called the Change Center requires you to lock the Console before you begin to modify a domain configuration.

When you finish making changes, you save and distribute them to all server instances in the domain, or you can roll back changes and release the lock. Each server determines whether it can accept the change. If all servers can accept the change, they update their working configuration tree and the change is completed. If one or more servers do not accept the change, the changes are not made in any server, thus avoiding an incomplete intermediate state. This feature helps ensure that WebLogic Server configuration information is correct and consistent at all times.

WebLogic Server controls configuration changes in generally the same manner, whether the changes are implemented using the Administration Console, the WebLogic Scripting Tool, or the Configuration Manager service and JMX APIs:

- To learn about using change management with the WebLogic Scripting Tool, see Editing Configuration MBeans in *WebLogic Scripting Tool.*
- To learn about using change management with the Configuration Manager service and JMX, see Managing Configuration Changes in *Understanding Domain Configuration*.

Automation of Domain Configuration Tasks with WLST

WebLogic Scripting Tool (WLST) is a new command-line interface that you use to configure WebLogic Server instances and domains, and to manage and persist WebLogic Server configuration changes.

WLST enables you to:

- Retrieve domain configuration and runtime information.
- Edit the domain configuration and persist the changes in the `config.xml` file.
- Navigate and edit custom, user-created MBeans and non-WebLogic Server MBeans, such as WebLogic Integration Server and WebLogic Portal Server MBeans.
- Automate configuration tasks and application deployment.
- Clone WebLogic Server domains.
- Access Node Manager to start, stop, and suspend server instances remotely or locally, without requiring the presence of a running Administration Server.

Based on the Java scripting interpreter, Jython, WLST interprets commands interactively, supplied one-at-a-time from a command prompt, or in batches, supplied in a file (script), or embedded in your Java code. You can use the scripting tool *online* (connected to a running server) and *offline* (not connected to a running server).

- Online, WLST provides simplified access to MBeans. You can perform administrative tasks and initiate WebLogic Server configuration changes while connected to a running server.
- Offline, WLST only provides access to persisted configuration information. You can create a new domain or update an existing domain without connecting to a running WebLogic Server. This functionality resembles that of the Configuration Wizard.

Note: BEA Systems recommends that you use WLST in place of `weblogic.Admin` and `wlconfig`. See "Restricted and Deprecated Utilities: wlconfig and weblogic.Admin" on page 14.

Centralized Diagnostic Service with More Visibility and Runtime Control

The new WebLogic Diagnostic Service integrates all diagnostics features and functionality into a centralized, unified framework that enables you to create, collect, analyze, and archive diagnostic data in a standard format. The Diagnostic Service provides more visibility into and control of server runtime performance and applications, allowing you to diagnose and isolate faults as they occur.

The following sections describe WebLogic Diagnostic Service functionality. For more detailed information, see *Configuring and Using the WebLogic Diagnostic Framework*.

- "Increased Visibility" on page 9
- "Logging in a Standard Format" on page 10
- "Dynamic, Custom Instrumentation" on page 10
- "Diagnostic Context for Reconstruction and Correlation of Events" on page 10
- "Event Capture for a Single Request or a Single Client" on page 10
- "Data Harvesting from the Server and Applications" on page 11
- "Server Image Capture for Post-Failure Analysis" on page 11
- "Tight Integration with Legacy Monitoring Software" on page 11
- "Watches to Detect Specific Conditions, Analyze Data, and Trigger Listeners" on page 11
- "Dynamic and Offline Access to Current and Historical Data" on page 11
- "Standard Integration of Third-Party Analytic Tools" on page 12

Increased Visibility

The WebLogic Diagnostic Service provides a dynamic view of metrics, data events, and logging and debug information. It also exposes new diagnostic data in areas of the server that were not previously exposed.

Logging in a Standard Format

Previous releases of WebLogic Server included a number of independent logging capabilities, such as the standard server and domain logs, JDBC log, and access log. The logs relied on different implementations and thus did not benefit from the same services and facilities, such as rotation, as the standard log. These logging solutions are now unified by a single implementation that provides the same qualities of service for all server logging. You can configure the WebLogic Diagnostic Service to collect, analyze, and archive all events generated by WebLogic Logging Services.

Dynamic, Custom Instrumentation

Instrumentation capabilities in the Diagnostic Service enable you to:

- Selectively add diagnostic code to and remove it from Weblogic Server and user-written applications in a running environment.
- Collect and analyze particular types of runtime data events.
- Under certain circumstances, dynamically alter the behavior of the diagnostic code executed at specific locations while the server is running, by changing the diagnostic actions active at those locations.

Diagnostic Context for Reconstruction and Correlation of Events

The diagnostic context is a means of reconstructing transactional events and of correlating events based on the timing of the occurrence or on logical relationships. You can reconstruct a thread of execution from request to response, or generate diagnostic information only when contextual information in the diagnostic context satisfies certain criteria. This capability keeps the volume and overhead of generated information to manageable levels.

Event Capture for a Single Request or a Single Client

Most event collection is accomplished by designating key points in the application flow and then monitoring every request that passes through those points. However, to minimize collection volume and facilitate event isolation, it is sometimes preferable to capture only events for a single request or requests coming from a single client. The WebLogic Diagnostic Service allows you to mark, or dye, a particular request in such a way that the WebLogic Diagnostic Service can determine whether it should be monitored. The WebLogic Diagnostic Service marks requests when they enter the system by setting flags in the diagnostic context, discussed in "Diagnostic Context for Reconstruction and Correlation of Events" on page 10.

Data Harvesting from the Server and Applications

The Harvester component of the WebLogic Diagnostic Service can be configured to collect metrics contained in server MBeans. Additionally, because the WebLogic Diagnostic Service enables you to harvest metrics from custom MBeans that you provide, you can also collect metrics from your own applications.

Server Image Capture for Post-Failure Analysis

The Server Image Capture component of the WebLogic Diagnostic Service creates a diagnostic snapshot from the server, either on-demand or automatically. The most common sources of server state— configuration, Log Cache, WorkManager, JNDI state, and harvestable data— are captured, as well as images from JMS, JDBC, EJB, JNDI, and other server subsystems.

When a server fails and recovers repeatedly over a short period, for example, in storm-related power failures, system administrators can also specify an image-lockout period, which prevents the server from repeatedly capturing and persisting similar diagnostic images that unnecessarily consume system resources.

Tight Integration with Legacy Monitoring Software

Customers who develop software for trapping, routing, and handling of events from systems within their enterprise can configure the Diagnostic Service to analyze these custom application events and automatically generate notifications to alert system administrators. This capability enables a tighter integration with legacy monitoring frameworks in larger customer environments.

Watches to Detect Specific Conditions, Analyze Data, and Trigger Listeners

To monitor WebLogic Server, watches can be configured to detect specific conditions and to analyze logging and debugging log records, data events, and harvested metrics. Watches can also trigger different types of notification listeners, including Simple Mail Transfer Protocol (SMTP), Simple Network Management Protocol (SNMP), Java Management Extensions (JMX), and Java Message Service (JMS).

Dynamic and Offline Access to Current and Historical Data

All data collected from the server and applications running on it are persisted to permanent storage. The Data Accessor component provides access to all current and historical data collected by the WebLogic Diagnostic Service, including data events, log records, and harvested metrics.

You can access archived data online (on a running server) and offline (after the server shuts down).

Standard Integration of Third-Party Analytic Tools

The WebLogic Diagnostic Service provides standard integration capabilities for third-party monitoring and analytic tools:

- Standard interfaces and patterns of integration—Third-party tools and clients can collect data similar to the way in which the WebLogic Sever Administration Console collects data. Third-party monitoring tools with well-defined publishing interfaces can be integrated with the WebLogic Diagnostic Service; can collect data and consume events via JMX, JMS, SMTP, and SNMP; and can display data through a user-defined console.
- Historical archive of diagnostic data—Data collected by the WebLogic Diagnostic Service is archived so that the Administration Console and third-party tools can access the current state of a running server and historical data, and file system mechanisms can have offline access to persisted, historical data for a server that has shut down.

The WebLogic Diagnostic Service is compatible with existing diagnostic tools and capabilities, such as instrumentation, developed by BEA partners. The WebLogic Diagnostic Service makes it easier for partners to integrate their tools, while BEA Systems takes ownership of the integration interfaces and provides support, documentation, and an improved quality of service.

WebLogic Logging Services

WebLogic Logging Services provide several new features and configuration options.

Logging Volume Control

Enhancements to LogMBean interfaces let you control logging output by setting the severity level and filters on both loggers and handlers.

In earlier versions of WebLogic Server, system administrators and developers had only programmatic access to loggers and handlers. In this release, you can configure handlers by using MBeans, eliminating the need to write code for most basic logging configurations. The Administration Console and WebLogic Scripting Tool (WLST) provide an interface for interacting with logging MBeans. Loggers are configured only through the API.

Support for Log4J

WebLogic Server supports Jakarta Project Log4j. In the Administration Console, you specify Log4j or the default Java Logging implementation. Alternatively, you can configure Log4j logging through the `LogMBean` interface using the `LogMBean.isLog4jLoggingEnabled` attribute.

Support for Commons API

The Jakarta Commons Logging APIs provide an abstraction layer that insulates users from the underlying logging implementation. WebLogic Server provides an implementation of the Commons `LogFactory` interface, letting you issue requests to the server `Logger` by using this API.

Log Message Content Enhancements

All log messages include:

- Diagnostic context information and a request identifier to correlate messages coming from a specific request or application.
- Time stamp in milliseconds.

See *Configuring Log Files and Filtering Log Messages*.

New Log File Locations

WebLogic Server 9.0 includes the following changes to log file locations:

- The server log file is located in the `logs` directory below the server instance root directory; for example, *`root-directory`*`\servers\`*`server-name`*`\logs\`*`server-name.`*`log`.
- The domain log resides in the Administration Server `logs` directory. The default name and location for the domain log file is *`root-directory`*`\servers\`*`AdminServer-name`*`\logs\`*`domain-name.`*`log`.
- By default, the rotated files are stored in the same directory where the log file is stored. You can specify a different directory location for the archived log files by using the Administration Console or setting the `LogFileRotationDir` property of the `LogFileMBean` from the command line.

Web Application and Resource Adapter Logging

In this release of WebLogic Server, you can use WebLogic-specific deployment descriptors to configure Web application and resource adapter logging behavior. The logging configuration deployment descriptor elements define similar attributes used to configure server logging through the `LogFileMBean` interface, such as the log file name, location, and rotation policy.

Similarly, logging services are provided to J2EE resource adapters for `ManagedConnectionFactory` scoped logging. You configure the log file name, location, and rotation policy for resource adapter logs through the `weblogic-ra.xml` deployment descriptor.

See *Using WebLogic Logging Services for Application Logging*.

Changes to Data Storage That Affect Core Functionality

Data storage has changed as follows in WebLogic Server 9.0:

- New domain directory structure—Domain configuration data now resides in multiple files, including `config.xml`, in the new domain `config` directory. Configuration files are archived in JAR files in the `configArchive` directory under the domain directory. These changes affect recommended backup procedures. See “Avoiding and Recovering From Server Failure” in *Managing Server Startup and Shutdown.*
- Managed Servers cache configuration—A Managed Server always maintains a local copy of the domain configuration. See “Managed Server Independence Mode” in *Managing Server Startup and Shutdown*.

Configuration Data Changes After WebLogic Server 9.0 BETA

Configuration data such as JDBC resource names, security data, persistent store, and the XML schema for the `config.xml` file may have changed since WebLogic Server 9.0 BETA. If you were using WebLogic Server 9.0 BETA, your `config.xml` file is probably not usable in WebLogic Server 9.0 GA.

Restricted and Deprecated Utilities: wlconfig and weblogic.Admin

The `weblogic.Admin` utility is deprecated as of WebLogic Server 9.0. Both `weblogic.Admin` and `wlconfig` are now restricted as follows:

- Cannot be used to access MBeans that are new to this release. These tools use the compatibility MBean server to access MBeans, and this MBean server does not contain MBeans that are new in WebLogic Server 9.0.
- Cannot be used to configure security MBeans, but can still be used to view and invoke methods on the security MBeans.
- Cannot be used to create or modify Local Configuration MBeans, but can be used to view these beans and invoke their operations.

BEA Systems recommends that you use the WebLogic Scripting Tool (WLST) for equivalent functionality. See *WebLogic Scripting Tool*.

Server Reliability, Availability, Scalability, and Performance

WebLogic Server 9.0 significantly improves server and cluster RASP:

- "Automatic and Manual Server Migration to Another Machine" on page 15
- "Support for CommonJ Timer and Work Manager API Specification" on page 16
- "HTTP Session Replication and Failover Across Wide and Metropolitan Area Networks" on page 16
- "Server Self-Tuning for Production Environments" on page 17
- "Node Manager Enhancements" on page 18
- "Dynamically Generated Cluster Address for Each Request" on page 19
- "New Overload Protection Increases Availability" on page 19

Automatic and Manual Server Migration to Another Machine

Note: Server Migration is not supported on all platforms. See Server Migration in *Supported Configurations for WebLogic Server 9.0.*

WebLogic Server now supports automatic and manual migration of a clustered server instance and all the services it hosts from one machine to another. This feature is designed for environments with high availability requirements. Use the server migration capability to:

- Increase availability of services that must run on only a single server instance at any given time in a cluster, such as JMS, when the hosting server instance fails.

- Manually migrate a server instance at any time, not just when it fails, for administrative reasons.

See “Server Migration” in *Using WebLogic Server Clusters*.

Support for CommonJ Timer and Work Manager API Specification

WebLogic Server 9.0 supports part of the BEA and IBM Joint Specifications (CommonJ) described at http://dev2dev.bea.com/technologies/commonj/index.jsp. In particular, this release implements the Timer and Work Manager 1.1 Specification, available at http://dev2dev.bea.com/technologies/commonj/twm/index.jsp.

The Timer API provides a simple API that applications can use to create and use timers managed within the application server, and is a recommended alternative to the J2SE `java.util.Timer` class.

The Work Manager API enables an application to break a single request task into multiple work items, and assign those work items to execute concurrently by using multiple Work Managers configured in WebLogic Server. (Applications that do not need to execute concurrent work items can also use configured Work Managers by referencing or creating Work Managers in their deployment descriptors or, for J2EE Connectors, by using the JCA API.) See also “Server Self-Tuning for Production Environments” on page 17 for more information about the new Work Manager tuning strategies.

HTTP Session Replication and Failover Across Wide and Metropolitan Area Networks

The state of an HTTP session running on a server instance in one cluster can be replicated on a server instance in a different WebLogic Server cluster. The clusters can be located on different LANs within the same metropolitan area network (MAN), or can be in geographically distant locations—in different cities or states—within a wide area network (WAN). Upon a failure of the primary server instance, another member of the same cluster can recover the session data from the remote instance (in another cluster) and make it available in the primary cluster. If no members of the cluster in which the primary failed are available, the request can fail over to the remote cluster hosting the session replica.

Here are two example environments in which WebLogic Server cross-cluster session replication features are useful:

- WAN replication—A company has one data center in San Francisco and another in Los Angeles, each with a WebLogic Server cluster. In the event of a complete data center failure, the company's service level agreement allows session requests to fail over to the other data center. WebLogic Server's WAN session state replication can be used. WAN session replication entails:
  - In-memory replication to a local server instance.
  - Asynchronous JDBC persistence to a remote cluster instance. Periodically, session states flushed to storage in a table on the server instance on the remote cluster. Because the JDBC persistence to the remote server instance is performed asynchronously, it provides improved performance over synchronous JDBC replication, in which the database write paces session completion.
- MAN replication—A company has two locations with a fast low-latency connection between the locations. In both sites, a WebLogic Server cluster hosts an application with high availability requirements. The session state is replicated between two clusters synchronously.

See *Using WebLogic Server Clusters*.

Server Self-Tuning for Production Environments

New self-tuning capabilities simplify the process of configuring WebLogic Server for production environments with service-level requirements that vary over time or by application. Self-tuning helps prevent deadlocks during periods of peak demand. Self-tuning features are also useful if your WebLogic Server environment hosts multiple applications with different performance and availability requirements. For example, you can allocate a greater percentage of resources to a user-facing order processing application than to a back-end inventory management application.

The new queue strategy enables administrators to allocate processing resources and manage performance more effectively, by avoiding the effort and complexity involved in configuring, monitoring, and tuning custom execute queues.

Key self-tuning features in WebLogic Server 9.0 include:

- Workload management—Administrators can define scheduling policies and constraints at the domain level, application level, and module level.
- Automatic thread count tuning—A thread pool can maximize throughput by automatically changing its size, based on throughput history and queue size.
- Thread scheduling functionality—WebLogic Server 9.0 implements the Commonj Work Manager API, exposing thread scheduling functionality to developers. Applications can

also use the Work Manager API to execute work asynchronously and receive notifications on the execution status.

See *Configuring WebLogic Server Environments*.

Node Manager Enhancements

Many enhancements make Node Manager more versatile and easier to use:

- Administration Server control—In previous versions, Node Manager required access to a running Administration Server, and could control and monitor only Managed Servers. In WebLogic Server 9.0, Node Manager can start, monitor, and restart Administration Servers.
- Shell Script Node Manager—WebLogic Server 9.0 provides a shell script version of Node Manager with the same functionality as the Java Node Manager. In addition to Remote Shell (RSH) protocol, Node Manager can be configured with the Secure Shell (SSH) for secure remote control of server instances running on UNIX or Linux systems.
- WebLogic Scripting Tool (WLST) support— You can now use WLST commands to access Node Manager and to start, stop, and suspend server instances remotely or locally; obtain server status; and retrieve the contents of the server output log, without requiring the presence of a running Administration Server. In addition, you can configure the machine on which WLST is running to be monitored by Node Manager. See "Automation of Domain Configuration Tasks with WLST" on page 8.
- Node Manager runs as a Windows service—BEA Systems recommends running Node Manager as an operating system service so that it is automatically restarted in the event of system failure or reboot and using Node Manager to start or restart servers. See "Installing the Node Manager as a Windows Service" in the *Installation Guide*.
- Clustered server migration—Node Manager is used to accomplish migration of servers in a WebLogic Server cluster. For more information, see "Server Migration" in *Using WebLogic Server Clusters*.
- Improved diagnostics and logging—Node Manager diagnostics are improved, and the logging strategy for Node Manager and the server instances it controls are simplified.
- Simplified setup—Among other improvements, Node Manager no longer requires two-way SSL. Only one-way SSL is required.
- Start Script Support—You can configure Node Manager to use a script to start WebLogic Server instances. Optimal settings for heap size are 512 MB for the Administration Console and other example server /xbus domains. You might want to start the server with

the default allocation of 256 MB of Java heap memory to the WebLogic Server. To do so, you can start the server with the java -Xms256m and -Xmx512m options. With settings of -Xms256m -Xmx512m, the JVM expands the heap up to 512 MB if necessary.

See Using Node Manager to Control Servers in *Managing Server Startup and Shutdown*.

Dynamically Generated Cluster Address for Each Request

In this release, you can still explicitly define a cluster address, but if you do not, WebLogic Server dynamically generates the cluster address for each request. This capability eases the configuration effort, and ensures that the cluster address correctly reflects the cluster membership at startup. See "Cluster Address" in *Using WebLogic Server Clusters*.

New Overload Protection Increases Availability

New overload features protect a server instance from out-of-memory (OOM) exceptions and execute queue overloads, thus increasing the availability of a server or a cluster.

See *Configuring WebLogic Server Environments*.

J2EE Standard Web Services

Web Services are now a J2EE standard, which has resulted in many changes between 8.1 and 9.0 WebLogic Web Services.

WebLogic Server 9.0 includes the following changes and new features:

- "Differences Between WebLogic 8.1 and 9.0 Web Services" on page 20
- "JWS Annotations-Based Programming Model" on page 20
- "Web Services for J2EE 1.1 Implementation" on page 21
- "Asynchronous, Loosely Coupled Web Services" on page 21
- "Digital Signatures and Encryption" on page 21
- "Automatic Data Binding Between XML and Java" on page 22
- "Use of WS-Policy Files" on page 22
- "Ant Tasks That Handle JWS Files" on page 22
- "Implementations of and Conformance to Standard Web Services and Java Specifications" on page 23

Differences Between WebLogic 8.1 and 9.0 Web Services

J2EE 1.4 introduces a standard Java component model for authoring Web Services, with new specifications such as *Web Services for J2EE, Version 1.1* (JSR-921, the 1.1 maintenance version of JSR-109) and *Java API for XML Registries* (JAX-R), as well as updated JAX-RPC and SAAJ specifications.

As a result, the programming model used to create WebLogic Web Services has also changed. The model now takes advantage of the powerful new metadata annotations feature introduced in Version 5.0 of the JDK. Additionally, the runtime environment upon which WebLogic Web Services run now supports the *Web Services for J2EE, Version 1.1* specification.

The entire 8.1 Web Services runtime, which includes the `weblogic.webservice.*` APIs, is deprecated. You should now use both the new programming model (JWS files with accompanying Ant task) and the `weblogic.wsee.*` packages instead. See the javadocs for the full list of public WebLogic APIs. See Programming Web Services for WebLogic Server for information about the new 9.0 WebLogic Web Services runtime.

Note: 8.1 WebLogic Web Services continue to run, without any changes, on Version 9.0 of WebLogic Server because the 8.1 Web Services runtime is still supported in 9.0, although it is deprecated and will be removed from the product in a future release. For this reason, BEA highly recommends that you upgrade your 8.1 Web Service to 9.0. See Upgrading an 8.1 Web Service to 9.0.

There is an additional small change in the Web Services of the Avitek Medical Records (MedRec) sample application between version 8.1 and 9.0. In 8.1, the MedRec application used a non-public API in the `weblogic.webservice.tools.wsdlp` package. In 9.0, this package has moved to `weblogic.webservice.wsdl` and is deprecated. The Web Services in the 9.0 Medrec application use the 9.0 Web Services runtime and programming model.

See Differences Between 8.1 and 9.0 WebLogic Web Services at {DOCROOT}/webserv/intro.html#8.1diff.

JWS Annotations-Based Programming Model

The programming model used to create 9.0 WebLogic Web Services is based on the new JDK 5.0 metadata annotations feature. In this model, the implementation of your Web Service is a Java file that uses Java Web Service (JWS) annotations. These JWS annotations are a combination of standard ones, defined by the *Web Services Metadata for the Java Platform* specification (JSR-181), and WebLogic-specific ones.

Note: Although using JWS annotations is the preferred programming model for creating WebLogic Web Services, you can also create one manually by programming the EJB or Java class that implements the Web Service and creating all the required components, such as the Web Service deployment descriptors and the WSDL file.

See Programming the JWS File at {DOCROOT}/webserv/jws.html.

Web Services for J2EE 1.1 Implementation

The *Web Services for J2EE*, Version 1.1 specification defines the standard J2EE runtime architecture for implementing Web Services in Java.

See Anatomy of a WebLogic Web Service at {DOCROOT}/webserv/overview.html#Anatomy.

Asynchronous, Loosely Coupled Web Services

WebLogic Web Services support a variety of asynchronous features:

- *Web Services Reliable Messaging*—Two Web Services running on different application servers, each of which implements the WS-ReliableMessaging specification, can communicate reliably in the presence of failures in software components, systems, or networks.
- *Conversational Web Services*—Two or more parties maintain state across multiple invocations.
- *Buffered Web Services*—WebLogic Server places the invocation of the operation on a JMS queue and processes it asynchronously.
- *Asynchronous Request-Response*—A client application, when invoking a Web Service, does not wait for the response, but rather, continues on with its work and handles the response later on.

Digital Signatures and Encryption

WebLogic Web Services support the digital signature and encryption of request and response SOAP messages generated during invocation of a Web Service. This capability derives from conformance with the following OASIS Standard 1.0 Web Services Security specifications:

- SOAP Message security
- Username Token Profile
- X.509 Token Profile

You configure message-level security through security WS-Policy statements, as specified by the WS-Policy specification.

See Configuring Security at {DOCROOT}/webserv/security.html.

Automatic Data Binding Between XML and Java

As in previous releases, WebLogic Web Services support a full set of built-in XML Schema, Java, and SOAP types, as specified by the JAX-RPC 1.1 specification, that you can use in your Web Service operations without performing any additional programming steps.

Additionally, you can define a variety of XML and Java data types, including `com.bea.xml.XMLBeans`, as input parameters and return values for your Web Service. The WebLogic Web Services Ant tasks automatically generate the data binding components needed to convert the user-defined data types between their XML and Java representations.

See Data Types and Data Binding at {DOCROOT}/webserv/data_types.html.

Use of WS-Policy Files

WebLogic Web Services 9.0 implement the WS-Policy specification, which provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service. In this release, policy files are used only for configuring the Web Services reliable messaging and message-level security features.

For information on how WS-Policy files are used to configure security and reliable messaging, see Use of WS-Policy File for Message-Level Security Configuration and Use of WS-Policy Files for Reliable SOAP Message Configuration.

Ant Tasks That Handle JWS Files

A set of Ant tasks handle JWS annotated files and integrate them into the WebLogic split development directory environment. This development environment consists of a directory layout and associated Ant tasks that help you repeatedly build, change, and deploy J2EE applications, including Web Services.

See Web Services Ant Task Reference at {DOCROOT}/webserv/anttasks.html.

Implementations of and Conformance to Standard Web Services and Java Specifications

Web Services in WebLogic Server 9.0 implement a variety of standard Web Services-related Java specifications and conform to a variety of standard Web Services specifications. For the full list, see Standards Supported by WebLogic Web Services.

Messaging (JMS)

WebLogic Server 9.0 introduces major changes in the configuration, deployment, and dynamic administration of WebLogic JMS.

- "JMS 1.1 Specification Support for Production Environments" on page 24
- "Modular Configuration and Deployment of JMS Resources" on page 24
- "Store-and-Forward for Highly Available Message Production" on page 24
- "Enhanced Runtime Message Management" on page 24
- "Pause-and-Resume Message Operations on Destinations" on page 25
- "More Transparency with Message Life Cycle Logging" on page 25
- "Debug and Diagnostic Information More Readily Available" on page 25
- "Strict Message Ordering with Unit-of-Order" on page 25
- "Uniform Distributed Destinations" on page 26
- "Improved Message Paging" on page 26
- "Access to JMS Applications from Programs Written in C" on page 26
- "Message-Driven Bean (MDB) Enhancements for JMS" on page 26
- "Document Object Model (DOM) Support for XML Messages" on page 27
- "Deprecated JMS Features, Methods, and Interfaces" on page 27

For more information about these and other new features and changes in WebLogic JMS, see "New and Changed JMS Features In This Release" in *Configuring and Managing WebLogic JMS*.

JMS 1.1 Specification Support for Production Environments

WebLogic Server 9.0 is compliant with the new JMS 1.1 Specification for use in production, and so supports unified APIs that work for both queues and topics. See the Java JMS technology page on the Sun Web site at `http://java.sun.com/products/jms/`.

Modular Configuration and Deployment of JMS Resources

JMS configurations in WebLogic Server 9.0 are stored as modules, defined by an XML file that conforms to the `weblogic-jmsmd.xsd schema`, similar to standard J2EE modules. An administrator can create and manage JMS modules as global system resources, as modules packaged with a J2EE application (as a packaged resource), or as stand-stand-alone modules that can be made globally available. For more details, see "JMS and JDBC Deployable Resource Configuration" on page 49.

With modular deployment of JMS resources, you can migrate your application and the required JMS configuration from environment to environment, such as from a testing environment to a production environment, without opening an enterprise application file (such as an EAR file) or a JMS stand-alone module, and without extensive manual JMS reconfiguration.

See Understanding JMS Resource Configuration in *Configuring and Managing WebLogic JMS*.

Store-and-Forward for Highly Available Message Production

The JMS Store-and-Forward feature is built on the WebLogic Store-and-Forward (SAF) service to provide highly-available JMS message production. For example, a JMS message producer connected to a local server instance can reliably forward messages to a remote JMS destination, even though that remote destination might be temporarily unavailable when the message was sent. JMS Store-and-Forward is transparent to JMS applications; therefore, JMS client code still uses the existing JMS APIs to access remote destinations.

See "Understanding the Store-and-Forward Service" in *Configuring and Managing WebLogic Store-and-Forward*.

Enhanced Runtime Message Management

New message administration enhancements greatly enhance a JMS administrator's ability to view and browse *all* messages, and to manipulate *most* messages in a running JMS server, by using the Administration Console or new public runtime APIs. These message management enhancements include message browsing (for sorting), message manipulation (such as move and delete), message import and export, transaction management, durable subscriber management, and JMS client connection management.

See Monitoring JMS Statistics and Managing Messages in *Configuring and Managing WebLogic JMS*.

Pause-and-Resume Message Operations on Destinations

New WebLogic JMS configuration and runtime APIs enable an administrator to pause and resume message production, message insertion (in-flight messages), and message consumption operations on a given JMS destination, or on all destinations hosted by a single JMS server. This capability enables administrative control of the JMS subsystem behavior in the event of an external resource failure.

See Troubleshooting WebLogic JMS in *Configuring and Managing WebLogic JMS*.

More Transparency with Message Life Cycle Logging

The message life cycle is an external view of the basic events that a JMS message will traverse once it is accepted by the JMS server, either through the JMS APIs or the JMS Message Management APIs. Message Life Cycle Logging provides an administrator with more information about JMS messages from the JMS server viewpoint, in particular about basic life cycle events such as message production, consumption, and removal.

See Troubleshooting WebLogic JMS in *Configuring and Managing WebLogic JMS*.

Debug and Diagnostic Information More Readily Available

The JMS subsystem uses the new system-wide WebLogic Diagnostic Service for centralized debug access and logging. Debug and diagnostic information is more readily available so you can easily diagnose and fix problems.

See *Understanding the WebLogic Diagnostic Service*.

Strict Message Ordering with Unit-of-Order

The Message Unit-of-Order feature goes beyond the message delivery ordering requirements in the JMS 1.1 Specification by enabling JMS message producers to group ordered messages into a single unit. The Unit-of-Order feature guarantees that all messages created from that unit are processed sequentially, in the order that they were created, by the same consumer until that consumer acknowledges them or the queue is closed. For example, if a queue has messages with many consumers, and each message has an account number as a Unit-of-Order, then two consumers will not process messages with the same account number at the same time.

See Using Message Unit-of-Order in *Programming WebLogic JMS*.

Uniform Distributed Destinations

A new type of distributed destination, the *uniform distributed destination*, greatly simplifies the management and development of distributed destination applications. Using this feature, the administrator no longer needs to create or designate destination members, but relies on the system to uniformly create the necessary members on the JMS servers to which a JMS module is targeted. This feature ensures the consistent configuration of all distributed destination parameters, particularly in regard to weighting, security, persistence, paging, and quotas.

See Configuring Cluster WebLogic JMS Resources in *Configuring and Managing WebLogic JMS*.

Improved Message Paging

The *message paging* feature for freeing up virtual memory during peak message load situations is always enabled on JMS servers. Additionally, administrators no longer need to create a dedicated message paging store since paged out messages can be stored in a directory on your file system. However, for the best performance you should specify that messages be paged to a directory other than the one used by the JMS server's persistent store.

See "Paging Out Messages To Free Up Memory" in *Configuring and Managing WebLogic JMS*.

Access to JMS Applications from Programs Written in C

The Weblogic JMS C API enables programs written in "C" to participate in JMS applications. This implementation of the JMS C API uses JNI in order to access a Java Virtual Machine (JVM).

See WebLogic C API in *Programming WebLogic JMS*.

Message-Driven Bean (MDB) Enhancements for JMS

MDB enhancements include transaction batching (processing multiple messages in a single transaction) and load balancing of distributed destinations across member destinations in different clusters or domains, regardless of whether the MDB and destination reside in the same cluster or in different clusters or domains.

See "Message-Driven Bean Enhancements" on page 37.

Document Object Model (DOM) Support for XML Messages

The WebLogic JMS API now provides native support for the Document Object Model (DOM) in the transmission of XML messages. This capability improves performance for implementations that already use a DOM, because those applications do not have to flatten the DOM before sending XML messages.

See Sending XML Messages in *Programming WebLogic JMS*.

Deprecated JMS Features, Methods, and Interfaces

In WebLogic Server 9.0, many changes were made to the JMS subsystem, including the removal of some classes and the deprecation of many MBeans. For a complete listing of deprecated JMS APIs, see New and Changed JMS Features In This Release in *Configuring and Managing WebLogic JMS*.

Legacy JMS Configuration Interfaces

The new descriptor-based method of configuring JMS resources uses Java Descriptor Bean interfaces to create deployable JMS resource modules. This fundamental change necessitated the deprecation of most JMS configuration MBean interfaces, with the exception of the JMSServerMBean interface.

JMS Helper APIs

The descriptor-based method of configuring JMS module resources necessitated the deprecation of the JMSHelper class for locating JMS runtime and configuration JMX MBeans. This class is replaced by a new JMSModuleHelper class, with methods for locating JMS runtime MBeans, and managing JMS Module configuration entities in a given module, including the JMS Interop Module.

See the JMSModuleHelper Javadoc.

JMS Session Pool and JMS Connection Consumer MBean Interfaces

The JMSSessionPoolMBean, its associated methods on the JMSServerMBean, and the JMSConnectionConsumerMBean interfaces have been deprecated. These interfaces were used to automatically create a JMS session pool and start the JMS consumers on the server side. The ConnectionConsumer and ServerSessionPool APIs are still supported, but BEA strongly recommends using message-driven beans (MDBs), which are simpler, easier to manage, and more capable.

JMS File Store and JMS JDBC Store MBean Interfaces

The new WebLogic Persistent Store necessitated the deprecation of the JMSStoreMBean, JMSFileStoreMBean, and JMSJDBCStoreMBean interfaces. This deprecation also includes any associated JMS Store methods on the JMSServerMBean interface.

See Using The WebLogic Persistent Store in *Configuring WebLogic Server Environments*.

Resource Adapters

WebLogic Server now fully supports the J2EE 1.5 Connector Architecture as well as resource adapters based on the J2EE 1.0 Connector Architecture. Deployment descriptors for version 1.5 are schema-based, while deployment descriptors for version 1.0 are DTD-based. Except where noted, the following sections describe new functionality for version 1.5 adapters:

- “Multiple Outbound Connections” on page 28
- “Inbound-Outbound Transactions” on page 29
- “Connection Proxies No Longer Necessary” on page 29
- “RAR Visibility to External Application Components” on page 29
- “Resource Adapter Life Cycle Management” on page 29
- “Faster In-Flight Transactions” on page 30
- “Self-Tuning Work Manager to Avoid Direct Creation of Threads” on page 30
- “New Resource Adapter Security Identities” on page 30
- “Adapters Bound in JNDI Tree as Independent Objects” on page 30
- “Connection Factory-Specific and Adapter-Scoped Properties” on page 30
- “Viability Test (ping) for Single and Pooled Connections” on page 30
- “Configurable Connection Proxy Generation for 1.0 Adapters” on page 31
- “Deprecation of link-ref Mechanism” on page 31

Multiple Outbound Connections

The J2EE 1.5 Connector Architecture now supports resource adapters with multiple outbound connections. The outbound resource adapter enables an application to connect to an EIS system and perform work. All communication is initiated by the application. The resource adapter serves as a passive library for connecting to an EIS and executes in the context of the application threads. See "Understanding Resource Adapters" in *Programming WebLogic Resource Adapters*.

Inbound-Outbound Transactions

Resource adapters in previous versions supported outbound messaging. Now, Version 1.5 resource adapters can also receive transactions, including messages, from an EIS. An EIS can send a transactional context under which messages are delivered or work is performed in the form of Work Requests. A message endpoint application (a message-driven bean and possibly other J2EE components) receives inbound messages from the EIS through the resource adapter.This functionality makes the WebLogic Server proprietary InterposedTransactionManager available through a J2EE standard interface, allowing J2EE applications to fully participate in an enterprise environment controlled by an external Transaction Manager. Prior to EJB 2.1, a message-driven bean (MDB) only supported Java Message Service (JMS) messaging. See "Messaging and Transaction Inflow" in *Programming WebLogic Resource Adapters*.

Connection Proxies No Longer Necessary

Late transaction enlistment and idle connection detection are now available to 1.5 resource adapters without requiring connection proxies. See "Connection Management" in *Programming WebLogic Resource Adapters*.

RAR Visibility to External Application Components

The Connector specification prohibits the application server from providing access to resource adapters defined in an EAR by application components outside the EAR. However, WebLogic Integration has a continuing requirement to provide such access. The `enable-access-outside-app` element of the `weblogic-ra.xml` deployment descriptor provides a configuration parameter for explicitly enabling this access. See "weblogic-ra.xml Schema" in *Programming WebLogic Resource Adapters*.

Resource Adapter Life Cycle Management

The application server calls new start() and stop() methods as part of its deployment and undeployment actions on 1.5 resource adapters. See "Packaging and Deploying Resource Adapters" in *Programming WebLogic Resource Adapters*.

Faster In-Flight Transactions

New suspend() and resume() methods enable the 1.5 resource adapter to shut down all incoming messages while allowing in-flight transactions to complete and then to resume normal operations. See "Creating and Configuring Resource Adapters" in *Programming WebLogic Resource Adapters*.

Self-Tuning Work Manager to Avoid Direct Creation of Threads

The J2EE 1.5 Connector Architecture specification advises against a resource adapter creating threads. To enable resource adapters to perform work without direct creation of threads, a self-tuning, configurable Work Manager has been added to create WorkRequests under the control of the application server. The Work Manager is configurable and allows you to manage resources. See "Messaging and Transaction Inflow" in *Programming WebLogic Resource Adapters*.

New Resource Adapter Security Identities

A number of new security identities can now be configured through the weblogic-ra.xml deployment descriptor. See "Security" in *Programming WebLogic Resource Adapters*.

Adapters Bound in JNDI Tree as Independent Objects

Version 1.0 resource adapters are identified by their ConnectionFactory objects bound in the JNDI tree. Version 1.5 resource adapters are now bound in the JNDI tree as independent objects, making them available as system resources in their own right or as message sources for MDBs. See "Connection Management" in *Programming WebLogic Resource Adapters*.

Connection Factory-Specific and Adapter-Scoped Properties

All transaction and security property settings apply to all outbound connection factories. BEA has extended this functionality to allow per-connection-factory settings as well as resource adapter-scoped properties. This includes a credential map per connection factory. See "Security" in *Programming WebLogic Resource Adapters*.

Viability Test (ping) for Single and Pooled Connections

You can now test either a specific outbound connection or the entire pool of outbound connections for a particular ManagedConnectionFactory. Testing the entire pool will test each connection in the pool individually. See "Connection Management" in *Programming WebLogic Resource Adapters*.

Configurable Connection Proxy Generation for 1.0 Adapters

If you already know whether a connection proxy can be used in the resource adapter, you can avoid a proxy test by explicitly setting the `use-connection-proxies` element in the WebLogic Server 8.1 version of `weblogic-ra.xml` to `true` or `false`. This is applicable to resource adapters based on the J2EE 1.0 Connector Architecture, which are still supported in this release. See "Connection Management" in *Programming WebLogic Resource Adapters*.

Deprecation of link-ref Mechanism

The link-ref mechanism is a 1.0 resource adapter feature that is deprecated in the current release. The feature allows a base resource adapter to be referenced by a child resource adapter, giving the child access to the base resource adapter's classes and configuration. For 1.5 resource adapters, this mechanism is replaced by the federated application, which is a stand-alone module that can be accessed by any application. See "Creating and Configuring Resource Adapters" in *Programming WebLogic Resource Adapters*.

JNDI

Foreign JNDI is an API that allows you to access objects on a remote JNDI tree without having to connect directly to the remote tree. For more information on Foreign JNDI, see "Setting Up Foreign JNDI" in *Programming WebLogic JNDI*.

JDBC

The following sections describe new features and changes for JDBC in WebLogic Server 9.0:

- "JDBC 3.0 Support" on page 32
- "RowSet Extensions" on page 32
- "Support for the J2EE Management Model (JSR-77)" on page 32
- "Modular Configuration and Deployment of JDBC Resources" on page 33
- "Fewer Resource Types for Simpler JDBC Configuration" on page 33
- "JDBC Monitoring and Diagnostics Enhancements" on page 33

- "Optimized Performance for Non-XA Resources in Global Transactions" on page 34
- "Credential Mapping of WebLogic and Database User IDs" on page 35
- "Multi Data Sources Supported for XA Transactions" on page 35
- "Updated WebLogic Type 4 JDBC Drivers" on page 35
- "Deprecated JDBC Features, Methods, Interfaces, and MBeans" on page 35

For more information, see "New and Changed Features in WebLogic JDBC" in *Configuring and Managing WebLogic JDBC*.

JDBC 3.0 Support

WebLogic Server 9.0 is compliant with the JDBC 3.0 specification. See the Java JDBC technology page on the Sun Web site at http://java.sun.com/products/jdbc/ and *Configuring and Managing WebLogic JDBC*.

RowSet Extensions

The WebLogic RowSet implementation complies with and extends the new JDBC RowSet Implementations Specification (JSR-114). See the Java JDBC technology page on the Sun Web site at http://java.sun.com/products/jdbc/.

WLCachedRowSets extends and can be used interchangeably with several standard RowSet types. It also includes methods for setting optimistic concurrency options and data synchronization options. SharedRowSet extends CachedRowSets so that additional CachedRowSets can be created for use in other threads based on data in an original CachedRowSet. SortedRowSets extends CachedRowSets so that rows in a CachedRowSet can be sorted in memory rather than depending on the database management system for sort processing. SharedRowSets and SortedRowSets increase performance by reducing the number of database round-trips required by an application.

See "Using RowSets with WebLogic Server" in *Programming WebLogic JDBC*.

Support for the J2EE Management Model (JSR-77)

WebLogic Server 9.0 JDBC fully supports JSR-77, which defines the J2EE Management Model. You access the J2EE Management Model to monitor resources, including the WebLogic JDBC system as a whole, JDBC drivers loaded into memory, and JDBC data sources.

See *Configuring and Managing WebLogic JDBC*.

Modular Configuration and Deployment of JDBC Resources

JDBC configurations in WebLogic Server 9.0 are now stored in XML documents that conform to the new `weblogic-jdbc.xsd` schema. You create and manage JDBC resources either as system modules, similar to the way they were managed prior to version 9.0, or as application modules. JDBC application modules are a WebLogic-specific extension of J2EE modules and can be deployed either within a J2EE application or as stand-alone modules. See "JMS and JDBC Deployable Resource Configuration" on page 49.

In support of the new deployment model for JDBC application modules in WebLogic Server 9.0, BEA Systems provides a schema for WebLogic JDBC modules. Each JDBC system module or application module that you create must conform to the schema. IDEs and other tools can validate JDBC modules based on the schema.

For more information about WebLogic JDBC resources, see *Configuring and Managing WebLogic JDBC*.

Fewer Resource Types for Simpler JDBC Configuration

Fewer JDBC resource types simplify JDBC configuration and reduce the likelihood of configuration errors. Instead of configuring a JDBC connection pool and then configuring a data source or transactional (tx) data source to point to the connection pool and bind to the JNDI tree, you configure a data source that encompasses a connection pool.

Also, MultiDataSources replace MultiPools. A MultiDataSource does not require a separate data source to bind it to the JNDI tree. If you use the Administration Console, you can configure the MultiDataSource and all encompassed data sources in one step.

See *Configuring and Managing WebLogic JDBC*.

JDBC Monitoring and Diagnostics Enhancements

The following enhancements facilitate JDBC monitoring and diagnostics.

New Monitoring Statistics and Profile Information for JDBC Resources

New data source usage information is available through the Administration Console and JMX:

- Profile information—Detailed usage profiles of JDBC resources, including current applications that are using pooled connections, current applications waiting for a connection, and current entries in the prepared statement cache.

- Monitoring and tuning statistics —Total number of connections used over the life of a connection pool, the number of applications waiting for a connection from the connection pool, average time an application waits for a connection, and average time a connection is in use by an application.

See “Monitoring WebLogic JDBC Resources” in *Configuring and Managing WebLogic JDBC.*

Callbacks for Monitoring Driver-Level Statistics

Callbacks for methods called on a JDBC driver enable you to monitor and profile JDBC driver usage, including methods being executed, exceptions thrown, and the time spent executing driver methods.

See “Monitoring WebLogic JDBC Resources” in *Configuring and Managing WebLogic JDBC.*

JDBC Debugging Enhancements

The JDBC subsystem uses the new system-wide WebLogic Diagnostic Service for centralized debugging access and logging.

See *Understanding the WebLogic Diagnostic Service*.

Optimized Performance for Non-XA Resources in Global Transactions

The Logging Last Resource (LLR) transaction option for a data source enables one non-XA resource to participate in a global transaction with improved performance and with the same ACID (atomic, consistent, isolated, and durable) guarantee as XA.

LLR transaction optimization confers these advantages:

- No need for XA processing at the database level if the database is the one non-XA resource.
- No need for an XA JDBC driver to connect to the database. XA JDBC drivers are typically inefficient compared to non-XA JDBC drivers.
- Fewer transaction processing steps, which reduces network traffic and the number of disk I/Os.

See “Understanding the Logging Last Resource Transaction Option” in *Configuring and Managing WebLogic JDBC*.

Credential Mapping of WebLogic and Database User IDs

Credential mapping for a JDBC data source enables WebLogic Server to set a mapped database user ID as a light-weight client ID on a database connection. This capability can reduce the need to create new connections with a specific database user ID and may enable your application to take advantage of connection pooling performance advantages.

See "Configuring Credential Mapping for a Data Source" in *Configuring and Managing WebLogic JDBC*.

Multi Data Sources Supported for XA Transactions

JDBC multi data sources are supported for use in XA transactions with Oracle Real Application Clusters (RAC).

See "Using Multi Data Sources with Global Transactions" in *Configuring and Managing WebLogic JDBC*.

Updated WebLogic Type 4 JDBC Drivers

Updates to the WebLogic Type 4 JDBC drivers from DataDirect resolve important issues and include some notable enhancements. See *WebLogic Type 4 JDBC Drivers* for more information.

Deprecated JDBC Features, Methods, Interfaces, and MBeans

The following drivers were removed from WebLogic Server 9.0:

- WebLogic JDriver for Oracle—Replaced by the WebLogic Type 4 JDBC Driver for Oracle. See "Supported Database Configurations" in *Supported Configurations for WebLogic Platform 9.0*.
- WebLogic JDriver for MS SQL Server—Replaced by the WebLogic Type 4 JDBC Driver for MS SQL Server. See "The MS SQL Server Driver" in WebLogic *Type 4 JDBC Drivers*.

The following features were deprecated in WebLogic Server 9.0:

- Data source factories and legacy application-scoped connection pools—Replaced by JDBC modules. See "Modular Configuration and Deployment of JDBC Resources" on page 33.
- Legacy driver-level logging—Replaced by DebugJDBCDriverLogging. See "JDBC Debugging Enhancements" on page 34.

- JDBCXADebugLevel of the JDBCConnectionPoolMBean—Replaced by the JTAJDBC debugging scope on the ServerDebugMBean. For more information, see *The WebLogic Server MBean Reference*.
- Legacy connection leak profiling, statement cache profiling, and statement usage profiling—Replaced by new JDBC resource profiling. See "New Monitoring Statistics and Profile Information for JDBC Resources" on page 33.
- JDBCConnectionPoolMBean, JDBCDataSourceMBean, JDBCTxDataSourceMBean, and JDBCMultiPoolMBean—Replaced by JDBCSystemResourceMBean. See *The WebLogic Server MBean Reference*.
- JDBCDataSourceFactoryMBean—No replacement.
- JDBCConnectionPoolRuntimeMBean—Replaced by JDBCDataSourceRuntimeMBean. See *The WebLogic Server MBean Reference*.
- `weblogic.jdbc.jts.driver`—Obsolete. Use a data source to get a database connection. See "Using WebLogic JDBC in an Application" in *Programming WebLogic JDBC*.
- `weblogic.jdbc.pool.driver`—Obsolete. Use a data source to get a database connection. See "Using WebLogic JDBC in an Application" in *Programming WebLogic JDBC*.
- `weblogic.jdbc.rmi.driver`—Obsolete. Use a data source to get a database connection. See "Using WebLogic JDBC in an Application" in *Programming WebLogic JDBC*.

Enterprise JavaBeans

WebLogic Server 9.0 complies with the EJB 2.1 specification and includes many EJB usability improvements, as described in the following sections.

For more information about all these features, see *Programming WebLogic Server EJBs*.

- "New Features to Support the EJB 2.1 Specification" on page 37
- "Message-Driven Bean Enhancements" on page 37
- "Improved EJB Caching and Pooling" on page 38
- "Automatic Retry of Rolled Back Transactions" on page 39
- "SQL Query Support" on page 39
- "Trimming String-Type Values" on page 40

- "Improved Operation Ordering for CMP Entities" on page 40

New Features to Support the EJB 2.1 Specification

The following new WebLogic Server features comply with the J2EE EJB 2.1 Specification.

EJB Timer Service for Modeling Business Processes

The WebLogic Server EJB Timer Service enables you to schedule a notification at a particular time, at the end of an elapsed period of time, or at recurring intervals.

EJB-QL Changes

The following Enterprise JavaBeans query language (EJB-QL) functions are new or enhanced in WebLogic Server 9.0:

- MOD arithmetic function (new)
- Aggregate functions (enhanced)
- Order By clause (enhanced)

Message Destination References

Logical message destinations can be declared in a deployment descriptor and mapped to an actual message destination, such as a JMS queue. EJBs can look up the message destination by performing a JNDI lookup with the logical message destination name.

Stateless Session Beans Exposed as Web Services

WebLogic Server 9.0 complies with EJB 2.1 requirements related to declaring and accessing external Web Services and exposing stateless session EJBs as Web Services. These features make it easier to develop and maintain EJBs that access Web Services. Both Java and non-Java clients can access stateless session beans as Web Services.

Message-Driven Bean Enhancements

Several enhancements make MDBs easier to use and support.

- MDBs can receive messages from JCA 1.5-compliant adapters, thus facilitating the integration of WebLogic Server applications with Enterprise Information Systems.

- A unique `client-id` for can be created for every instance of an MDB, making it easier to deploy durable MDBs to multiple servers instances in a WebLogic Server cluster.
- The EJB container suppresses repetitive exceptions that can occur as it repeatedly tries and fails to deliver a JMS message. The container prints out the exception and the number of times it has been thrown, rather than printing the exception message each time it occurs.
- You can automatically pause message processing for a configurable time period when MDB applications throw exceptions.
- You can administratively pause and resume message delivery without undeploying.
- Additional statistics are available, including last exception thrown by the application, which helps in diagnosing application failures.
- An API extension enables non-transactional applications to force message redelivery without also forcing the MDB instance to be destroyed and recreated.
- A performance extension supports the processing of multiple messages under a single transaction, rather than one message per transaction.
- Load-balancing capabilities are improved for MDB consumers from remote distributed destinations.

Improved EJB Caching and Pooling

Administrators can exercise more control over how EJBs are cached and pooled. For more information on all the following features, see *Programming WebLogic Server EJBs*.

Dynamic Entity Cache and EJB Pool to Match Demand

An administrator can configure the entity cache to release unused memory when demand increases, and to release pooled memory when demand decreases.

EJB Cache and Pool Initialization and Re-Initialization on Demand

This feature and the configurable option described in "Dynamic Entity Cache and EJB Pool to Match Demand" enable customers to take advance of the response time benefits that caching and pooling offer, without continuing to consume memory as load decreases.

Passivation During a Transaction

The EJB container can sustain a greater load before throwing a `CacheFullException`. In some circumstances, non-idle EJBs in cache can be passivated, so that a new request can be served.

Application developers can passivate non-idle EJB instances with the new `operationsComplete` method of `weblogic.ejb.interfaces.EJBLocalObject` This method indicates that all operations on that bean are complete for the current transaction. The container uses this information when evaluating beans for passivation.

Query Caching

Read-only entity EJBs can be cached at the query level and can be accessed in cache by any finder. This capability improves performance by avoiding database access.

Query caching is done implicitly by the EJB container; no application code is required. The behavior can be configured for a bean-level or application-level cache, using the `max-queries-in-cache` element in the `entity-cache` element of `weblogic-ejb-jar.xml`.

Concurrency Options for Optimistic Beans

Container-managed persistence (CMP) entity EJBs that use optimistic concurrency have new configuration options:

- Explicit invalidation—A developer can explicitly invalidate optimistic entity EJBs when the underlying data is updated by a program running outside the EJB container. Previously, this functionality was supported only for read-only entity EJBs.
- Time-to-live—Reduces the possibility that optimistic data will become out-dated.
- Database triggers—Maintain the version or timestamp value for versioned optimistic data. This new feature is useful in legacy environments for maintaining version information.

Automatic Retry of Rolled Back Transactions

Automatic retry of container-managed transactions can improve runtime performance in environments in which transactions are expected to roll back occasionally or regularly. Automatic transaction retry is supported for session and entity beans that use container-managed transaction demarcation.

SQL Query Support

In addition to EJB-QL queries, EJB developers can now specify SQL queries in `weblogic-cmp-jar.xml`.

Coding SQL queries is useful when the query logic cannot be expressed in EJB-QL, or if a database vendor-specific query is not supported by EJB-QL. Developers can use both EJB-QL and SQL in combination, taking advantage of EJB-QL portability benefits for most queries, and using SQL for those that cannot be accomplished with EJB-QL.

Trimming String-Type Values

The EJB container trims trailing blanks at the end of string type values in primary key and other container-managed persistence fields. Trimming string-type values in this fashion avoids comparison errors and portability issues that might otherwise arise.

Improved Operation Ordering for CMP Entities

The EJB container detects symmetric and circular relationships between container-managed persistence (CMP) entities when batching and ordering database operations.

In previous versions of WebLogic Server, batch database operations on entities in symmetrical or circular relationships could result in a foreign key constraint or non-null exception.

Web Applications, JSPs, and Servlets

The following sections describe new features in Web applications, JSPs, and servlets:

- "Web Applications" on page 40
- "Servlet 2.4 Implementation" on page 41
- "JSP 2.0 Implementation" on page 42
- "Deprecated and Obsolete Web Application Features" on page 43

Web Applications

- The `weblogic.xml` deployment descriptor is now schema-based rather than document type definition (DTD)-based. See "weblogic.xml Schema" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- You can configure virtual hosts for Web applications as channel-based or host-based. Channel-based virtual hosting is a new feature with this release. See "Creating and Configuring Web Applications" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- Several new backwards compatibility flags allow you to restore capabilities from releases prior to WebLogic Server 9.0. For a complete list and description of these flags, see Compatibility with Previous Releases in *Upgrading WebLogic Application Environments*.

- New Web application modules deployed as libraries are referenced from the weblogic.xml file using the same syntax that references application libraries in the weblogic-application.xml file, except that the `<context-root>` element is ignored. See "Creating and Configuring Web Applications" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- The `sharing-enabled` attribute has been added to the `session` element so that Web applications can share the same session. See "Using Sessions and Session Persistence" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server*.

Servlet 2.4 Implementation

WebLogic Server now supports the Servlet 2.4 Specification from Sun Microsystems, which is downloadable at `http://java.sun.com/products/servlet/download.html#specs`. See the Servlet 2.4 Specification for a complete list of all of the differences between the Servlet 2.4 Specification and earlier Specifications. The following changes to WebLogic Server 9.0 support the new specification:

- The Servlet 2.3 Specification from Sun Microsystems does not specify whether filters should be applied on forwards and includes. The Servlet 2.4 specification clarifies this by introducing a new `dispatcher` element in the `web.xml` deployment descriptor. Using this `dispatcher` element, you can configure a `filter-mapping` to be applied on `REQUEST/FORWARD/INCLUDE/ERROR`. In WebLogic Server 8.1, the default was `REQUEST+FORWARD+INCLUDE`. For the old DTD-based deployment descriptors, the default value is unchanged in order to preserve backward compatibility. For the new descriptors (schema-based), the default is `REQUEST`.

  The `<filter-dispatched-requests-enabled>` element controls whether filters are applied to dispatched requests. The default value is `false`. Because 2.4 servlets are backward compatible with 2.3 servlets (per the 2.4 specification), when 2.3 descriptor elements are detected by WebLogic Server, the `<filter-dispatched-requests-enabled>` element defaults to `true`. See "weblogic.xml Schema" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

  See "Servlet Programming Tasks" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- According to the Servlet 2.4 specification, the ServletContextListener is now called prior to servlet initialization. See "Templates for Event Listening Classes" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

- The `WEB-INF` directory is not part of the public document tree of the application. No file contained in the `WEB-INF` directory can be served directly to a client by the container. However, the contents of the `WEB-INF` directory are visible to servlet code using the `getResource` and `getResourceAsStream` method calls on the `ServletContext`, and may be exposed using the `RequestDispatcher` calls. See "Creating and Configuring Web Applications" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

- Welcome files can now be servlets, as well as JSPs and static pages. See "Creating and Configuring JSPs" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

- The new `disable-implicit-servlet-mappings` flag, when set to `true`, prevents the Web application container from creating implicit mappings for internal servlets. See "weblogic.xml Schema" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

- Two new servlet request events (`ServletRequestListener` and `ServletRequestAttributeListener`) can manage state across the life cycle of servlet requests and the methods invoked when the request events occur. See "Application Events and Event Listener Classes" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

JSP 2.0 Implementation

WebLogic Server now supports the JSP 2.0 Specification from Sun Microsystems. The following changes to WebLogic Server 9.0 support the new specification:

- Simplified creation of tag extensions—See *Programming WebLogic JSP Tag Extensions*.

- New `SimpleTag` tag handler API for JSPs—Provides a much simpler invocation protocol than do classic tag handlers. See "Implementing the Tag Handler" in *Programming WebLogic JSP Tag Extensions*.

- Extensible `DynamicAttributes` interface—Optionally, any tag handler can extend the `DynamicAttributes` interface to indicate that it supports dynamic attributes. In addition to implementing the `DynamicAttributes` interface, tag handlers that support dynamic attributes must declare that they do so in the Tag Library Descriptor (TLD). For more information on dynamic attributes, see "Creating a Tag Library Descriptor" in *Programming WebLogic JSP Tag Extensions*.

- New JSP expression language (EL) is inspired by both ECMAScript and the XPath expression languages—The EL is available in attribute values for standard and custom actions and within template text. In both cases, the EL is invoked consistently by way of the construct ${expr}. For more information on the JSP EL, see "WebLogic JSP Reference" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

- JSP property group—A collection of properties that apply to a set of files representing JSP pages. You define these properties in one or more subelements of the `jsp-property-group` element in the `web.xml` deployment descriptor. See "Creating and Configuring JSPs" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- Implicit includes—You can now configure implicit includes at the beginning and end of a JSP by using the `include-prelude` and `include-coda` elements of the `jsp-property-group` respectively. See "Creating and Configuring JSPs" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- New JSP document syntax support— JSPs are written in XML format. See "Creating and Configuring JSPs" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

Deprecated and Obsolete Web Application Features

The following Web Application features are deprecated in this release of WebLogic Server:

- The Servlet 2.4 Specification has deprecated `javax.servlet.SingleThreadModel` instance pools. WebLogic Server still supports `SingleThreadModel`; however, its use is not recommended. See "weblogic.xml Schema" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- The `<redirect-content-type>` element is obsolete in this release. See "weblogic.xml Schema" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*
- The `<redirect-content>` element is obsolete in this release. See "weblogic.xml Schema" in *Developing Web Applications, Servlets, and JSPs for WebLogic Server.*

Application Development

The following sections describe new features and functionality in application development for WebLogic Server:

- "J2EE Library Support for Easier Sharing of J2EE Modules" on page 44
- "Optional Package Support for Sharing JAR Files" on page 44
- "Context Propagation" on page 45

- "Split Development Directory Support for Deployment Plans" on page 45
- "New Features in Medical Records Sample Application" on page 45
- "Upgrading Deployment Descriptors from Previous Releases of J2EE and WebLogic Server" on page 45
- "Deprecated Startup and Shutdown Classes" on page 46

J2EE Library Support for Easier Sharing of J2EE Modules

J2EE library support in WebLogic Server 9.0 makes it easier to share J2EE modules among multiple applications. A J2EE library is a J2EE module or collection of modules packaged in an Enterprise application (EAR) that is first deployed and then registered with the J2EE application container. After a library is registered with the container, you can deploy Enterprise applications that reference the library. Each deployed Enterprise application that references a J2EE library can use the library modules as if they were packaged within that particular EAR.

You can, for example, deploy and register a single Web application module as a J2EE library that is used in multiple Enterprise applications in a domain. If the Web application requires updates, you can modify the code in a single place and redeploy the J2EE library. Referencing applications can then be redeployed, if necessary, to use the latest version of the Web application.

See Creating Application Libraries and Optional Packages in *Developing Applications for WebLogic Server* for information about creating libraries and referencing libraries in J2EE Applications. See Deploying Applications in *Deploying Applications to WebLogic Server* for information about registering libraries and deploying referencing applications.

Optional Package Support for Sharing JAR Files

WebLogic Server supports optional packages as described in the J2EE 1.4 Specification, Section 8.2 Optional Package Support, with versioning described in Optional Package Versioning. Optional packages allow you to share the contents of a JAR file with multiple J2EE modules deployed either as stand-alone modules or as part of an Enterprise application. For example, third-party Web application framework classes needed by multiple Web applications can be packaged and deployed in a single JAR file, and shared among multiple applications in the domain.

In general, you use J2EE library support when you need to share one or more J2EE modules among different Enterprise applications, and use optional packages when you need to share one or more JAR files among different J2EE modules. See Creating Application Libraries and

Optional Packages in *Developing Applications for WebLogic Server* and Deploying Applications in *Deploying Applications to WebLogic Server*.

Context Propagation

Context propagation allows programmers to associate information with an application that is then carried along with every request. Furthermore, downstream components can add or modify this information so that it can be carried back to the originator. Context propagation is also known as *work areas*, *work contexts*, or *application transactions*. See Programming Context Propagation.

Split Development Directory Support for Deployment Plans

The WebLogic split development directory environment and associated Ant tasks now support deployment plans and the new application installation root, as described in "Deploying Applications" on page 1-20.

The `wldeploy` command includes a new argument for specifying a deployment plan, and `appc` now performs deployment descriptor validation for applications that include a deployment plan.

New Features in Medical Records Sample Application

The Avitek Medical Records sample application now implements EJB entity-relationship caching, Log4j integration, custom diagnostics logging, and log access through the WebLogic Diagnostic Service, JDBC RowSets, Struts SSL, DBA authentication, XMLBeans, and a security realm extension that uses JMX.

For more information about how these features are implemented, see the New Features topic for the Avitek Medical Records Sample Application in the *WebLogic Server Code Examples* documentation. This documentation is provided as a collection of HTML files installed in the WebLogic Server installation directory at
`WL_HOME/samples/server/docs/core/index.html.`

Upgrading Deployment Descriptors from Previous Releases of J2EE and WebLogic Server

BEA recommends that you always upgrade deployment descriptors when you migrate applications to a new release of WebLogic Server so that your applications can take advantage of the features in the current J2EE specification and release of WebLogic Server. BEA provides the new `weblogic.DDConverter` tool for this purpose.

See Upgrading Deployment Descriptors From Previous Releases of J2EE and WebLogic Server.

Deprecated Startup and Shutdown Classes

WebLogic Server startup and shutdown classes are deprecated in favor of applications that respond to application life cycle events. See Programming Application Lifecycle Events in *Developing Applications with WebLogic Server*.

Application Deployment

The following sections describe new application deployment functionality for WebLogic Server:

- "WebLogic Deployment API Implements and Extends JSR-88" on page 46
- "Application Configurations Deployable to Multiple Domains" on page 47
- "New Directory Structure for Easier Production Deployment" on page 47
- "Production Redeployment and Maintaining Availability" on page 48
- "Version Designation to Support Production Redeployment" on page 49
- "Final Checking in Production with No Disruptions to Clients" on page 49
- "JMS and JDBC Deployable Resource Configuration" on page 49
- "Deployment Targets in WebLogic Server 9.0" on page 50
- "New Deployment Configuration Tools" on page 51
- "Deployment to Multiple Environments Without Modifying Descriptor Files" on page 51
- "Deprecated Deployment Features" on page 52

WebLogic Deployment API Implements and Extends JSR-88

In J2EE 1.4, the JSR-88 specification defines a standard API application configuration and deployment to a target application server environment. WebLogic Server 9.0 implements the JSR-88 Service Provider Interface (SPI) plug-in and model plug-in to comply with the J2EE 1.4 deployment specification. You can use a basic JSR-88 deployment tool with the WebLogic Server plug-ins (with no extensions) to configure, deploy, and redeploy J2EE applications and modules to WebLogic Server.

The new WebLogic Server deployment API implements and extends JSR-88 to support the advanced deployment capabilities described in the sections that follow. All WebLogic Server deployment tools, including the Administration Console, `weblogic.Deployer`,`wldeploy` Ant task, and the WebLogic Scripting Tool, are used in conjunction with the deployment API to configure, deploy, and redeploy applications in a domain. You can use the deployment API to build your own WebLogic Server deployment tools, or to integrate WebLogic Server configuration and deployment operations with an existing JSR-88-compliant tool.

See the J2EE v1.4 Documentation for more information about the JSR-88 deployment specification. For information about extensions to JSR-88 deployment, see Introduction to WebLogic Server Deployment in *Deploying WebLogic Server Applications* and Programming WebLogic Deployment in *WebLogic Server 9.0 API Reference*.

For information about configuring applications for deployment, see Configuring Applications for Production Deployment in *Deploying WebLogic Server Applications*.

Application Configurations Deployable to Multiple Domains

The basic JSR-88 configuration process does not support migration of an application configuration from one environment to another within an organization. WebLogic Server extends JSR-88 with a command-line tool, `weblogic.PlanGenerator`, whereby you can export an application configuration into a deployment plan for deployment to multiple domains.

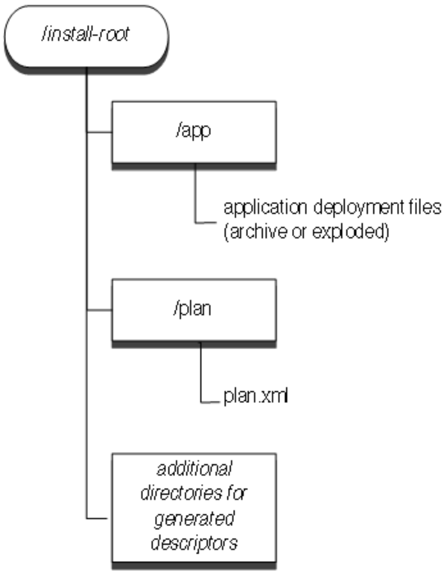
You select WebLogic Server deployment descriptor properties that you know will need to change, such as global resource names and tuning parameters, before the application is deployed into another environment. The export process creates variable definitions in the deployment plan to act as placeholders for the selected descriptor properties. Through the Administration Console, a deployer can easily assign values to those descriptor properties for different server domains, without changing the actual deployment descriptors files in the application itself.

See Exporting an Application for Deployment to New Environments in *Deploying Applications to WebLogic Server* and weblogic.PlanGenerator Command Line Reference in *Deploying Applications to WebLogic Server*.

New Directory Structure for Easier Production Deployment

A new application directory structure separates generated configuration files from core application files, so that configuration files can be easily changed or replaced without disturbing the application itself. Figure 1-1 shows the directory hierarchy for storing a deployable application or module.

Figure 1 Application Installation Root


/install-root
/app
application deployment files
(archive or exploded)
/plan
plan.xml
additional directories for generated descriptors

You can easily deploy applications by specifying only the installation root. Also, the Administration Console automatically creates the directory structure shown in Figure 1 for deployments that are staged on target servers, so that deployment configuration files are available in a well-known location. BEA recommends this directory structure for all production deployments.

Production Redeployment and Maintaining Availability

You can redeploy a revised version of a production application alongside the older version of the application without affecting existing clients to the application, and without interrupting the availability of the application to new client requests. WebLogic Server automatically manages client connections so that existing clients continue to use the older application, while new client requests are directed to the newer application. The older version is undeployed after all current clients complete their work, the Administrator explicitly undeploys the older version, or a configured timeout is reached. A rollback capability allows you to stop the redeployment process if, for example, problems are detected in the newer application version.

Production redeployment can also be used with Administration mode (see "Final Checking in Production with No Disruptions to Clients" on page 49) to isolate the new application in the production environment for testing purposes.

See Redeploying Applications in a Production Environment in *Deploying Applications to Weblogic Server*.

Version Designation to Support Production Redeployment

To support the production redeployment strategy, WebLogic Server now recognizes a unique version string entry in an Enterprise Application `MANIFEST.MF` file. When a redeployment operation is requested, WebLogic Server checks the version string to determine whether to deploy a new version of the application.

See Redeploying Applications in a Production Environment in *Deploying Applications to Weblogic Server*.

Final Checking in Production with No Disruptions to Clients

Using Administration mode, you can deploy or redeploy an application into a production environment while isolating the application from external client connections. Administration mode allows access to the application only through a configured Administration channel. You can perform a final check of the application directly in the production environment without disrupting clients.

See "Production Redeployment and Maintaining Availability" on page -48 and Using Production Redeployment to Update Applications in *Deploying Applications to WebLogic Server*.

JMS and JDBC Deployable Resource Configuration

JMS and JDBC configurations in WebLogic Server 9.0 are now stored in XML documents that conform to the appropriate WebLogic Server schema for the resource: `weblogic-jmsmd.xsd` or `weblogic-jdbc.xsd`. You create and manage JMS and JDBC resources either as system modules, similar to the way they were managed prior to 9.0, or as application modules, similar to standard J2EE modules.

A JMS or JDBC application module can be deployed as a stand-alone resource, in which case the resource is available to the servers or cluster targeted during the deployment process, or as part of an Enterprise application. An application module deployed as part of an Enterprise Application is available only to the enclosing application (an *application-scoped resource*). Using application-scoped resources ensures that an application always has access to required resources, and simplifies the process of deploying the application into new environments.

In contrast to system modules, application modules are owned by the developer who created and packaged the module, rather than the Administrator who deploys the module. This means that the Administrator has limited control over JDBC and JMS application modules. When deploying an application module, an Administrator can change resource properties that were specified in the module, but cannot add or delete resources. System modules are created by the Administrator through the WebLogic Server Administration Console, and can be changed or deleted as necessary by the Administrator.

See:

- *Configuring and Managing WebLogic JMS*
- *Configuring and Managing WebLogic JDBC*
- Deploying Applications in *Deploying Applications to WebLogic Server*

Deployment Targets in WebLogic Server 9.0

WebLogic Server 9.0 introduces a new JMS server deployment target that you can select when deploying JMS application modules to a domain. Table 1 describes all valid deployment targets and lists the types of modules that you can deploy to them.

Table 1 WebLogic Server 9.0 Deployment Targets

Target Type	Description	Valid Deployments
WebLogic Server	A single WebLogic Server instance, such as an Administration Server or a Managed Server	J2EE Applications J2EE modules JMS or JDBC application modules J2EE Libraries and optional packages
Cluster	A configured cluster of multiple WebLogic Server instances	J2EE Applications J2EE modules JMS or JDBC application modules J2EE Libraries and optional packages

Target Type	Description	Valid Deployments
Virtual host	A configured host name that routes requests for a particular DNS name to a WebLogic Server instance or cluster	Web Applications
JMS server	A JMS server configured in a Weblogic Server domain	A JMS queue, topic, or connection factory defined within a JMS application module

New Deployment Configuration Tools

WebLogic Server includes these new deployment tools:

- `weblogic.PlanGenerator`—Generates WebLogic Server deployment plans and deployment descriptors as necessary to configure an application. It also exports deployment descriptor properties to deployment plan variables using the Formal Classification for Deployment Descriptor Properties. See the weblogic.PlanGenerator Command Line Reference in *Deploying Applications to WebLogic Server*.
- WebLogic Scripting Tool (WLST)— Provides commands for configuring applications and automated deployment, as well as other WebLogic Server administration tasks. See "Automation of Domain Configuration Tasks with WLST" on page 8.

Deployment to Multiple Environments Without Modifying Descriptor Files

Prior to WebLogic Server 9.0, in the Administration Console you could dynamically edit selected deployment descriptors for an application that was deployed as an exploded archive directory. Changes to the deployment configuration were persisted directly to the application's WebLogic Server deployment descriptor files.

The Administration Console in WebLogic Server 9.0 enables you to edit the WebLogic Server deployment configuration for any deployed application or module, regardless of whether the application was deployed as an archive file or as an exploded archive directory. Changes to the deployment configuration are now persisted to a WebLogic Server deployment plan, leaving the original deployment descriptors untouched. Using a deployment plan preserves the original deployment files and makes use of WebLogic Server extensions to the JSR-88 deployment specification introduced in J2EE 1.4.

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

Deprecated Deployment Features

The following deployment features are deprecated or unsupported:

- WebLogic Server 6.x single-phase deployment protocol—Deprecated in versions 7.x and 8.x, and unsupported in WebLogic Server 9.0. All deployments now use the two-phase deployment protocol.
- The `weblogic.management.runtime.DeployerRuntimeMBean` API —Deprecated. Use the new `weblogic.deploy.api` packages for all application configuration and deployment tasks.
- The use of alternate deployment descriptors for deploying applications modules—Instead of alternate deployment descriptors, use JSR-88-style deployment configuration and deployment plans, to maintain custom configurations outside of a packaged application.
- The use of the `weblogic.Deployer -redeploy` *`module-uri`* syntax for redeploying a single module in a deployment—Deprecated. Instead, use production redeployment or use the `-redeploy -targets` *`module@target`* syntax.

XML

WebLogic Server 9.0 implements the following new XML standards:

- “StAX Implementation” on page 53
- “JAX-P 1.2 Implementation” on page 53
- “JAX-R 1.0 Implementation” on page 53

See “Deprecated XML Features” on page 53 for information about the deprecated XML features in this release.

See “Parsing XML Documents in a Servlet No Longer a Default” on page 53 for information about the changed XML feature in this release.

StAX Implementation

The *Streaming API for XML* (StAX) is a Java Community Process specification that describes a bi-directional API for reading and writing XML. StAX gives parsing control to the programmer by exposing a simple iteration-based API and an underlying stream of events.

See Using the Streaming API for XML at {DOCROOT}/xml/stax.html.

JAX-P 1.2 Implementation

Version 1.2 of the *Java API for XML Processing* (JAX-P) specification includes an XSLT framework based on TrAX (Transformation API for XML), plus some updates to the parsing API to support DOM Level 2 and SAX version 2.0 and an improved schema to locate pluggable implementations. JAXP provides support for XML schema and an XSLT compiler (XSLTC).

See Developing XML Applications with WebLogic Server at {DOCROOT}/xml/programming.html.

JAX-R 1.0 Implementation

Version 1.0 of the *Java API for XML Registries* (JAX-R) specification documents a standard means of accessing different kinds of XML registries.

See Using the Java API for XML Registries at {DOCROOT}/xml/api.html#jaxr.

Parsing XML Documents in a Servlet No Longer a Default

Parsing XML documents from within a servlet using the `setAttribute` and `getAttribute` methods no longer works by default. You must now first configure a WebLogic Server servlet filter to use this feature.

See Parsing XML Documents in a Servlet at {DOCROOT}/xml/programming.html#apps005.

Deprecated XML Features

The following WebLogic XML features are deprecated in this release.

WebLogic XML Streaming API

Although the WebLogic XML Streaming API is still accessible in this release of WebLogic Server, its functionality has been deprecated. Programmers should use StAX instead.

See Using the Streaming API for XML at {DOCROOT}/xml/stax.html.

Default XML Parser Based on Apache Xerces

The default parser in Versions 8.1and previous of WebLogic Server was one that was `based` on Apache's Xerces parser and whose package name started with `weblogic.apache.xerces.*`. In Version 9.0 of WebLogic Server, this parser has been deprecated. Instead, the default parser is the one that is shipped in JDK 5.0.

See Difference In Default Parsers Between Versions 8.1 and 9.0 of WebLogic Server at {DOCROOT}/xml/overview.html#deprecated_xerces.

Installation Changes

Note the following changes to WebLogic Server installation:

- The WebLogic Web Server plug-ins are no longer installed by default. In 9.0, you must choose Custom Installation to install the Web Server plug-ins. The installation path has changed for 9.0. Table 2 lists the installation paths for plug-ins.

Table 2 Installation Paths for WebLogic Web Server Plug-ins

Operating System	Shared Object Location
AIX	`WL_HOME/weblogic90/server/plugin/aix/ppc`
HPUX11	`WL_HOME/weblogic90/server/plugin/hpux11/IPF64` `WL_HOME/weblogic90/server/plugin/hpux11/PA_RISC`

Linux	`WL_HOME/weblogic90/server/plugin/linux/i686`
	`WL_HOME/weblogic90/server/plugin/linux/ia64`
	`WL_HOME/weblogic90/server/plugin/linux/s390`

Solaris	`WL_HOME/weblogic90/server/plugin/solaris/sparc`
Windows (Apache 2.0 only)	`WL_HOME\weblogic90\server\plugin\win\32` or `WL_HOME\weblogic90\server\plugin\win\64`

- Prior to 9.0, only one instance of Node Manager could be installed on a machine. In 9.0, Node Manager is installed as a Windows service by default, and it is registered with the product installation directory in which you are installing the software; for example, `C:_bea_weblogic90b`. Node Manager can then be used to manage communication with all domains associated with that installation directory.

Note: If you do not want Node Manager installed, perform a custom installation, and deselect Node Manager Service in the **Install Windows Service** window.

See the *Installation Guide*.

WebLogic Upgrade Wizard

The WebLogic Upgrade Wizard enables you to upgrade your application environment from a WebLogic Server 6.1, 7.0, or 8.1 release to WebLogic Server 9.0. Specifically, you can use the WebLogic Upgrade Wizard to upgrade:

- WebLogic Server 6.1, 7.0, or 8.1 domains to run in a WebLogic Server 9.0 environment.
- Node Manager, if it is used in your current environment to provide high-availability to Managed Servers.
- Custom security providers.

See *Upgrading WebLogic Application Environments*.

Java Management Extensions (JMX)

Release 9.0 introduces several important changes to the WebLogic Server JMX implementation:

- "JMX 1.2 and JMX Remote API (JSR-160)" on page 56
- "Revised Model for Distributing Domain Configuration Data" on page 56
- "Changes to the MBean Data Model" on page 56
- "New, Functionally Aligned MBean Servers" on page 56
- "New Reference Document for Public WebLogic Server MBeans" on page 57
- "Facilities for Registering Custom MBeans" on page 57
- "New and Deprecated MBeans and Interfaces" on page 57

For more information about these features, see "New and Changed JMX Features in This Release" in *Developing Custom Management Utilities with JMX*.

JMX 1.2 and JMX Remote API (JSR-160)

The WebLogic Server implementation of Java Management Extensions (JMX) is upgraded from 1.0 to 1.2. JMX 1.2 includes a new group of public APIs that enable JMX components to communicate across JVMs (JSR-160).

Now that the JMX remote APIs are published, BEA's proprietary API for remote JMX access, `weblogic.management.MBeanHome`, is no longer needed and is therefore deprecated.

Revised Model for Distributing Domain Configuration Data

All changes to a domain configuration now follow a process that resembles a transaction. The Administration Server hosts a group of edit MBeans, which are the in-memory representation of all pending changes to a domain’s configuration. Changes in the edit MBeans must be explicitly distributed to server instances in a domain. If any server is unable to consume a change, the entire set of changes in a distribution process is rolled back.

See “Predictable Distribution of Domain Configuration Changes” on page 7.

Changes to the MBean Data Model

The JMX specification does not impose a model for organizing MBeans. However, because the configuration of a WebLogic Server domain is specified in an XML document, WebLogic Server organizes its MBeans into a hierarchical model that reflects the XML document structure.

For example, the root of a domain's configuration document is `<domain>` and below the root are child elements such as `<server>` and `<cluster>`. Each domain maintains a single MBean of type DomainMBean to represent the `<domain>` root element. Within DomainMBean, JMX attributes provide access to the MBeans that represent child elements such as `<server>` and `<cluster>`.

New, Functionally Aligned MBean Servers

An Administration Server maintains three MBean servers, each of which provides access to different MBean hierarchies. The Edit MBean Server provides access to the domain's editable configuration MBeans; the Domain Runtime MBean Server provides federated access to all runtime MBeans and read-only configuration MBeans in the domain; and the Runtime MBean Server provides access only to the runtime and read-only configuration MBeans on the Administration Server.

Each Managed Server maintains a Runtime MBean Server, which provides access only to its runtime and read-only configuration MBeans.

JMX clients use the standard `javax.remote.access` APIs to access and interact with MBeans registered in the MBean servers.

New Reference Document for Public WebLogic Server MBeans

All public WebLogic Server MBeans are described in a new document, WebLogic Server MBean Reference. For each MBean, the document describes:

- MBean factory methods and other points of access within WebLogic Server MBean trees.
- Data type, read-write privileges, and other information for each attribute.
- Parameters, signature, and other information for each operation.

Facilities for Registering Custom MBeans

Prior to 9.0, if you wanted to register custom MBeans in an MBean server on a WebLogic Server instance, you could either create your own MBean server or use `weblogic.management.RemoteMBeanServer` to register in WebLogic Server's MBean server.

As of 9.0 and JDK 1.5, you can do any of the following from a JMX client that is running in a WebLogic Server JVM:

- (Recommended) Access the WebLogic Runtime MBean Server through JNDI and register custom MBeans in the Runtime MBean Server.
- Register custom MBeans in the JVM platform MBean server.
- Create your own MBean server.

New and Deprecated MBeans and Interfaces

Many subsystems, such as logging, JMS, JDBC, and deployment, have deprecated all or part of their old JMX interfaces and replaced them with new or updated MBeans.

For details, see WebLogic Server MBean Reference.

Also, prior to 9.0, applications could access type-safe stub interfaces for WebLogic Server MBeans through the `weblogic.management.MBeanHome` interface. As of 9.0, the `MBeanHome` interface is deprecated. Instead of using this typed API layer, all JMX applications should use the standard JMX programming model. The WebLogic Server 9.0 API Reference no longer describes the `MBeanHome` interface or any of the MBeans that can be accessed through it. Instead, this deprecated access is described in a separate reference document, Type-Safe Access to WebLogic Server 9.0 MBeans.

If any of your classes import the type-safe interfaces (which are under `weblogic.management`), BEA recommends that you update to the standard JMX programming model.

J2EE Management APIs

The J2EE Management APIs enable a software developer to create a single Java program that can discover and browse resources, such as JDBC connection pools and deployed applications, on any J2EE Web application server. The APIs are part of the J2EE Management Specification, which requires all J2EE Web application servers to describe their resources in a standard data model.

WebLogic Server 9.0 implements the required features of the J2EE Management Specification (JSR-077), version 1.0.

See Monitoring and Managing with the J2EE Management APIs.

Security

The following sections describe new functionality in the WebLogic Security Service. See Understanding WebLogic Security for detailed information.

- “Support for Java Authorization Contract for Containers (JACC)” on page 58
- “Single Sign-On Capabilities” on page 59
- “Support for Certificate Lookup and Validation” on page 59
- “SSL Features” on page 59
- “New WebLogic Security Providers” on page 60
- “Enhancements to WebLogic Security Providers” on page 61
- “Enhancements to the Security Service Programming Interfaces (SSPIs)” on page 62

Support for Java Authorization Contract for Containers (JACC)

The Java Authorization Contract for Containers (JACC) Standard can replace the EJB and Servlet container deployment and authorization provided by WebLogic Server.

When JACC is configured for use in a WebLogic Server domain, EJB and servlet authorization decisions are made by the classes in the JACC framework. All other authorization decisions within WebLogic Server are still determined by the WebLogic Security Service.

Single Sign-On Capabilities

Single sign-on (SSO) requires a user to sign on to an application only once to access many different application components, even though these components may have their own authentication schemes. This release of WebLogic Server supports SSO with Web browsers and HTTP clients through the use of the Security Assertion Markup Language (SAML) and supports Windows desktops with the Simple and Protected Negotiate (SPNEGO) protocol.

SAML Support

WebLogic Server includes a SAML Inter-site Transfer Service (ITS), an Assertion Consumer Service (ACS), and an Assertion Retrieval Service (ARS) that support the SAML POST and Artifact profiles. The SAML capabilities in WebLogic Server allow single sign-on (SSO) between WebLogic domains as well as between WebLogic Server and other vendors' SAML-capable servers or between applications in a single WebLogic domain. WebLogic Server uses a SAML Credential Mapping and a SAML Identity Assertion provider to generate and consume the assertions used by the SSO profiles. This release of WebLogic Server supports SAML 1.1.

WebLogic Web Services supports the SAML Token profile, both as a Web Services client and a Web Services server.

For more information about SAML, see http://www.oasis-open.org.

Support for Certificate Lookup and Validation

The WebLogic Security Service finds and validates X509 certificate chains for inbound two-way SSL, outbound SSL, application code, and WebLogic Web Services. The security framework extends and completes the JDK CertPath functionality.

SSL Features

The following SSL features have been added:

- SSL configuration options for network channels — You can specify identity certificates, private key information, and one- and two-way SSL options for individual channels. In previous releases, network channels used the SSL attributes defined for the SSL port of the server.

- Dynamic SSL attributes for the server—Changes made to the SSL attributes for a particular server through the WebLogic Server Administration Console now take effect without rebooting the server.
- Restarts for all SSL server channels through the Administration Console—SSL server channels can now be restarted through the WebLogic Server Administration Console even when changes were not made through the Console.

New WebLogic Security Providers

The following sections describe the new security providers available in this release. For more detailed information, see Understanding WebLogic Security.

Authentication Providers

- Database Base Management System (DBMS) providers — Access user, password, group, and group membership information in databases for authentication purposes and can be used to upgrade from the RDBMS security realm. These providers include SQL Authentication, Read-only SQL Authentication, and Custom DBMS Authentication.
- Windows NT provider— Enable the use of Windows NT users and groups for authentication purposes.

Identity Assertion Providers

- New LDAP X509 Identity Assertion —Receives an X509 certificate; looks up the LDAP object for the associated user; ensures that the certificate in the LDAP object matches the presented certificate; and retrieves the user name from the LDAP object for authentication.
- Negotiate Identity Assertion —Decodes Simple and Protected Negotiate (SPNEGO) tokens to provide SSO with the Windows desktop by obtaining Kerberos tokens; validates the Kerberos tokens; and maps Kerberos tokens to WebLogic users.

SAML Providers

- SAML Credential Mapping —Generates SAML 1.1 assertions for authenticated subjects based on a target site or resource. If the requested target has not been configured and no defaults are set, an assertion is not generated. User information and group membership (if configured as such) are put in the AttributeStatement.

- SAML Identity Assertion —Validates SAML 1.1 assertions and verifies the issuer is trusted. If so, identity is asserted based on the AuthenticationStatement contained in the assertion.

Both providers support the following SAML subject confirmation methods:

- `artifact`
- `bearer`
- `sender-vouches`
- `holder-of-key`

Certificate Lookup and Validation Providers

- WebLogic CertPath —Completes certificate paths and validates certificates by using the trusted CA configured for a particular server instance. It also checks signatures in the chain, ensures that the chain has not expired, and checks that one certificate in the chain is issued by one of the trusted CAs configured for the server. If any checks fail, the chain is not valid.
- Certificate Registry —Supports the Certificate lookup and validation framework. The registry allows the system administrator to explicitly configure a list of trusted CA certificates that are allowed access to the server. The Certificate Registry provides an inexpensive mechanism for performing revocation checking. An administrator revokes a certificate by removing it from the certificate registry. The registry is stored in the embedded LDAP server.

Enhancements to WebLogic Security Providers

The following enhancements have been made to the WebLogic security providers:

- You can adjust the performance characteristics of the WebLogic Authentication provider, LDAP Authentication providers, and DBMS providers by configuring options that yield a faster group lookup response; optimize group membership caches; expose the internal PrincipalValidator cache; and increase its thresholds.
- The WebLogic Auditing provider can audit data for many types of context data. A set of supported context data (such as HTTP servlet requests or EJB parameters) is defined. The WebLogic Auditing provider also supports a new management interface, `weblogic.management.security.audit.ContextHandler`, which indicates whether the provider supports auditing context elements. Custom auditing providers can also implement this interface.

- The WebLogic Authentication provider now supports password digests. WebLogic Web Services use this functionality to support the username token profile of a Web Service, including the password digest.
- The WebLogic Authorization provider supports new default security predicates for accessing HTTP Servlet requests, HTTP Session attributes, and any element passed to the ContextHandler. In addition, new Date and Time predicates are available.

Enhancements to the Security Service Programming Interfaces (SSPIs)

The following enhancements were made to the SSPIs:

- Context handler —Contains additional context and container-specific information from the resource container, and provides that information to the security provider making the access or role mapping decision. Context handler support is now available for the following methods:
  - `Adjudicator.adjudicate()`
  - `LoginModule.login()`
  - `IdentityAsserter.assertIdentity()`
  - `AuditAtnEvent()`
  - `CredentialMapper.getCredentials()`
- Servlet authentication filters —Perform pre-processing and post-processing for identity assertion and authentication functions. You can use the filters to encapsulate recurring tasks in reusable units and transform the response from a servlet or JSP page.

WebLogic Tuxedo Connector

WebLogic Tuxedo Connector is enhanced as follows:

- Dynamic Domain Administration utility —This utility enables WebLogic Tuxedo Connector users to start and stop individual connections between WebLogic Server applications and remote Tuxedo domains. You can shut down individual connections to remote Tuxedo domains without affecting communication between a WebLogic Server application and other Tuxedo domains. You can then modify the configuration for the domains and re-establish the connection to implement the new configuration, without affecting other connected domains.

- MBSTRING support and codeset encoding updates —WebLogic Tuxedo Connector now supports the new Tuxedo MBSTRING buffer type as well as the MBSTRING field in FML32. MBSTRING support is provided for the conversion of the codesets that the underlying JVM can handle. This feature also provides aliasing capability to support mapping of the various names for a given encoding that are used with different platforms and standards.
- Link-level failover support —WebLogic Tuxedo Connector now supports link-level failover by specifying the correct failover sequence information in the comma separated syntax `<nw-addr>` XML tag in the `WTCRemoteTuxDomMBean` and `WTCLocalTuxDomMBean` definitions. The order of the network addresses determines the order of preference for failover.

  The semantic of the link-level failover is late binding, which means the existence and availability of the address are not checked when the MBean is created. This condition allows users to add the machine to DNS *after* the WTC configuration is created, but *before* the TDomain session connection is created.
- `The weblogic.wtc.applicationQueueSize` property is no longer supported. WTC now uses a workmanager to handle requests.

Examples

All EJB examples and other examples that use EJBs are now modified to use EJBGen for EJB 2.0 code and descriptor generation.

WebLogic Server now ships with a DBMS pluggable runtime authentication provider that replaces the custom authentication provider used by the Medical Records sample application.

To download additional WebLogic Server 9.0 code examples that are not included in the WebLogic Server 9.0 kit, see WebLogic Server 9 projects located at `https://wls9.projects.dev2dev.bea.com/`.

Supported Configurations

For the latest information about supported WebLogic Server configurations, see *Supported Configurations*.

Standards Support

The following tables list the standards supported in WebLogic Server 9.0:

- Table 3, “Java Standards,” on page -64

- Table 4, "Web Services Standards," on page -65
- Table 5, "Other Standards," on page -65

Table 3 Java Standards

Standard	Version
J2EE	1.4, 1.3
JDKs	5.0 (aka 1.5), 1.4 (clients only)
J2EE Enterprise Web Services	1.1
J2EE EJB	2.1, 2.0, and 1.1
J2EE JMS	1.1, 1.0.2b
J2EE JDBC (with third-party drivers)	3.0
MS SQL jDriver	1.0
Oracle OCI jDriver	1.0 and some 2.0 features (batching)
J2EE JNDI	1.2
OTS/JTA	1.2 and 1.0.1b
J2EE Servlet	2.4, 2.3, and 2.2
J2EE JSP	2.0, 1.2, and 1.1
RMI/IIOP	1.0
JMX	1.2, 1.0
JavaMail	1.2
JAAS	1.0 Full
J2EE CA	1.5, 1.0
JCE	1.4
Java RMI	1.0
JAX-P	1.2, 1.1

Table 3 Java Standards

JAX-RPC	1.1, 1.0
JAX-R	1.0
SOAP Attachments for Java (SAAJ)	1.2

Table 4 Web Services Standards

Standard	Version
Enterprise Web Services	1.1
SOAP	1.1
WSDL	1.1
UDDI	2.0
WS-Security	1.0

Table 5 Other Standards

Standard	Version
SSL	v3
X.509	v3
LDAP	v3
TLS	v1
HTTP	1.1

Third-Party JAR Files

Table 6 lists third-party JAR files included with WebLogic Server.

Table 6 Third-Party JAR Files

JAR File	Description
J2EE, javax, and subdirectories	• `com/sun/activation/` • `com/sun/mail` • `corba idl` • `javax/activation` • `javax/connector` • `javax/ejb` • `javax/jms` • `javax/jts` • `javax/mail` • `javax/management` • `javax/net` • `javax/servlet` • `javax/transaction` • `javax/xml/messaging` • `javax/xml/soap` • `javax/xml/rpc` • `jta` • `jts`

JAR File	Description
Libraries	• Ant 1.6.2 • Antlr 2.7.1 (MageLang Institute) • Cert-J 2.0.2 from certicom • Certicom SSL 3.1.14 • RSA Crypto-J 3.5 • Netscape LDAP 3.1 • Oracle Thin JDBC driver • AdventNet SNMP 3.2 SP1 • JavaScript 1.5 from Mozilla • JCom from J-Integra • PointBase 4.3 • Octetstring 1.5
XML	• Acumen UDDI • Apache Xerces DOM

Deprecated APIs

For a list of deprecated WebLogic Server classes, see http://e-docs.bea.com/wls/docs90/javadocs/deprecated-list.html.

Deprecated Command-Line Properties

The command-line property, `-Dweblogic.ProductionModeEnabled={true | false}`, is deprecated in WebLogic Server 9.0. It determined whether a server started in production mode. For information on how to change to production mode, see Change to production mode in the *Administration Console Online Help*.

Documentation Updates

The documentation for WebLogic Server has been updated for the 9.0 release.

New Books

The following books were added to the documentation set for WebLogic Server 9.0:

- *Programming Web Services for WebLogic Server* (documents completely new 9.0 Web Services runtime and programming model)
- *Upgrading WebLogic Application Environments*
- *Understanding the WebLogic Diagnostic Framework*
- *Configuring and Using the WebLogic Diagnostic Framework*
- *Extending the Administration Console*
- *Programming WebLogic Deployment*
- *Programming Stand-alone Clients*
- *WebLogic C API*
- *Beehive Integration in WebLogic Server*
- *Configuring WebLogic Server Environments*
- *Managing Server Startup and Shutdown*
- *Understanding Domain Configuration*
- *Configuring and Managing WebLogic Store-and-Forward*
- *Configuring and Managing the WebLogic Messaging Bridge*
- *Understanding the WebLogic Diagnostic Service*
- *WebLogic Scripting Tool*
- *Monitoring and Managing with the J2EE Management APIs*
- *WebLogic Server MBean Reference*
- *Developing Custom Management Utilities with JMX*
- *Developing Manageable Applications with JMX*
- *Configuring Log Files and Filtering Log Messages*
- *Using WebLogic Logging Services for Application Logging*
- *Accessibility Notes for the WebLogic Server Administration Console*
- *Developing Web Applications, Servlets, and JSPs for WebLogic Server*

Reorganized Information

The information that previously constituted *Configuring and Managing WebLogic Server* has been reorganized into the following books, some of which are new to the 9.0 release:

- *Configuring WebLogic Server Environments*
- *Managing Server Startup and Shutdown*
- *Understanding Domain Configuration*
- *Configuring and Managing WebLogic JDBC*
- *Configuring and Managing WebLogic JMS*
- *Configuring and Managing WebLogic Store-and-Forward*
- *Configuring and Managing the WebLogic Messaging Bridge*
- *Monitoring and Managing with the J2EE Management APIs*
- *WebLogic Server MBean Reference*

Retired Documentation

The following documentation no longer exists as of the WebLogic Server 9.0 release.

- *Using Applets with WebLogic Server*
- *Programming WebLogic Server for Wireless Services*

RMI/IIOP Documentation

Much of the IIOP documentation has been consolidated into *Programming WebLogic RMI*. The IIOP client information has been moved to *Programming Stand-alone Clients*.

Documentation Name Change

WebLogic Server Partner's Guide is now *ISV Partners Guide*.