Oracle® Application Server 10*g*

Migrating From WebLogic

10*g* Release 3 (10.1.3)

B16027-01

January 2006

ORACLE®

Oracle Application Server 10*g* Migrating From WebLogic, 10*g* Release 3 (10.1.3)

B16027-01

Copyright © 2006, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

3 Migrating Java Servlets

4 Migrating JSP Pages

5 Migrating Enterprise JavaBean Components

6 Migrating JDBC

A Additional Features

Index

Preface

This manual provides the information required to migrate applications from WebLogic Server to Oracle Application Server 10*g* Release 3 (10.1.3). See the release notes for platform-specific details and any late-breaking information.

Intended Audience

Oracle Application Server 10g Migrating From WebLogic is intended for administrators, developers, and others whose role is to deploy and manage Oracle Application Server with high availability requirements.

Documentation Accessibility

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

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Where to Find More Information

The following documents in the Oracle Application Server 10*g* Release 3 (10.1.3) Documentation Library provide further reading to the information in this book:

- *Oracle Application Server Release Notes*, 10*g* Release 3 (10.1.3), which contains a chapter for each component of Oracle Application Server.
- *Oracle Containers for J2EE Services Guide*
- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*
- *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide*
- *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*
- *Oracle Containers for J2EE Servlet Developer's Guide*

Related Manuals

Oracle Application Server Migrating from WebLogic is one of several migration guides. The others are:

- *Oracle Application Server Migrating from WebSphere*
- *Oracle Application Server Migrating from JBoss*

Documentation Formats

Documentation for Oracle Application Server Migrating from WebLogic is provided in PDF and HTML formats.

To view the PDF files, you will need

- Adobe Acrobat Reader 3.0 or later, which you can download from `http://www.adobe.com`.

To view the HTML files, you will need

- Netscape 4.x or later, or
- Internet Explorer 4.x or later

Online Help

The OracleAS Personalization Administrative UI includes extensive online help that can be summoned from a list of contents and from Help buttons.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Overview

This chapter provides an overview of the issues involved in migrating J2EE Web applications from WebLogic Server to Oracle Application Server 10*g* Release 3 (10.1.3), and the effort required.

The chapter contains these topics:

1.1 Overview of Oracle Application Server 10*g*

Oracle Application Server 10*g* is a comprehensive and integrated standards-based application platform suite that provides all of the infrastructure and functionality required to run an agile business. It offers a number of technology solutions based on service-oriented architecture (SOA):

- A J2EE-based SOA platform to develop, deploy, and manage Web services.
- Enterprise integration services for data integration, business process automation, and business activity monitoring.

These solutions share common infrastructures for security, systems management, and grid computing, which allows flexible and scalable deployment of applications on low-cost, modular servers and storage.

Oracle Application Server's SOA infrastructure provides out-of-the-box facilities to develop, wrap, orchestrate, provision, manage, secure, federate, discover, and access enterprise applications and legacy systems as services. All these facilities are available through a single-installed, integrated product.

The following are the main features of Oracle Application Server:

- Oracle Containers for J2EE (OC4J) - a fully J2EE 1.4 compatible container implementation (also backward compatible to J2EE 1.3)
- Oracle Process Manager and Notification Server (OPMN) - a configurable and distributed process manager
- Oracle HTTP Server with mod_oc4j plug-in
- Oracle Enterprise Manager 10*g* Application Server Control
- JMX management infrastructure
- Web services runtime and management based on JAX-RPC

- Java Object - XML mapping
- Oracle TopLink object - relational mapping with CMP support

For a comparison of features between Oracle Application Server and WebLogic Server, see Chapter 2.

1.2 Migration Highlights

In quantifying the migration effort, it is helpful to examine the application components to be migrated with the following issues in mind:

- Portability

  Code may not be portable because it contains embedded references to vendor-specific extensions to the J2EE specification. Evaluating and planning for code modifications may be a significant part of the migration effort.

- Proprietary extensions

  If vendor-specific extensions are in use, migration of those components becomes difficult or unfeasible. Complete redesign toward J2EE specifications is not addressed in this document. If vendor-specific extensions are in use, they may need to be redesigned and reimplemented, rather than being identified as migration candidates.

- Deviations from J2EE specifications.

  If a component is largely non-compliant with the J2EE specifications, this guide will not be helpful in determining the migration path to Oracle Application Server. If the J2EE specification version of the component is not of version 1.4 (the version on which this guide is based), then the specification implementation differences will need to be addressed.

1.2.1 Migration Approach

The approach in developing this migration guide was to document our experience migrating J2EE application components from WebLogic Server to Oracle Application Server. Examples shipped with WebLogic Server were selected, tested on WebLogic Server, and migrated to Oracle Application Server. Issues encountered in the migration of these examples are the basis for this document.

For most migration projects, a J2EE application can be migrated in the following order:

1. Identify the following differences between WebLogic Server and Oracle Application Server: deployment, runtime (classloaders), third party library support.
2. Remove (if possible) platform specific proprietary features. For example, for WebLogic Server: BEA JOLT, JSP-based Tags.
3. Port platform-specific deployment descriptors.
4. Port a J2EE application in tiers in the following sequence:
   a. Data tier (EJBs).
   b. Web tier.
   c. Java clients.
   d. Data sources, JMS message queues, and/or JCA adaptors.
   e. Other components (for example, single sign-on, JAAS, LDAP repository).

Note:

- JNDI names can be different. Use the JNDI tree browser in Application Server Control Console or Oracle JDeveloper to browse the JNDI tree in OC4J.
- After migrating the above tiers and components, the recommended deployment approach is to deploy your application to a standalone OC4J instance to validate its full runtime functionality. When the validation is complete, your application can be deployed using Application Server Control to a single node or multi-node cluster of Oracle Application Server installations.

5. Implement clustering and performance tune OC4J J2EE container.

1.2.2 J2EE Application Migration Challenges

The varying degrees of compliance to J2EE standards can make migrating applications from one application server to another a daunting task. Some of the challenges in migrating J2EE applications from one application server to another are:

- Though in theory, any J2EE application can be deployed on any J2EE-compliant application server, in practice, this is not strictly true.
- Lack of knowledge of the implementation details of the given J2EE application.
- Ambiguity in the meaning of 'J2EE-compliant' (usually, this means the application server has J2EE compliant features, not code-level compatibility with the J2EE specification).
- The number of vendor-supplied extensions to the J2EE standards in use, which differ in deployment methods and reduce portability of Java code from one application server to another.
- Differences in clustering, load balancing, and fail over implementations among application servers; these differences are sparsely documented, and are thus an even bigger challenge to the migration process.

This guide addresses the abovementioned challenges in migrating your applications from WebLogic Server to Oracle Application Server. It provides an approach to migration with solutions based on the J2EE version 1.4 specification.

1.2.3 Migration Effort

Moving from WebLogic Server to Oracle Application Server is a relatively simple process. Standard J2EE applications, using no proprietary APIs, can be deployed with no required code changes. The only actions required are configuration and deployment. Those applications using proprietary utilities or APIs can be ported easily.

1.2.4 Migration Tool

The Oracle JDeveloper Application Migration Assistant (AMA) is a new tool developed by Oracle to simplify the process of migrating applications to the Oracle platform. The tool provides code navigation and progress reporting to guide you through migrating from WebLogic Server to Oracle Application Server 10*g*.

The AMA tool is installed as a plug-in to Oracle JDeveloper. It uses regular expressions to identify code in your application files that may require modification to work on the Oracle platform. These regular expressions are contained in an XML file called search rules file. AMA can analyze your WebLogic Server application and generate an analysis report that summarizes project statistics, allows navigation between review items, and provides comprehensive status tracking for your migration changes. AMA is customizable by providing an extensible API that allows additional search rules files to be written and tailored for your specific application.

Oracle provides a number of search rules files through the AMA Search Rules Exchange (*http://www.oracle.com/technology/tech/migration/ama/exchange/exchange.html*). One of these files is the AMA Search Rules for BEA WebLogic Migrations. The rules defined in this file can be used to identify WebLogic-specific code that may require modification for migration to Oracle Application Server. For example, the tool can identify whether you are using Jolt or BEA JCOM. It can also locate any references in your application to Defweblogic startup/shutdown, T3 services, WebLogic XA, and native WebLogic JDBC drivers.

To download the tool and for more information on it, go to *http://www.oracle.com/technology/tech/migration/ama/*.

1.3 Using This Guide

This guide details the migration of components from WebLogic Server to Oracle Application Server. While it does not claim to be an exhaustive source of solutions for every possible configuration, it provides solutions for some of the migration issues listed above, which will surface, along with others, in your migration effort. The information in this guide helps you to assess the WebLogic Server applications and plan and execute their migration to Oracle Application Server. The material in this guide supports these high-level tasks:

- Survey the components according to the issues listed above
- Identify migration candidates
- Prepare the migration environment and tools
- Migrate and test the candidate components

2

Comparison of Oracle Application Server and WebLogic Server

Although WebLogic Server and Oracle Application Server are both J2EE servers that support J2EE 1.3 or 1.4 features (depending on WebLogic Server version), both application servers have intrinsic differences ranging from product packaging to runtime architecture. This chapter seeks to discuss these differences and is organized as follows:

2.1 Architectures

This section describes and compares the overall architectures of WebLogic Server and Oracle Application Server.

2.1.1 Specifications Levels Supported

The following table provides the specific level of support for J2EE standards provided by Oracle Application Server and WebLogic Server:

Table 2–1 J2EE Support in Oracle Application Server and WebLogic Server

Product	J2EE 1.3	J2EE 1.4
Oracle Application Server version	9.0.4	10.1.3[1]
	10.1.2	
WebLogic Server version	7.0	9.0 Beta[2]
	8.1	

[1] Backward compatible to J2EE 1.3

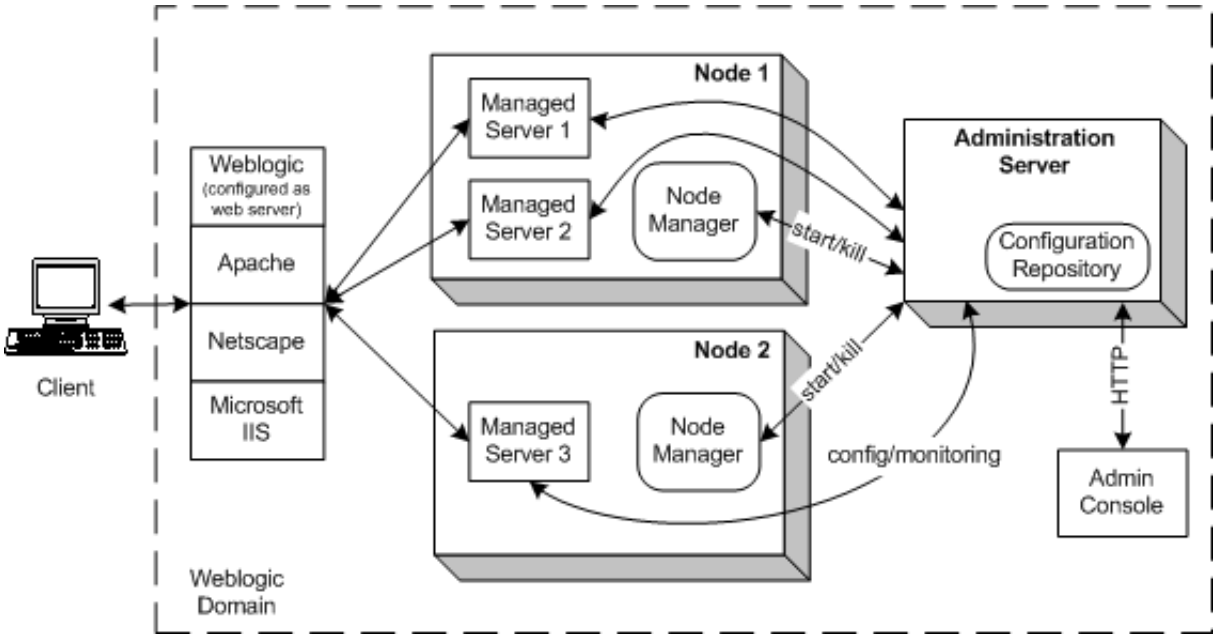
[2] At the time of this writing (November 2005)

The discussion in this guide focuses on Oracle Application Server 10*g* Release 3 (10.1.3) and WebLogic Server 7.0 and 8.1.

2.1.2 WebLogic Server

WebLogic Server has several components and concepts peculiar to it. Each WebLogic Server can be configured and deployed either as a Managed Server or an Administration Server. A Managed Server hosts and executes the application logic deployed in it when requests are received from clients. An Administration Server configures and monitors Managed Servers but does not host applications. Figure 2–1 depicts the components in WebLogic Server and their interactions.

Figure 2–1 WebLogic Server Components



In any node, more than one Managed Server can exist. Each Managed Server is a Java process (JVM) executing J2EE containers (Web and EJB). An Administration Server, which is also a Java process, is required to propagate configuration information to Managed Servers when they start-up. The configuration information is stored in the file system on the Administration Server node.

The Administration Server is also used to monitor and log information about individual Managed Servers and the entire WebLogic domain. A WebLogic domain can consist of standalone Managed Servers, clusters of Managed Servers, and one Administration Server. If the Administration Server goes offline, client requests can still be serviced by the Managed Servers. However, configuration information is not available for new Managed Servers to start-up, and monitoring services are not available for server clusters. The Administration Server does not have automatic fail over or replication. Configuration data for the WebLogic domain has to be manually backed up. The Administration Server functions can be accessed through a console GUI (remotely over HTTP) or a command line utility.

In order for the Administration Server to start Managed Servers remotely, a Node Manager must be running on each node where there are Managed Servers. This Node Manager is a Java program executing in the background as a UNIX daemon or Windows service. With the Node Manager, the Administration Server can also kill a Managed Server if the latter hangs or does not respond to commands from the former.

WebLogic Server can also be set up to run as a Web server. In this mode, it supports HTTP 1.1 and resolves client requests to Managed Servers based on the settings in the

XML configuration files. Instead of WebLogic Server, third-party proxy plug-ins can also be used for servicing HTTP requests. Supported plug-ins are Apache, Netscape, and Microsoft IIS.

> **Note:** For a discussion on WebLogic Server clustering, see Section 2.3.1, "WebLogic Server Support for High Availability and Load Balancing"

2.1.3 Oracle Application Server Components and Concepts

This section describes components and several concepts peculiar to Oracle Application Server. The discussion here provides an overview scope.

See Also:

Oracle Application Server Concepts,

Oracle Application Server Administrator's Guide,

Oracle Application Server High Availability Guide

Oracle Application Server Containers for J2EE User's Guide.

2.1.3.1 J2EE in Oracle Application Server

The J2EE server implementation in Oracle Application Server is called Oracle Application Server Containers for J2EE (OC4J). OC4J is a pure Java implementation. It runs on the standard JDK and is extremely lightweight with a small foot print. OC4J provides high performance and scalability, and is simple to deploy and manage. With Oracle Application Server 10*g* Release 3 (10.1.3), OC4J supports J2EE 1.4 APIs. Figure 2–3 provides an architectural view of the J2EE component layers in Oracle Application Server.

This migration guide seeks to help you understand the migration challenges you may face when migrating your J2EE applications from WebLogic Server to Oracle Application Server 10*g* Release 3 (10.1.3).

> **Note:** In this document, where WebLogic Server is mentioned without a version number, WebLogic Server 7.0 and 8.1 are implied. Otherwise, a version number is explicitly mentioned.

2.1.3.2 Oracle Application Server Instance

An OracleAS instance is a runtime occurrence of an installation of Oracle Application Server. An Oracle Application Server installation has a corresponding Oracle home where the Oracle Application Server files are installed. Each Oracle Application Server installation can provide only one OracleAS instance at runtime. A physical node can have multiple Oracle homes, and hence, more than one Oracle Application Server installation and OracleAS instance.

Each OracleAS instance consists of several interoperating components that enable Oracle Application Server to service user requests in a reliable and scalable manner. These components are:

- Oracle HTTP Server
- OC4J Instances
- Oracle Process Management Notification (OPMN) Server

- Oracle Enterprise Manager 10g Application Server Control Console

2.1.3.3 Oracle HTTP Server

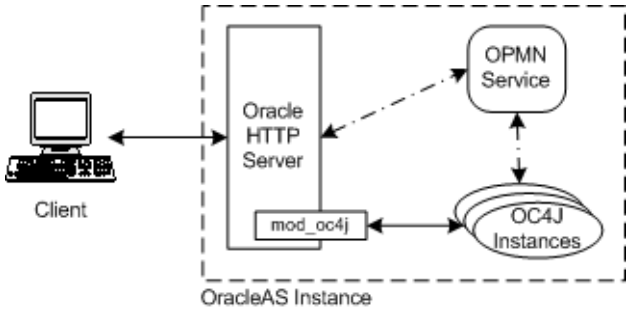
Oracle Application Server contains two listeners: Oracle HTTP Server (based on the Apache open source project) and the listener that is part of OC4J, which runs in a separate thread of execution. Each OracleAS instance has one Oracle HTTP Server.

The OC4J listener listens to requests coming from the mod_oc4j module of the Oracle HTTP Server and forwards them to the appropriate OC4J process. From a functional viewpoint, the Oracle HTTP Server acts as a proxy server to OC4J, wherein all servlet or JSP requests are redirected to OC4J processes.

mod_oc4j communicates with the OC4J listener using the Apache JServ Protocol version 1.3 (AJP 1.3). mod_oc4j works with the Oracle HTTP Server as an Apache module. The OC4J listener can also accept HTTP and RMI requests, in addition to AJP 1.3 requests.

The following diagram depicts Oracle HTTP Server and other Oracle Application Server runtime components in a single instance of Oracle Application Server.

Figure 2–2 Components of an OracleAS Instance

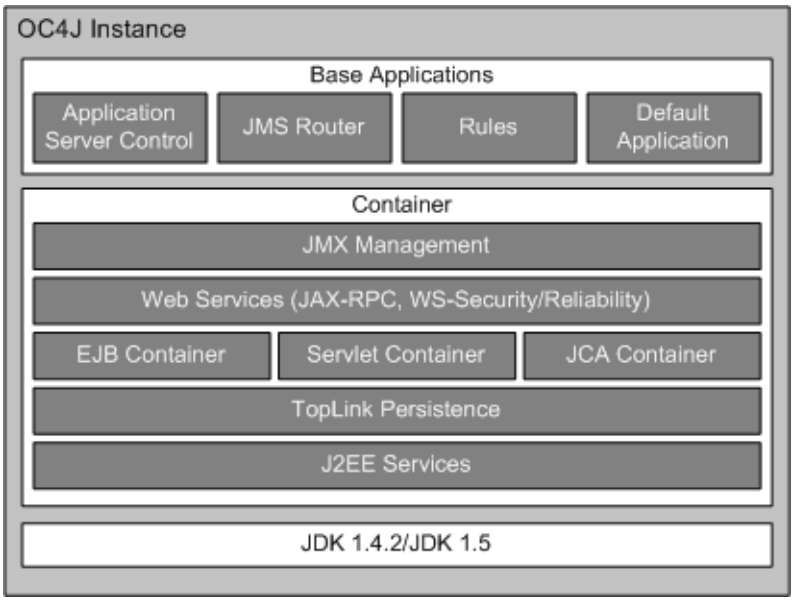


2.1.3.4 OC4J Instances

An OC4J instance is a logical instantiation of the OC4J implementation in Oracle Application Server. This implementation is Java 2 Enterprise Edition (J2EE) 1.4 complete and written entirely in Java. It executes on the standard Java Development Kit (JDK) 1.4.2 and 5.0 Java Virtual Machine (JDK 5.0 is installed with Oracle Application Server). It has a lower disk and memory footprint than other Java application servers. Note that each OC4J instance can consist of more than one JVM process where each process can be executing multiple J2EE containers. The number of JVM processes can be specified for each OC4J instance using the Oracle Enterprise Manager 10*g* Application Server Control Console GUI.

OC4J can be installed and run in standalone configuration. In this configuration, the OC4J listener directly accepts client requests as Oracle HTTP Server is not installed. Another component that is not installed is the Oracle Process Manager and Notification Server, which performs process monitoring and management functions. Hence, in standalone configurations, OC4J is unmanaged.

Figure 2–3 J2EE Architecture in Oracle Application Server



The logical architecture of an OC4J instance, as depicted in Figure 2–3, consists of the following components:

- JMX

  The Application Server Control Console user interface is built on a JMX-compliant client that can be used to completely manage and monitor an OC4J instance. The JMX functionality provided through Application Server Control Console is enabled through Java components known as MBeans. An MBean, or managed bean, is a Java object that represents a JMX manageable resource. MBeans are defined in the J2EE Management Specification (JSR-77), which is part of the J2EE 1.4 specification as published by Sun Microsystems. For more information on MBeans, see *Oracle Application Server Containers for J2EE Configuration and Administration Guide* and *Oracle Application Server Containers for J2EE Services Guide*.

- Web Services

  OC4J provides full support for Web Services in accordance with the J2EE 1.4 standard, including JAX-RPC 1.1 (WebLogic Server 8.1 supports up to JAX-RPC 1.0). Web services interoperability is also supported. See *Oracle Application Server Web Services Developer's Guide*.

- EJB, Servlet, JCA containers

  OC4J implements the full J2EE 1.4 container functionality for EJB, servlet, and JCA. It provides clustering features to enable container components for availability and scalability.

- Oracle Application Server TopLink persistence layer

  OracleAS TopLink provides a Java object-to-relational mapping persistence architecture. It provides a highly flexible and productive mechanism for storing Java objects and EJBs in relational database tables enabling developers to focus on pure object-oriented design and methodologies. In this release of Oracle Application Server, EJB container-managed persistence is integrated with OracleAS TopLink. OracleAS TopLink container-managed persistence is the default container-managed persistence provider. Rich user interface tools in OracleAS TopLink Mapping Workbench and Oracle JDeveloper 10*g* allow

developers to configure object-relational mapping quickly and easily. For existing container-managed EJBs, a migration utility is provided to automate migration to OracleAS TopLink container-managed persistence. See Section A.3, "Oracle TopLink", *Oracle Application Server TopLink Getting Started Guide* for more information on OracleAS TopLink.

Oracle Application Server allows several OC4J instances to be clustered together as part of an Oracle Application Server Cluster for scalability and high availability purposes. When OC4J instances are clustered together, they have a consistent configuration and the same applications deployed throughout the instances. A more in-depth discussion on clustering is found in Section 2.3.2, "Oracle Application Server Support for High Availability and Load Balancing".

2.1.3.5 Oracle Process Management Notification (OPMN) Server

Each OracleAS instance has an OPMN component, which performs monitoring and process management functions of processes in that instant. OPMN manages the processes in an OracleAS instance to enable startup, death-detection, and recovery of processes.

OPMN components in separate OracleAS instances can also send messages to each other to enable each OPMN to be aware of the components in the other OracleAS instances. When OPMN performs such a role, the OracleAS instances are managed by OPMN together as a cluster. Such a cluster is called an Oracle Application Server Cluster (OracleAS Cluster). (See Section 2.3.2, "Oracle Application Server Support for High Availability and Load Balancing" for more details on clusters.)

OPMN also communicates and interfaces with Application Server Control Console to provide a consolidated interface for monitoring, configurating, and managing Oracle Application Server. Some administrative tasks can also be accomplished using an OPMN command line utility. Oracle HTTP Server, and OC4J instances use a subscribe-publish messaging mechanism to communicate with the OPMN service. For process-level fail over and availability, the process that implements the OPMN service has a shadow process that restarts the OPMN process if it fails.

> **See Also:** *Oracle Process Manager and Notification Server Administrator's Guide*

2.1.3.6 Oracle Enterprise Manager 10*g* Application Server Control Console

Oracle Enterprise Manager 10*g* Application Server Control Console (Application Server Control Console) provides a Web-based interface for managing Oracle Application Server components and applications. It is deployed by default as an application in OC4J. Using the Application Server Control Console, you can do the following:

- monitor OracleAS components, OracleAS instances, OracleAS clusters, Oracle HTTP Server, and deployed J2EE applications and their components
- configure Oracle Application Server components, instances, clusters, and deployed applications
- operate OracleAS components, instances, clusters, and deployed applications
- manage security for OracleAS components and deployed applications
- provide performance metrics for OC4J instances and applications

For more information on Oracle Enterprise Manager and its two frameworks, see *Oracle Enterprise Manager Concepts*.

See Also: *Oracle Application Server Administrator's Guide* - provides descriptions on Application Server Control Console and instructions on how to use it.

2.2 Web Services

The following table provide the specific levels of support for Web services standards provided by Oracle Application Server 10*g* Release 3 (10.1.3) and WebLogic Server 7.0 and 8.1:

Table 2–2 Web Services Standards Supported

	Oracle Application Server 10*g* Release 3 (10.1.3)	**WebLogic Server 7.0**	**WebLogic Server 8.1**
WS-I Basic Profile	1.0, 1.1	Pre 1.0	1.0
WSDL	1.1	1.1	1.1
SOAP	1.1/1.2	1.1/1.2 incoming 1.1 outgoing	1.1/1.2
WS-Reliability	1.0	Not Supported	Not Supported
WS-Security	Following from version 1.0 are supported: ■ XML Signature ■ XML Encryption ■ Username Token ■ X.509 Token ■ SAML Token	Not Supported	Following from version one 1.0 are supported: ■ SOAP Message Security ■ Username Token Profile ■ X.509 Token Profile
UDDI	2.0	2.0	2.0

In addition, consider the following:

- Oracle Application Server supports Apache Axis. If a WebLogic Server application is running Axis, the same application should be deployable in Oracle Application Server provided no proprietary modifications were made to the standard Axis library.
- WebLogic Web services are packaged as standard J2EE enterprise applications; deploying a Web service is the same as deploying an enterprise application.
- Optional JAX-RPC datatype is support by both Oracle Application Server and WebLogic Server.
- WebLogic Server uses ANT `autotype` and `servicegen` tasks. These present some limitations. `autotype` does not comply with the JAX-RPC specification if the data type is a `complexType`, the `complexType` contains a single sequence, or the sequence contains a single element with `maxOccurs` greater than 1 or is unbounded. The JAX-RPC specification, in turn, states that this type of XML Schema data type should map to a Java array with a pair of setter and getter methods in a JavaBean class. WebLogic Web services do not adhere to this last part of the specification.
- REST style Web services, which uses XML documents instead of SOAP envelopes, is supported by Oracle Application Server.

See Also: Section A.1, "Migrating Web Services"

2.3 High Availability and Load balancing

This section describes high availability and load balancing and their importance to application server operation. It compares the methodologies for each in WebLogic Server and Oracle Application Server.

2.3.1 WebLogic Server Support for High Availability and Load Balancing

One or more WebLogic Servers can be grouped together as a cluster. Applications can be deployed commonly in all servers in a cluster, through cluster-wide deployment, to allow client requests to be load balanced across the cluster and the applications to have fail over capabilities. In a WebLogic cluster, the entities that benefit from clustering are HTTP session states, and EJB and RMI objects. Several load balancing algorithms are used by WebLogic. These include round-robin, weight-based, and parameter-based.

Note: The discussion in the following sections provides an overview of the high availability and load balancing mechanisms used by WebLogic Server. For deeper details, see *http://www.weblogic.com*.

2.3.1.1 HTTP Session State Load Balancing and Fail Over

Clients making requests to a WebLogic cluster can have their requests load balanced across the servers in the cluster. For this to work, a Web server installed with the WebLogic proxy plug-in or a hardware load balancer must be used. The WebLogic proxy plug-in uses a round-robin load balancing mechanism to distribute the request load. If a hardware load balancer is used, the cluster can be load balanced using the hardware's mechanism.

WebLogic Server achieves fail over for servlets and JSPs by replicating the HTTP session states of clients. When a WebLogic Server receives the very first request for a servlet or JSP, it replicates the servlet's session state to another server. The replicated session state is always kept up-to-date with the original. The WebLogic proxy plug-in returns the names of the two servers to the client through a cookie or by rewriting the URL. If the server hosting the original session state fails, the WebLogic proxy plug-in uses the information in the cookie or URL to redirect the client to the server with the replicated session state. At any one time, the cluster maintains an original and replica of each active session state. In this scenario, the session state is replicated in memory. WebLogic Server also supports replication to the file system or a database through JDBC, however, the fail over is not automatic for these replication methods.

2.3.1.2 EJB and RMI Object Load Balancing and Fail Over

WebLogic Server provides load balancing and fail over for EJB and RMI objects by using a JNDI service and client stubs which are both cluster-aware.

Each WebLogic Server that is a member of a cluster maintains a local JNDI tree. This tree contains information on objects deployed on the local server and around the cluster (for objects that are clusterable). If a clusterable object is deployed on more than one server, each JNDI tree reflects the existence of that object on those servers. When a clusterable object is deployed on a server, that server, through multicast, notifies the other servers in the cluster of the new deployment. The other servers' update their JNDI trees accordingly. Note that the server with the deployed object also sends the object's stub to the other servers.

When a client looks up a clusterable object in the JNDI service, the server servicing the request returns a stub of the object to the client. This stub contains information about

which server(s) the object is actually deployed in. The stub also has load balancing logic to balance method calls to the object. The load balancing algorithms available are round-robin, weight-based, random, and parameter-based. From the client's point-of-view, the cluster is transparent. The JNDI look ups and load balancing are done without the client knowing that it is working with a clustered object at the server end.

In the case where a clustered object is stateful, for example, a stateful session EJB, the object's state is replicated to a second server. The replication is achieved in a similar manner as for HTTP session state. The server that is chosen to service a client's very first request replicates the object's state to another server. The client stub is updated to note this second server. If the first server fails, the stub receives an exception when it tries to invoke a method. The stub then redirects the invocation to the server with the replicated object state. This server instantiates the object with the replicated state and executes the method invocation. This server also selects another server to replicate the state to since the original server has failed. Fail over of stateful objects is achieved this way.

Fail over of stateless objects is more straightforward to achieve as state need not be replicated. Upon receiving an exception indicating that a server has failed, the client stub simply selects another server which is hosting another instance of the called object and redirects the method invocation there.

2.3.2 Oracle Application Server Support for High Availability and Load Balancing

The Oracle Application Server architecture supports high availability for Oracle Application Server instances that in many cases can prevent unplanned down time for deployed applications. In general, Oracle Application Server achieves high availability through clustering and process monitoring. Clustering ensures that fail over, load balancing, and scalability are achieved for deployed J2EE applications. Additionally, monitoring of individual processes in a cluster ensures that processes are reliable. The following sections discuss how the benefits of clustering and process monitoring are attained by Oracle Application Server:

- Section 2.3.2.1, "Process Monitoring"
- Section 2.3.2.2, "Session State Replication"
- Section 2.3.2.3, "Load Balancing"
- Section 2.3.2.4, "Java Object Cache"

See Also:

- *Oracle Application Server Containers for J2EE Configuration and Administration Guide*
- *Oracle Application Server High Availability Guide*

2.3.2.1 Process Monitoring

Processes in an Oracle Application Server instance are monitored by OPMN. The OPMN system provides for process death detection and process restarting in the event that component processes in an Oracle Application Server instance "hang" or "crash." Monitored components include Oracle HTTP Server and OC4J processes. Refer to Section 2.1.3.5, "Oracle Process Management Notification (OPMN) Server" for more information on OPMN.

OPMN provides the functionality for managing clustered Oracle Application Server components. The OPMN process in an Oracle Application Server instance can be

configured to be aware of the availability of Oracle HTTP Server and OC4J processes in other Oracle Application Server instances. This allows OPMN to have a real-time picture of the health of all component processes in a cluster.

2.3.2.2 Session State Replication

High availability of HTTP session state and stateful session EJBs is provided by application clustering. This type of clustering enables fail over and redundancy at the application level. For applications deployed in OC4J, objects and values in HTTP sessions and stateful session EJBs can be replicated to OC4J instances in different server nodes. These OC4J instances host the same applications. If one node fails, requests to a clustered application on that node can be directed to the a surviving node hosting that clustered application. The fail over of the application is transparent to the client. The replication of session state can be made in one of two ways:

- **In memory**

  When replication of session state is made in memory, the participating nodes can communicate in one of two ways: multicast or peer-to-peer. For multicast, session state information is multicast over a common address and port. The address and port information is specified in an application's deployment configuration file. All applications deployed with the same multicast address and port have their state information replicated to other nodes with the same application and multicast address and port specified. The number of nodes involved in the replication can be explicitly specified so that multicast traffic can be minimized.

  For peer-to-peer, the session state information is unicast from one peer node to another. The participating nodes can be dynamically discovered or statically defined (OC4J standalone deployments only). For dynamic discovery, a node registers with OPMN to discover other peers and to add itself to the list of peers. State information is then replicated to each peer in the list.

- **In a database**

  Session state information can be persisted into a database. The database's JNDI datasource name is specified in an application's deployment configuration file. The replicated information is stored in three database tables, which are created the first time the database replication is invoked. The session data is stored for the duration of the session's time-to-live. Provided the database itself is protected from failures and has a backup and recovery scheme in place, persisting state information in the database allows for recovery of the information in cases where all processes and nodes participating in the cluster have failed.

> **Note:** All session objects must be serializable if they are to be persisted into memory or a database.

2.3.2.3 Load Balancing

Within each cluster, there is no mechanism to load balance or fail over the OracleAS instances. That is, there is no internal mechanism in the cluster to load balance or fail over requests to the Oracle HTTP Server component in the instances. A separate load balancer such as OracleAS Web Cache or hardware load balancing product can be used to load balance the OracleAS instances in the cluster and fail over the Oracle HTTP Server instances in the cluster.

Smart Routing – Oracle Application Server Web Cache and Oracle HTTP Server (`mod_oc4j`) provide configurable and intelligent routing for incoming requests. Requests are routed only to processes and components that `mod_oc4J` determines to be alive,

through communication with the Oracle Process Manager and Notification Server system. This smart routing mechanism load balances requests to J2EE applications deployed in clustered OC4J instances.

2.3.2.4 Java Object Cache

Oracle Application Server Java Object Cache provides a distributed cache that can serve as a high availability solution for applications deployed to OC4J. The Java Object Cache is an in-process cache of Java objects that can be used on any Java platform by any Java application. It allows applications to share objects across requests and across users, and coordinates the life cycle of the objects across processes.

Java Object Cache enables data replication among OC4J processes even if they do not belong to the same OC4J cluster, application server instance, or Oracle Application Server Cluster.

By using Java Object Cache, performance can be improved since shared Java objects are cached locally, regardless of which application produces the objects. This also improves availability; in the event that the source for an object becomes unavailable, the locally cached version will still be available.

2.4 Java Development and Deployment Tools

This section compares the Java tools offered by the WebLogic Platform and Oracle Application Server.

2.4.1 WebLogic Development and Deployment Tools

The WebLogic development environment and Administration Console are described below.

2.4.1.1 WebLogic Server Workshop

WebLogic Workshop is a visual development environment for building and deploying Web services using Java and XML. Workshop provides a framework and set of controls to interact with EJBs, databases, JMS topics and queues, and other Web services and applications. Several of these controls are proprietary to the WebLogic Platform, in addition to the Java Web Services (JWS) file definition. A JWS file contains the logic to implement a Web service on WebLogic Server. However, JWS is not a J2EE or Web services standard and is not portable to other application services.

2.4.1.2 WebLogic Server Administration Console

The WebLogic Server administrative console provides a GUI for managing the WebLogic Server domain. A WebLogic Server domain consists of one or more WebLogic Server instances (where each instance runs one or more applications) or clusters of instances. The administrative console connects to the designated administrative server running in the domain and can be used to change the configuration or run-time state on any machine in a domain. The administrative console is used to define clusters, add servers, deploy applications, configure applications, and manage Web servers, services, and resources in the domain.

2.4.2 Oracle Application Server Development and Deployment Tools

This section describes development and deployment tools for creating J2EE applications. The tools are part of the Oracle Developer Suite.

2.4.2.1 Development Tools

Application developers can use the tools in Oracle JDeveloper to build J2EE-compliant applications for deployment on OC4J. JDeveloper is a component in Oracle Internet Developer Suite, a full-featured, integrated development environment for creating multitier Java applications. It enables you to develop, debug, and deploy Java client applications, dynamic HTML applications, Web and application server components and database stored procedures based on industry-standard models. For creating multitier Java applications, JDeveloper has the following features:

- Web application development
- Java client application development
- Java in the database
- Component-Based Development with JavaBeans
- Simplified database access
- Visual Integrated Development Environment
- Complete J2EE 1.4 support
- Automatic generation of `.ear` files, `.war` files, `ejb-jar.xml` file, and deployment descriptors.

You can build applications with Oracle JDeveloper and deploy them manually, using Application Server Control Console, or with the OC4J Administration Console. Also note that you are not restricted to using JDeveloper to build applications; you can deploy applications built with IBM VisualAge or Borland JBuilder on OC4J.

Note: In addition to JDeveloper, Oracle Application Server TopLink, an object-relational mapping tool, also comes with Oracle Application Server. See *Oracle Application Server TopLink Application Developer's Guide*.

Note: The Oracle JDeveloper Application Migration Assistant, a migration tool that plugs into Oracle JDeveloper, can be used to quickly identify application code that needs to be migrated. See the section "Migration Tool" on page 1-6 for more information and where to download the tool.

2.4.2.2 Assembly Tools

Oracle Application Server provides a number of assembly tools to configure and package J2EE Applications. The output from these tools is compliant with J2EE standards and is not specific to OC4J. These include:

- A WAR file assembly tool to assemble JSP, servlets, tag libraries and static content into WAR files.
- An EJB assembler, which packages an EJB home, remote interface, deployment descriptor, and the EJB into a standard JAR file.
- An EAR file assembly tool, which assembles WAR Files and EJB JARs into standard EAR files.
- A tag library assembly tool, which assembles JSP tag libraries into standard JAR files.

2.4.2.3 Administration Tools

Oracle Application Server also provides two different administration facilities to configure, monitor, and administer its components.

- A graphical management tool, Oracle Enterprise Manager 10*g* Application Server Control Console, which provides a single point of administration across OracleAS Clusters, Farms, and OC4J containers.
- A command line tool for performing administrative tasks locally or remotely from a command prompt. (Application Server Control Console is the preferred administration environment over this command line tool as it provides a more integrated set of administration services.)

3

Migrating Java Servlets

This chapter provides the information you need to migrate Java servlets from WebLogic Server to Oracle Application Server. It covers the migration of simple servlets, WAR files, and exploded Web applications.

This chapter contains these topics:

3.1 Introduction

Migrating Java servlets from WebLogic Server to Oracle Application Server is straightforward, requiring little or no code changes to the servlets migrated.

Oracle Application Server 10*g* Release 3 (10.1.3) is fully compliant with Sun Microsystem's J2EE Servlet specification, version 2.4. WebLogic Server 8.1 is compatible with version 2.3. However, Oracle Application Server 10*g* Release 3 (10.1.3) is backward compatible to Servlet 2.3. Hence, servlets written to the standard 2.3 specification should work correctly in Oracle Application Server and require minimal migration effort.

The primary tasks involved in migrating servlets to a new environment are configuration and deployment. The use of proprietary extensions, such as htmlKona, will require additional tasks and complicate the migration effort.

The tasks involved in migrating servlets also depend on how the servlets have been packaged and deployed. Servlets can be deployed as a simple servlet, as a Web application packaged with other resources in a standard directory structure, or as a Web archive (WAR) file.

3.2 Migration Approach for Servlets

Migrating servlets to OC4J is a straightforward process using the following overall steps:

1. Configuration - Create or modify the Oracle Application Server deployment descriptors for the servlets.
2. Packaging:
   - Simple servlets can be deployed individually (see Section 3.3).
   - Servlets can be packaged as part of a Web application in a WAR file (see Section 3.4).
3. Deployment - Application Server Control Console can be used to deploy servlets in a WAR file. Individual servlets and servlets in exploded Web applications can be deployed automatically by copying them to the appropriate directories.

> **Note:** Oracle JDeveloper provides tools and wizards to automate all the above.

Servlet Migration Issues

The following are possible issues you may face during servlet migration:

- Earlier versions of WebLogic Server (6.0) Servlet 2.3 implementation is based on non-finalized Servlet 2.3 specification, which may require some code upgrade to use the finalized Servlet 2.3 API, which is the highest level specification supported by WebLogic Server 8.1. Oracle Application Server 10*g* Release 3 (10.1.3) supports Servlet 2.4 and is backward compatible to Servlet 2.3. Hence, WebLogic Server 6.0 and earlier servlets should be upgraded to the Servlet 2.3 API before being deployed in Oracle Application Server.
- Setting up `HttpSession` persistence varies between both Weblogic and OC4J.
- Proprietary server extensions (WebLogic htmlKona) need to be rewritten.
- If a servlet in WebLogic Server performs JNDI lookups to resources in applications other than its own, it may cause a `NameNotFoundException` when performing the same lookup after it is migrated to OC4J. This exception occurs if OC4J is not configured for global JNDI lookup. Refer to Section 5.1.2.1, "Global JNDI Lookups and Oracle Application Server" for a resolution.

3.3 Migrating a Simple Servlet

Simple servlets are easily configured and deployed in OC4J. The manual process used to deploy a servlet is the same in both WebLogic Server and OC4J.

> **Note:** The recommended and preferred way to deploy a servlet is by packaging it in a WAR or EAR file and using Oracle Enterprise Manager 10*g* Application Server Control Console. The manual processes described in this chapter of editing XML files and starting OC4J at the command line using the `java` command should preferably be used in a development environment.

A servlet must be registered and configured as part of a Web application. To register and configure a servlet, several entries must be added to the Web application deployment descriptor.

The overall steps to deploy a simple servlet are as follows (detailed steps are in Table 3–1):

1. Update the Web application deployment descriptor (`web.xml`) with the name of the servlet class and the URL pattern used to resolve requests for the servlet.
2. Copy the servlet class file to the `WEB-INF/classes/` directory. If the servlet class file contains a package statement, create additional subdirectories for each level of the package statement. The servlet class file must then be placed in the lowest subdirectory created for that package.
3. Copy other supporting files required by the servlet to the appropriate directory in the Oracle Application Server installation.
4. Invoke the servlet from your browser by entering its URL.

To determine the effort involved in migrating servlets, we selected and migrated example servlets provided with WebLogic Server. We chose examples that did not use proprietary extensions.

Table 3–1 presents the manual process for migrating a simple servlet, HelloWorld, from WebLogic Server to Oracle Application Server OC4J.

Table 3–1 Migrating a Simple Servlet

Step	Description	Process
1	Modify the Web application deployment descriptor	Add the descriptor information below to the `web.xml` file located in the following directory in your Oracle Application Server installation: For UNIX, `web.xml` can be found in: *<ORACLE_HOME>*`/j2ee/home/default-web-app/WEB-INF/` For Windows, `web.xml` can be found in: *<ORACLE_HOME>*`\j2ee\home\default-web-app\WEB-INF\` The descriptor information to be entered is: `<servlet>` `  <servlet-name>` `  HelloWorldServlet` `  </servlet-name>` `  <servlet-class>` `  examples.servlets.HelloWorldServlet` `  </servlet-class>` `</servlet>` `<servlet-mapping>` `  <servlet-name>` `  HelloWorldServlet` `  </servlet-name>` `  <url-pattern>` `  /HelloWorldMigrate/*` `  </url-pattern>` `</servlet-mapping>`

Table 3–1 (Cont.) Migrating a Simple Servlet

Step	Description	Process
2	Copy the servlet class file to the appropriate directory	After building and running successfully the samples that came with WebLogic Server, copy `HelloWorldServlet.class` from a directory in your WebLogic Server installation to the appropriate directory in Oracle Application Server as follows: In UNIX, from: `<BEA_HOME>/weblogic81/samples/server/examples/build/examplesWebApp/WEB-INF/classes/examples/servlets/` to: `<ORACLE_HOME>/j2ee/home/default-web-app/WEB-INF/classes/examples/servlets/` In Windows, from: `<BEA_HOME>\weblogic81\samples\server\examples\build\examplesWebApp\WEB-INF\classes\examples\servlets\` `to:` `<ORACLE_HOME>\j2ee\home\default-web-app\WEB-INF\classes\examples\servlets\` NOTE: This servlet provided with the WebLogic Server installation belongs to a package called examples.servlets. When copying its class file to Oracle Application Server, you need to create the corresponding package subdirectories (examples/servlets/).
3	Extract and copy the supporting utility class file required by the servlet.	`HelloWorldServlet.class` imports a utility class, `ExampleUtils.class`, found in the JAR file, `utils_common.jar`, in the WebLogic Server installation. This utility class needs to be copied to your Oracle Application Server installation. `ExampleUtils.class` can be found in the following location: (UNIX) `<BEA_HOME>/weblogic81/samples/server/examples/build/examplesWebApp/WEB-INF/lib/utils_common.jar` (Windows) `<BEA_HOME>\weblogic81\samples\server\examples\build\examplesWebApp\WEB-INF\lib\utils_common.jar` Extract `ExampleUtils.class` to: (UNIX): `<ORACLE_HOME>/j2ee/home/default-web-app/WEB-INF/classes/` (Windows): `<ORACLE_HOME>\j2ee\home\default-web-app\WEB-INF\classes\` Note: The `examples/utils/common` directory is created from the extraction.
4	Restart the `home` OC4J "home" instance, or start it if it is not currently running	Use the Oracle Enterprise Manager 10*g* Application Server Control Console administration Web pages or the following `opmnctl` command (executed locally): `opmnctl @instance restartproc ias-component=OC4J`

Table 3–1 (Cont.) Migrating a Simple Servlet

Step	Description	Process
5	Run the servlet from your Web browser	Access the servlet from your Web browser using the URL `http://localhost:7777/j2ee/HelloWorldMigrate` (Substitute "`localhost`" with your OC4J instance's host name if using the browser from another machine.)

See Also: *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for detailed information on configuring and deploying servlets.

3.4 Migrating a WAR File

A Web application can be configured and deployed as a WAR file. This is easily accomplished in OC4J by using the Application Server Control Console administration GUI or manually copying the WAR file to the appropriate directory. This is also true for WebLogic Server. We will illustrate the process using Application Server Control Console to deploy an example WAR file from WebLogic Server.

Note: Manually copying a WAR file to the appropriate directory to deploy it should only be done in a development environment where OC4J is in standalone mode (not a component of an Oracle Application Server instance).

Production Web applications are typically deployed using WAR or EAR files through Application Server Control Console. During the development of a Web application, it may be faster to deploy and test edited code using an exploded directory format (see Section 3.5).

Table 3–2 presents the typical process for migrating a WAR file from WebLogic Server to OC4J.

Table 3–2 Migrating a WAR File

Step	Description	Process
1	Create the WAR file for the sample application.	If you have not run all the WebLogic Server samples that came with that product, build the cookie sample Web application in the following WebLogic Server directory (UNIX is shown but Windows has equivalent directory structure): `<BEA_HOME>/weblogic81/samples/server/examples/src/examples/webapp/cookie` In this directory, build the application by typing `<BEA_HOME>/ant/weblogic81/server/bin/ant` When built, a WAR file for this application, `cookie.war`, is created in the following directory: `<BEA_HOME>/weblogic81/samples/domains/examples/applications/`

Table 3–2 (Cont.) Migrating a WAR File

Step	Description	Process
2	Deploy the sample application to Oracle Application Server.	**1.** On the machine where the `cookie.war` file is located, open a browser and go to the Application Server Control Console URL of your Oracle Application Server installation. For example: `http://<hostname>:7777/em` where *<hostname>* is the qualified name of the host where Oracle Application Server is installed. **2.** Enter your administrator username (`oc4jadmin`) and password if prompted. **3.** Click the "+" icon of the Oracle Application Server instance you want to deploy the WAR file to. **4.** Click the `home` OC4J component, which brings up its settings page. **5.** Click "Applications" and then the "Deploy" button. The "Deploy: Select Archive" page appears. **6.** Select "Archive is present on local host." Click the "Browse" button and enter the location of the `cookie.war` file. For deployment plan, leave the selection to automatically create a deployment plan. Click Next. The "Deploy: Application Attributes" page appears. **7.** In the "Application Name" and "Context Root" text boxes, enter "cookie" and "/cookie" respectively. Click Next. The "Deploy: Deployment Settings" page appears. **8.** Click Deploy. A page showing the deployment progress appears. A success message appears if the deployment is successful. If an error occurs, view the progress messages and attempt to troubleshoot the problems logged. **9.** Click Return. The `home` settings page appears, and the cookie application should appear in the list of deployed applications.
3	Test the deployed application.	In a browser, enter the following URL: `http://<hostname>:7777/cookie` where *<hostname>* is the Oracle Application Server host where you deployed the cookie WAR file.

See Also: *Oracle Application Server Containers for J2EE Servlet Developer's Guide* and *Oracle Application Server Containers for J2EE User's Guide* for more detailed information on deploying WAR and EAR files.

3.5 Migrating an Exploded Web Application

Web applications can also be configured and deployed as a collection of files stored in a standard directory structure or exploded directory format. This can be accomplished in OC4J by manually copying the contents of the standard directory structure to the appropriate directory in the OC4J installation. The same method can also be used for WebLogic Server. In this section, we will describe the manual process for deploying an exploded Web application.

> **See Also:** *Oracle Application Server Administrator's Guide* for detailed information on using the Application Server Control Console GUI.

Deploying a Web application in exploded directory format is used primarily during the development of a Web application. It provides a fast and easy way to deploy and test changes. When deploying a production Web application, package the Web application in a WAR file and deploy the WAR file using Application Server Control Console.

WebLogic Server

To manually deploy an exploded Web application in WebLogic Server (development mode), copy the top-level directory containing the exploded Web application files into the following directories of your WebLogic Server installation:

(UNIX) `<BEA_HOME>/user_projects/domains/<domain_name>/applications`
(Windows) `<BEA_HOME>\user_projects\domains\<domain_name>\applications`

*For **WebLogic Server** 7.0* - Once the top-level directory is copied to the appropriate directory, create an empty file with the name "`REDEPLOY`" within the top-level directory. WebLogic Server detects this file and deploys the Web application. (WebLogic Server reads the timestamp of this file every few minutes to determine if the application needs redeploying. Hence, whenever an application file is updated, the `REDEPLOY` file's timestamp has to be updated to redeploy the file. In UNIX, this can be done by using the `touch` command.)

*For **WebLogic Server** 8.1* - When WebLogic Server is running in development mode, any exploded directory files placed in the abovementioned directory is automatically deployed. There is no need to create the "`REDEPLOY`" file as for version 7.0.

Oracle Application Server

Manually deploying an exploded Web application in OC4J varies slightly. Copy the top-level directory containing the exploded Web application into the following directory of your OC4J installation:

(UNIX) `<ORACLE_HOME>/j2ee/home/applications`
(Windows) `<ORACLE_HOME>\j2ee\home\applications`

Then, modify the following application deployment descriptor to include the Web application:

(UNIX) `<ORACLE_HOME>/config/application.xml`
(Windows) `<ORACLE_HOME>\config\application.xml`

Bind the Web application to your Web site by adding an entry in the following descriptor file:

(UNIX) `<ORACLE_HOME>/config/default-web-site.xml`
(Windows) `<ORACLE_HOME>\config\default-web-site.xml`

Finally, register the new application by adding a new `<application>` tag entry in the following files:

(UNIX) `<ORACLE_HOME>/config/server.xml`
(Windows) `<ORACLE_HOME>\config\server.xml`

When you modify `server.xml` and save it, OC4J detects the timestamp change of this file and deploys the application automatically. OC4J need not be restarted.

3.6 Migrating Configuration and Deployment Descriptors

Since WebLogic Server and Oracle Application Server fully support J2EE 1.4, there is a standard set of XML configuration files supported by both application servers. These are:

- **`web.xml`** (found in the `WEB-INF` directory of a Web application's WAR file)
- **`application.xml`** (found in the `META-INF` directory of a Web application's WAR file)
- **`ejb-jar.xml`** (found in the `META-INF` directory of an EJB module's exploded directory hierarchy)

In addition to the standard files, each application server has specific files used only by their respective environments. These are:

3.6.1 Oracle Application Server

- **`server.xml`**
  Found in

  (UNIX) <*ORACLE_HOME*>`/j2ee/home/config/`
  (Windows) <*ORACLE_HOME*>`\j2ee\home\config\`

  This is the overall OC4J runtime configuration file. It defines attributes such as the deployed applications directory, the server log file path and name, path and names of other XML files, names of applications and their EAR files, paths to runtime libraries, and so on.

- **`application.xml`**
  Found in

  (UNIX) <*ORACLE_HOME*>`/j2ee/home/config\`
  (Windows) <*ORACLE_HOME*>`\j2ee\home\config\`

  This is the global configuration file common settings for all applications deployed on a particular OC4J installation. Note that this is different from the `application.xml` in a J2EE WAR file.

- ***`<website_name>`*`-web-site.xml`**
  Found in

  (UNIX) <*ORACLE_HOME*>`/j2ee/home/config\`
  (Windows) <*ORACLE_HOME*>`\j2ee\home\config\`

  This file defines a Web site and specifies attributes such as host name, HTTP listener port number, Web applications it services and their URL contexts, and HTTP access log file and path. Note that the name and path of each `*-web-site.xml` file has to be specified in the `server.xml` file for OC4J to configure the defined Web site at runtime.

- **`data-sources.xml`**
  Found in

  (UNIX) <*ORACLE_HOME*>`/j2ee/home/config/`
  (Windows) <*ORACLE_HOME*>`\j2ee\home\config\`

  This file contains configuration information for data sources used by the OC4J runtime. Information in this file include: JDBC drivers used, JNDI binding for each data source, username and password for each data source, database schemas to use, maximum connections to each database, and time out values.

- **principals.xml**
  Found in

  (UNIX) <*ORACLE_HOME*>/j2ee/home/config/
  (Windows) <*ORACLE_HOME*>\j2ee\home\config\

  This file contains the user repository for the default `XMLUserManager` class. Groups, users belonging to them, and group permissions are defined in this file. The mapping of groups to roles is defined in the global `application.xml` file.

- **orion-application.xml**
  Found in

  UNIX: <*ORACLE_HOME*>/j2ee/home/application-deployments/<app_name>

  or

  Windows: <*ORACLE_HOME*>\j2ee\home\application-deployments\<*app_name*>

  This file contains OC4J-specific information for an application (<app_name>) deployed on an OC4J installation. Web and EJB module names and security information for the application are included in the file. This file is generated by OC4J at deploy time.

- **global-web-application.xml**
  Found in

  (UNIX) <*ORACLE_HOME*>/j2ee/home/config/
  (Windows) <*ORACLE_HOME*>\j2ee\home\config\

  This file contains servlet configuration information used internally by the OC4J runtime. An example is the JSP translator servlet.

- **orion-web.xml**
  Found in

  UNIX:
  <*ORACLE_HOME*>/j2ee/home/application-deployments/
  <app_name>/<*web_app_name*>/

  or

  Windows:
  <*ORACLE_HOME*>\j2ee\home\application-deployments\
  <*app_name*>\<*web_app_name*>\

  OC4J internal JSP and servlet information for <web_app_name> is specified in this file. This file is generated by OC4J at deploy time.

- **orion-ejb-jar.xml**
  Found in

  UNIX:
  <*ORACLE_HOME*>/j2ee/home/application-deployments/
  <app_name>/<ejb_jarfile_name>/

  or

  Windows:
  <*ORACLE_HOME*>\j2ee\home\application-deployments\
  <*app_name*>\<ejb_jarfile_name>\

This file contains OC4J internal deployment information for EJBs in the JAR file specified by <ejb_jarfile_name> belonging to the application <app_name>. This file is generated by OC4J at deploy time.

- **oc4j-connectors.xml**
  Found in

  (UNIX) <*ORACLE_HOME*>/j2ee/home/config/
  (Windows) <*ORACLE_HOME*>\j2ee\home\config\

  This file contains connector information for the OC4J installation.

3.6.2 WebLogic Server

- **config.xml**
  Found in

  (UNIX) <*BEA_HOME*>/config/<domain_name>/
  (Windows) <*BEA_HOME*>\config\<*domain_name*>\

  This file contains configuration information for an entire WebLogic Server domain. Information specified in this file include the domain administration server's host name and admin port number, JNDI mappings to data sources, JDBC connection pool information, applications deployed to all nodes in the domain, SSL certificate information,

- **weblogic.xml**
  Found in

  UNIX:
  <*BEA_HOME*>/config/<domain_name>/applications/
  <*web_app_name*>/WEB_INF/

  or

  Windows:
  <*BEA_HOME*>\config\<*domain_name*>\applications\
  <*web_app_name*>\WEB_INF\

  This file defines JSP properties, JNDI mappings, resource references, security role mappings, and HTTP session and cookie parameters for a Web application. This file is WebLogic Server-specific but is created manually.

- **weblogic-ejb-jar.xml**
  Found in an EJB module's META-INF subdirectory. This file maps WebLogic Server resources to EJBs. These resources include security role names, data sources, JMS connections, and other EJBs. This file also has performance attributes for caching and clustering for the EJBs defined in the corresponding ejb-jar.xml file.

Note: The files mentioned above are not an exhaustive list of all XML configuration file used by each application server. They are files which are relevant to the configuration and deployment of servlet applications. Other XML files also exist to configure components such as HTTP listeners, RMI, security.

3.7 Migrating Cluster Aware Applications

Oracle Application Server provides more comprehensive clustering features than WebLogic Server.

WebLogic Server provides two primary cluster services, HTTP session state clustering and object clustering. The focus of this section is on HTTP session state clustering or Web application clustering.

WebLogic Server supports clustering for servlets and JSP pages by replicating the HTTP session state of clients accessing clustered servlets and JSP pages. To benefit from HTTP session state clustering, you must ensure that the HTTP session state is persistent by configuring either in-memory replication, file system persistence, or JDBC persistence.

Oracle Application Server provides clustering support similar to that of WebLogic Server. In addition, Oracle Application Server provides:

- **Servlet Clustering**—OC4J provides facilities to cluster servlets without requiring any changes to the Web application. The changes necessary are deployment configuration modifications that are transparent to the Web application and allows session fail over to multiple OC4J processes.
- **Clustering Architecture and Simplicity**—An important differentiator for Oracle Application Server is the ease with which different instances can be clustered and the robustness of the architecture used for clustering.
- **Clustering Simplicity**—Oracle Enterprise Manager 10*g* Application Server Control Console provides a GUI to configure various OracleAS instances to belong to a single cluster, whether they are multiple servers with load balancing on a single machine or on different machines. Alternatively, you can also edit a single XML file. In contrast, it is more complex to configure WebLogic Server clusters with load balancing either with multiple instances on one machine or on multiple machines.
- **Superior Clustering Architecture—**OC4J uses dynamic IP addresses to register instances as part of a cluster. Any standard load balancer such as Cisco Local Director or BigIP has the ability to use a variety of load balancing mechanisms to route requests to different Oracle Application Server instances. Additionally, `mod_oc4j` intelligently routes requests from Oracle HTTP Server to OC4J processes using one of several load balancing algorithms. In contrast, WebLogic Server uses static IP addresses to configure clustering. Static IP addresses preclude the use of a load balancer to distribute requests across instances. As a result, you get either clustering or load balancing with WebLogic Server but not both.

> **See Also:** *Oracle Application Server High Availability Guide* and *Oracle Containers for J2EE Configuration and Administration Guide*

4

Migrating JSP Pages

This chapter provides the information you need to migrate JavaServer pages from WebLogic Server to Oracle Application Server. It covers the migration of simple JSP pages, custom JSP tag libraries, and WebLogic custom tags.

This chapter contains these topics:

4.1 Introduction

Migrating JSP pages from WebLogic Server to Oracle Application Server is straight forward and requires little or no code changes.

WebLogic Server 8.1 is compliant with Sun Microsystem's JavaServer Page specifications, version 1.2. Oracle Application Server 10*g* Release 3 (10.1.3) is compliant with JSP 2.0 and is backward compatible to version 1.2. Hence, JSP pages written to the standard version 1.2 specification should work correctly in Oracle Application Server and require minimal migration effort.

The primary tasks involved in migrating JSP pages to a new environment are configuration and deployment. The use of proprietary extensions and tag libraries will require additional tasks and complicate the migration effort.

The tasks involved in migrating JSP pages also depend on how the JSP pages have been packaged and deployed. JSP pages can be deployed as a simple JSP page, as a Web application packaged with other resources in a standard directory structure (WAR file), or as a enterprise application archive (EAR) file. The migration of Web applications in exploded directory format and WAR files is addressed in Chapter 3, "Migrating Java Servlets".

4.1.1 Differences Between WebLogic Server and Oracle Application Server JSP Implementations

Since both WebLogic Server and Oracle Application Server Containers for J2EE (OC4J) have implemented the same versions of the Java Server Pages specifications, there are

no differences between the two in the core JSP specification areas. There are differences in areas outside the core specifications. These are listed in Table 4–1.

Table 4–1 JSP feature comparison

Feature	Oracle Application Server 10*g* Release 3 (10.1.3)	WebLogic Server 8.1
JSP Version Support	2.0	1.2
Basic JSP Tag Libraries	Yes	Yes
Advanced JSP Tag Libraries	Yes	No
JSP Source Level Debugging	Yes	No
ASP to JSP Source Level Conversion	Yes	No

Each vendor provides their own custom JSP tags. WebLogic Server provides four specialized JSP tags that you can use in your JSP pages. OC4J also provides various JSP tags - Oracle JSP Markup Language (JML) Custom Tag Library, tags for XML and XSL integration, and several JSP utility tags. A comprehensive discussion of these tags can be found in *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*.

4.1.1.1 OC4J JSP Features

Oracle Application Server provides one of the fastest JSP engines on the market. Further, it also provides several value-added features and enhancements such as support for globalization, SQLJ, and expression language. If you are familiar with Oracle9*i*AS 1.0.2.2, the first release of Oracle Application Server to include OC4J, there were two JSP containers: a container developed by Oracle and formerly known as OracleJSP and a container licensed from Ironflare AB and formerly known as the "Orion JSP container".

In Oracle Application Server, these have been integrated into a single JSP container, referred to as the "OC4J JSP container". This new container offers the best features of both previous versions, runs efficiently as a servlet in the OC4J servlet container, and is well integrated with other OC4J containers. The integrated container primarily consists of the OracleJSP translator and the Orion container runtime running with a new simplified dispatcher and the OC4J 1.0.2.2 core runtime classes. The result is one of the fastest JSP engines on the market with additional functionality over the standard JSP specifications.

OC4J JSP provides extended functionality through custom tag libraries and custom JavaBeans and classes that are generally portable to other JSP environments:

- Extended types implemented as JavaBeans that can have a specified scope
- `JspScopeListener` for event handling
- Integration with XML and XSL through custom tags
- Data-access JavaBeans
- The Oracle JSP Markup Language (JML) custom tag library, which reduces the level of Java proficiency required for JSP development
- OC4J JSP includes (non exhaustively) connection pooling tags, XML tags, EJB tags, file access tags, email tags, caching tags, OracleAS Personalization tags, OracleAS Ultrasearch tags, a custom tag library for SQL functionality, and support for JSTL (JavaServer Pages Standard Tag Library). WebLogic only has four: `cache`, `process`, `repeat`, and form validation.

- JESI (Edge Side Includes for Java) tags and Web Object Cache tags and API that work with content delivery network edge servers to provide an intelligent caching solution for Web content (see the following sub-sections).

  See Also: *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on custom JSP tag libraries.

The OC4J JSP container also offers several important features such as the ability to switch modes for automatic page recompilation and class reloading, JSP instance pooling, and tag handler instance pooling.

The following sections provide a summary of Edge Side Includes for Java (JESI) Tags and Web Object Cache Tags. See the *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference* for a discussion of other available tags.

4.1.1.1.1 Edge Side Includes for Java (JESI) Tags OC4J provides fine-grained control allowing developers to cache fragments of JSP pages down to each individual tag - these can be cached in OracleAS Web Cache and are automatically invalidated and refreshed when a JSP changes. The technology behind this is Edge Side Includes (ESI), a W3C standard XML schema/markup language that allows dynamic content to be cached in a Web Cache or to be assembled in an edge network. By caching this dynamic content, it reduces the need to execute JSPs or Servlets, thereby improving performance, off loading the application servers, and reducing latency. JESI (JSP to ESI) tags are layered on top of an Edge Side Includes (ESI) framework to provide ESI caching functionality in a JSP application. JESI tags enable the user to break down dynamic content of JSP pages into cacheable components or fragments.

4.1.1.1.2 Web Object Cache Tags The Web Object Cache is an Oracle Application Server feature that allows Web applications written in Java to capture, store, reuse, post-process, and maintain the partial and intermediate results generated by JSPs or Servlets. For programming interfaces, it provides a tag library (for use in JSP pages) and a Java API (for use in Servlets). Cached objects might consist of HTML or XML fragments, XML DOM objects, or Java serializable objects. By caching these objects in memory, various operations can be carried out on the cached objects including:

- Applying a different XSLT based on user profile or device characteristics on the stored XML
- Re-using a cached object outside HTTP, such as SMTP to send e-mail to clients.

4.1.1.2 Oracle JDeveloper and OC4J JSP Container

Oracle JDeveloper is integrated with the OC4J JSP container to support the full JSP application development cycle - editing, source-level debugging, and running JSP pages. It also provides an extensive set of data-enabled and Web-enabled JavaBeans, known as JDeveloper Web beans and a JSP element wizard which offers a convenient way to add predefined Web beans to a page. JDeveloper also provides a distinct feature that is very popular with developers. It allows you to set breakpoints within JSP page source and can follow calls from JSP pages into JavaBeans. This is much more convenient than manual debugging techniques, such as adding print statements within the JSP page to output state into the response stream for display on browser or to the server log.

4.2 Migration Approach

Migrating a JSP from WebLogic Server is straightforward and involves configuration, packaging (into a WAR file), and deployment tasks (to appropriate deployment directory). These tasks can be performed manually or by using Oracle JDeveloper.

In cases where proprietary tag libraries are used, they can be ported to either Oracle proprietary tags (see Section 4.4.1) or custom tags can be created to replace them.

Known JSP Migration Caveats

The following are known issues for migrating JSPs from WebLogic Server:

- WebLogic tags are located in the tagext directory.
- The WebLogic `cache` tag library is not supported. Use OC4J's Web Object Cache tag library (see Section 4.4.1.1).
- The following WebLogic-specific tags and properties have to be migrated:
  - `<process>` (see Section 4.4.1.2)
  - `<repeat>` (see Section 4.4.1.3)
  - `<%@ page extends="..." %>`

    This directive should be replaced with `<%@ page import="..." %>`.
  - htmlKona (see Section 4.5)

4.3 Migrating a Simple JSP Page

JSP pages do not require specific mappings as do HTTP servlets. To deploy a simple JSP page, you can copy the JSP page and any files required by the JSP page to the appropriate directories. No additional registrations are required.

> **Note:** Application Server Control Console should be used to deploy any type of applications including JSPs. But for the purpose of illustration, the JSP files in the following example are copied manually without using Application Server Control Console.

The deployment process has been simplified in OC4J by providing a J2EE Web application and various configuration files by default.

To determine the effort involved in migrating JSP pages, we selected and migrated example JSP pages provided with WebLogic Server. We chose examples that did not use proprietary extensions.

Table 4–2 presents the typical process for migrating a simple JSP page from WebLogic Server to OC4J.

Table 4–2 Migrating a Simple JSP Page

Step	Description	Process
1	Start an instance of OC4J, if none are currently running.	Use the Oracle Enterprise Manager 10*g* Application Server Control Console administration Web pages or the following `opmnctl` command (executed locally): `opmnctl @instance startproc ias-component=OC4J`

Table 4–2 (Cont.) Migrating a Simple JSP Page

Step	Description	Process
2	Copy the JSP page to the appropriate directory	Copy `HelloWorld.jsp` or `ShowDate.jsp` from its directory in your WebLogic Server installation to the appropriate directory in `Oracle Application Server` as follows: In UNIX, from: `<BEA_HOME>/weblogic81/samples/server/examples/src/examples/jsp/` to: `<ORACLE_HOME>/j2ee/home/default-web-app/` In Windows, from: `<BEA_HOME>\weblogic81\samples\server\examples\src\examples\jsp\` to: `<ORACLE_HOME>\j2ee\home\default-web-app\`
4	Request the JSP page from your Web browser	From a Web browser, request the JSP page through the URL: `http://<hostname>:7777/j2ee/HelloWorld.jsp` or `http://<hostname>:7777/j2ee/ShowDate.jsp` where *<hostname>* is the Oracle Application Server host you copied the JSP file to.

See Also: *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide* and *Oracle Application Server Containers for J2EE User's Guide* for detailed information on configuring and deploying JSP pages.

4.4 Migrating a Custom JSP Tag Library

WebLogic Server and OC4J provide the ability to create and use custom JSP tags. The process used to deploy a custom JSP tag library is similar for both WebLogic Server and OC4J.

Tag libraries can be packaged and deployed as part of a Web application and are declared in a specific section of the Web application deployment descriptor.

To determine the effort involved in migrating custom JSP tag libraries, we selected and migrated example JSP pages provided with WebLogic Server. We chose examples that did not use proprietary extensions.

Table 4–3 presents the typical process for migrating a JSP page that utilizes a custom JSP tag library from WebLogic Server 7.0 to OC4J. (Note that this example is not available in WebLogic Server 8.1.)

Table 4–3 Migrating a Custom JSP Tag Library from WebLogic Server 7.0

Step	Description	Process
1	Copy the tag library file to the appropriate directory	Copy `counter.tld` from UNIX: `<BEA_HOME>/weblogic700/samples/server/src/examples/jsp/tagext/counter/` Windows: `<BEA_HOME>\weblogic700\samples\server\src\examples\jsp\tagext\counter\` of your WebLogic Server installation to the following directory in your OC4J installation: UNIX: `<ORACLE_HOME>/j2ee/home/default-web-app/WEB-INF/` Windows: `<ORACLE_HOME>\j2ee\home\default-web-app\WEB-INF\`
2	Copy the JSP file that uses the tag library to the appropriate directory	Copy `pagehits.jsp` from UNIX: `<BEA_HOME>/weblogic700/samples/server/src/examples/jsp/tagext/counter/` Windows: `<BEA_HOME>\weblogic700\samples\server\src\examples\jsp\tagext\counter\` of your WebLogic Server installation to the following directory in your OC4J installation: UNIX: `<ORACLE_HOME>/j2ee/home/default-web-app/` Windows: `<ORACLE_HOME>\j2ee\home\default-web-app\`

Table 4–3 (Cont.) Migrating a Custom JSP Tag Library from WebLogic Server 7.0

Step	Description	Process
3	Copy any class files required by the tag library and used by the JSP file to the appropriate directory	Copy `Count.class`, `Display.class`, and `Increment.class` from UNIX: `<BEA_HOME>/weblogic700/samples/server/config/examples/applications/examplesWebApp/WEB-INF/classes/examples/jsp/tagext/counter/` Windows: `<BEA_HOME>\weblogic700\samples\server\config\examples\applications\examplesWebApp\WEB-INF\classes\examples\jsp\tagext\counter\` of your WebLogic Server installation to the following directory in your OC4J installation: UNIX: `<ORACLE_HOME>/j2ee/home/default-web-app/WEB-INF/classes/examples/jsp/tagext/counter/` Windows: `<ORACLE_HOME>\j2ee\home\default-web-app\WEB-INF\classes\examples\jsp\tagext\counter\` Note that these `.class` files provided with the WebLogic server installation belong to a package called `examples.jsp.tagext.counter`. You may need to create the `examples/jsp/tagext/counter/` directory (or Windows equivalent).
4	Copy image files used by the JSP file	Copy the directory containing the image files from UNIX: `<BEA_HOME>/weblogic700/samples/server/src/examples/jsp/tagext/counter/images/numbers/` Windows: `<BEA_HOME>\weblogic700\samples\server\src\examples\jsp\tagext\counter\images\numbers\` of the WebLogic Server installation to the following directory in your OC4J installation: UNIX: `<ORACLE_HOME>/j2ee/home/default-web-app/images/numbers/` Windows: `<ORACLE_HOME>\j2ee\home\default-web-app\images\numbers\` Note that you may have to create the `images/numbers` (or Windows equivalent) directory

Table 4–3 (Cont.) Migrating a Custom JSP Tag Library from WebLogic Server 7.0

Step	Description	Process
5	Modify the appropriate Web application deployment descriptor and save the changes	Add the directive entry below to the `web.xml` file located in the following directory of your OC4J installation: UNIX: `<ORACLE_HOME>/j2ee/home/default-web-app/WEB-INF/` Windows: `<ORACLE_HOME>\j2ee\home\default-web-app\WEB-INF\` Directive entry (`<taglib>` is a child element of `<web-app>`): `<taglib> <taglib-uri> counter </taglib-uri> <taglib-location> /WEB-INF/counter.tld </taglib-location> </taglib>`
6	Restart or start the OC4J instance, if it is not currently running.	Go to `http://<hostname>:1810` and restart/start the `home` OC4J instance. Or, use the following `opmnctl` command executed locally: `opmnctl @instance restartproc ias-component=OC4J`
7	Request the JSP file from your Web browser	From your Web browser, access the URL `http://<hostname>:7777/j2ee/pagehits.jsp` where *<hostname>* is the Oracle Application Server host you copied the files to.

See Also:

- *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide* for detailed information on configuring and deploying JSP pages.
- *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on custom JSP tag libraries.

4.4.1 Migrating from WebLogic Custom Tags

If WebLogic custom tags are used extensively throughout your Web application, then the best migration option is to use the WebLogic tag library by deploying it on OC4J. This option was discussed in the previous section, "Migrating a Custom JSP Tag Library". You can then migrate to the OC4J JSP tags if required.

If WebLogic custom tags are used sparingly throughout your Web application, then the best migration option is to modify the JSP pages to use the OC4J JSP tag library. This option is discussed below.

WebLogic Server provides three specialized JSP tags for use in JSP pages. They are `cache`, `process`, and `repeat`.

4.4.1.1 WebLogic Server `cache` Tag

OC4J provides a superset of the WebLogic Server `cache` tag in the form of Web Object Cache Tags. These tags provide additional functionality over the WebLogic `cache` tag. Further, the Web Object Cache Tags of OC4J are well integrated with other tag libraries such as the XML tag library. For example, the `cacheXMLObj` tag is well integrated with OC4J's XML tags.

One feature which does not have direct functionality mapping is "async". However, Edge Side Includes (ESI) and Edge Side Includes for Java (JESI) can provide similar functionality to it.

> **See Also:** *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on Web Object Cache tags and JESI tags.

4.4.1.2 WebLogic Server `process` Tag

OC4J does not have an exact equivalent for the `process` tag. The closest option is to substitute it with scriptlet `if` statements, the Apache Struts `<logic:exists>` tag, or JSTL tags. Alternatively, you could write Java code to implement the tag.

4.4.1.3 WebLogic Server `repeat` Tag

Prior to Oracle Application Server 10*g* Release 3 (10.1.3), this tag could be replaced with JML tag `jml:foreach`. However, since JML is now desupported and replaced with JSTL, the JSTL `c:forEach` or `x:forEach` tags can be used.

> **See Also:**
>
> - *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*
> - *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide*

4.5 Migrating htmlKona

htmlKona is an API available with WebLogic Server. It is used for creating and manipulating HTML programmatically. htmlKona was made available before JSP specifications were available. Since htmlKona is proprietary to WebLogic, Oracle Application Server does not support it, and the recommended approach to migrate applications using htmlKona to Oracle Application Server is to replace the use of htmlKona with JSPs. If there is a strong need to manipulate HTML programatically, Oracle JDeveloper provides UMX and other features to match and exceed the functionalities of htmlKona.

4.6 Precompiling JSP Pages

JSP pages are compiled automatically by the JSP compiler. However, when testing and debugging JSP pages, you may want to access the JSP compiler directly.

The JSP compiler parses a `.jsp` file into a `.java` file. The standard Java compiler is then used to compile the `.java` file into a `.class` file.

4.6.1 Using the WebLogic Server JSP Compiler

To start the WebLogic Server JSP compiler, type the following command in your WebLogic Server command line environment:

```
java weblogic.jspc -options fileName
```

The *fileName* parameter refers to the name of the JSP page to be compiled. Options may be specified before or after the JSP page name. The following example demonstrates the use of the `-d` option to compile `myFile.jsp` into the destination directory `weblogic/classes`:

```
java weblogic.jspc -d /weblogic/classes myFile.jsp
```

4.6.2 Using the OC4J JSP Pre-translator

In addition to the standard `jsp_precompile` mechanism, OC4J provides a command-line utility called `ojspc` for pre-translating JSP pages.

Consider the example where the JSP page, `HelloWorld.jsp`, is located in the following OC4J default Web application directory (copy the `HelloWorld.jsp` file from *<ORACLE_HOME>*`/j2ee/home/default-web-app/`, or the Windows equivalent, to this subdirectory):

UNIX:

```
<ORACLE_HOME>/j2ee/home/default-web-app/examples/jsp/
```

Windows:

```
<ORACLE_HOME>\j2ee\home\default-web-app\examples\jsp\
```

To pre-translate this JSP page, set your current directory to the application root directory, then, in `ojspc`, set the `_pages` directory as the output base directory using the `-d` option. This results in the appropriate package name and file hierarchy. To illustrate:

Note: Ensure that the `<ORACLE_HOME>/jdk/bin` is set in the path environment variable so that the correct `Java` executable is used for `ojspc`.

In UNIX (assume `%` is a UNIX prompt):

```
% cd j2ee/home/default-web-app
% ojspc -d ../application-deployments/default/defaultWebApp/persistence/_pages
     examples/jsp/HelloWorld.jsp
```

In Windows (in a command prompt window and where `Oracle` is the Oracle home for your Oracle Application Server installation):

```
C:\>cd Oracle\j2ee\home\default-web-app
C:\>ojspc -d ../application-deployments/default/defaultWebApp/persistence/_pages
     examples/jsp/HelloWorld.jsp
```

The directory structure above specifies an application-relative path of `examples/jsp/HelloWorld.jsp`. The translated JSP can be found in (for UNIX)

```
<ORACLE_HOME>/j2ee/home/application-deployments/default/
defaultWebApp/persistence/_pages/_examples/_jsp/
```

or, for Windows:

```
<ORACLE_HOME>\j2ee\home\application-deployments\default\
defaultWebApp\persistence\_pages\_examples\_jsp\
```

At execution time, the JSP container looks for compiled JSP files in the `_pages` subdirectory. The `_examples/_jsp/` subdirectory would be created automatically by `ojspc` if run as in the above example.

Invoke the JSP page through the URL `http://<hostname>:7777/j2ee/examples/jsp/HelloWorld.jsp`. Notice that response time is faster than without pre-translating.

> **See Also:** The chapter JSP Translation and Deployment in *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide*.

4.6.3 Standard JSP Pre-translation Without Execution (based on the JSP 1.1 specification)

You can specify JSP pre-translation, without execution, by enabling the standard `jsp_precompile` request parameter when invoking a JSP page from the browser. For instance, `http://<hostname>:<port>/foo.jsp?jsp_precompile=true`

Using the <*ORACLE_HOME*>`/j2ee/home/default-web-app/HelloWorld.jsp` file (or Windows equivalent) as an example, erase all the "`_HelloWorld*`" files in:

UNIX:

```
<ORACLE_HOME>/j2ee/home/application-deployments/default/
defaultWebApp/persistence/_pages/
```

Windows:

```
<ORACLE_HOME>\j2ee\home\application-deployments\default\
defaultWebApp\persistence\_pages\
```

Then, invoke the URL `http://<hostname>:7777/j2ee/HelloWorld.jsp?jsp_precompile=true`. The pre-translation is performed but the page does not appear on your browser. Check the `_pages` subdirectory for the translated files.

4.6.4 Configure the JSP Container for Execution with Binary Files Only

You can avoid exposing your JSP page source, for proprietary or security reasons, by pre-translating the pages and deploying only the translated and compiled binary files. JSP pages that are pre-translated, either from previous execution in an on-demand translation scenario or by using `ojspc`, can be deployed to any standard J2EE environment.

For further details, refer to the *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide*.

5

Migrating Enterprise JavaBean Components

This chapter provides the information you need to migrate Enterprise JavaBean components from WebLogic Server to Oracle Application Server. It addresses the migration of session and entity EJBs, as well as J2EE Web applications in the form of EAR files or in an exploded directory format.

This chapter contains these topics:

5.1 Introduction

Migrating Enterprise JavaBeans (EJB) from WebLogic Server to Oracle Application Server is straightforward requiring little or no code changes to the EJBs migrated. Both application servers support the EJB 2.0 specification with OC4J extending support to EJB 2.1 and EJB 3.0 Early Draft Review.

All EJBs written and designed to the EJB 2.0 specifications should work correctly and require minimal migration effort. The primary effort goes into configuring and deploying the applications in the new environment. Only in cases where proprietary extensions are used will the migration effort get complex.

In this chapter we cover the migration of EJBs deployed in the form of EAR files or in an exploded directory format.

5.1.1 Comparison of WebLogic Server and Oracle Application Server EJB Features

Since WebLogic Server 8.1 supports EJB 2.0 and Oracle Application Server Containers for J2EE (OC4J) supports EJB 2.1 and the EJB 3.0 Early Draft Review, some differences exist in the two implementations. The following table summarizes the EJB features available from both application servers.

Table 5–1 Comparison of EJB features

Feature	Oracle Application Server 10*g* (10.1.3)	WebLogic Server 7.0	WebLogic Server 8.1
Session Beans	Available	Available	Available
Container-Managed Persistence Entity Beans (CMP)	Available	Available	Available
Bean-Managed Persistence Entity Beans (BMP)	Available	Available	Available
Message Driven Beans	Available	Available	Available
JTA Transactions	Available	Available	Available
JCA Enterprise Connectivity	Available	Available	Available
IMS Messaging	Available	Available	Available
Dynamic EJB Stub Generation	Available	Available	Avaiable
Full EAR File Based Deployment	Available	Available	Available
Automatic Deployment of EJB Applications	Available	Available	Available
Stateless and Stateful EJB Clustering	Available	Available	Available
Local Interfaces for Enterprise JavaBeans	Available	Available except for MDBs.	Available except for MDBs.
EJB Query Language (EJB-QL) - Automatic Code Generation - Oracle and Non Oracle Database Support	Available	Available. Extended by WebLogic QL.	Available. Extended by WebLogic QL.
RMI-over-IIOP Support	Available	Available	Available
CMP with Relationships	Available	Available	Available
Concurrency Control - Read-Only Locking - Pessimistic Locking - Optimistic Locking	Available	Available	Available

5.1.1.1 More Efficient Container Managed Persistence

There are two specific facts that reflect the significant performance advantages in using OC4J's container-managed persistence (CMP) implementation compared to WebLogic Server's implementation:

- **Automatic Detection of Modified EJBs** - When using CMP, Oracle Application Server's J2EE container can automatically detect whether you have modified an EJB and writes the EJB's state to the database; it invokes `ejbStore` only when necessary. WebLogic Server does not provide such automatic detection, requiring a user to code `is-modified` methods, which the WebLogic Server container uses to know whether or not to perform `ejbStore` operations.
- **Simple and Complex Database mapping for CMP** - Oracle Application Server's J2EE container supports both simple (1:1, 1:many) and complex (many:many) database field mappings very efficiently. In contrast, WebLogic

Server provides rudimentary support for simple CMP database field mapping (1:many). Additionally, qualifying a `where` clause string in WebLogic Server results in unnecessary full table scans.

5.1.1.2 Clustering Support

From a comparative point of view, Oracle Application Server's J2EE container provides the following clustering features:

- **Servlet clustering** - Oracle Application Server provides facilities to cluster servlets without requiring any changes to the user's application. The changes are deployment configuration modifications which are transparent to the J2EE application.
- **Clustering architecture simplicity** - An important differentiator for Oracle Application Server's J2EE container is the ease with which different instances can be clustered and the robustness of the architecture used for clustering. Specifically, Oracle Application Server requires modification of a single XML file (can be done through Application Server Control Console) to configure various Oracle Application Server instances to belong to a single cluster whether they are multiple servers with load balancing on a single machine or multiple servers with load balancing on several machines.

  In contrast, it is much more complex to configure WebLogic Server clusters with load balancing either with multiple instances on one machine or on multiple machines. For instance, if you indicate that your EJBs are to be used in a cluster, you need to specify that fact during the time the EJB stubs are created using `appc`, which then results in the creation of special cluster-aware classes that are used for deployment. Overall, Oracle Application Server's J2EE container, together with other Oracle Application Server components, provide a more robust clustering architecture with better ease-of-use.
- **Stateful session bean and entity bean clustering** - Oracle Application Server supports clustering of stateful session beans and entity beans. The following aspects of design are focused upon:
  - clustered performance - Existing clustering facilities such as those in WebLogic Server impose a severe performance penalty when running the instances in a stateful fashion with clustering. As a result, most application developers choose to keep their middle tier completely stateless and write their state to a persistent store, such as a database. By design, OC4J's clustering implementation is optimized to avoid introducing performance penalties.
  - programmatic simplicity - Unlike servlets which have a natural session boundary at which to fail over their state, EJBs do not have such a clear boundary. As a result, Oracle Application Server provides simple programmatic facilities to allow developers to use EJB clustering without any changes to their applications.

    See Also:

    Oracle Application Server High Availability Guide
    Oracle Containers for J2EE Configuration and Administration Guide

5.1.1.3 Scalability and Performance Enhancements

- **Entity bean scalability** - Oracle Application Server enhances entity beans scalability by enabling multiple clients to concurrently look up and invoke methods on the same entity bean instance, using a configurable pool of bean wrapper instances per primary key value.

- **Better concurrency control** - Oracle Application Server introduces a number of new concurrency control options to improve scalability and performance of large J2EE applications:
  - Read-only locking - For read-only beans that are not updating the database, the bean developer can instruct the OC4J container to avoid calling or generating `ejbStore()`. The appropriate isolation mode is selected, depending on whether the state of the bean can be updated by external systems, such as non-EJB applications using SQL.
  - Pessimistic locking - Oracle Application Server can serialize access to bean state while providing each client with its own bean instance for deterministic timeout and deadlock detection.
  - Optimistic locking - Oracle Application Server also supports an alternate locking scheme, which does not use row locking - data consistency depends on the isolation mode of the bean ("*Non- Repeatable-Reads*" or "*Serializable*") and the order in which clients update the rows.

WebLogic Server provides a similar set of features.

5.1.2 EJB Migration Considerations

In practice, the process of migrating EJBs from WebLogic Server to Oracle Application Server does not entail any major hurdles. Generally, little or no code modifications are required. If there are modifications, these are often related to JNDI portability issues regarding instantiation of JNDI initial contexts and JNDI lookup names to use for data sources and EJB home and remote interfaces.

Migration then consists of performing implementation-specific adaptation tasks for container class generation and object-relational (O-R) mapping definitions, and customization of deployment properties, if required.

One of the goals of the EJB initiative is to deliver component portability between different environments not only at source code level, but also at the binary level. Another goal is to ensure portability of compiled and uniformly packaged components.

While it is true that EJBs do offer portability, there are still a number of non portable, implementation-specific aspects that need to be addressed when migrating components from one implementation to another. Typically, an EJB component requires low level interfaces with the container in the form of stub and skeleton classes which need to stay implementation-specific. In effect, a clear partitioning between portable and non portable elements of an EJB component can be drawn.

Portable EJB elements include:

- The actual component implementation classes and interfaces (bean class, and remote and home interfaces).
- The assembly and deployment descriptors that describe generic component properties such as JNDI names and transactional attributes.
- Security attributes.

Implementation-specific elements include:

- Low-level helper implementation classes (stubs and skeletons) that interface with the host container.

- O-R mapping definitions for CMP entity beans, including search logic for custom finder methods that are declared in an implementation-specific format proprietary to each platform.
- Every component has a set of properties that require systematic configuration at deployment time. For example, mapping of security roles declared in an EJB component to actual users and groups is a task that is systematically performed at deployment time because mappings may not be known in advance. Also, they may have dependencies on the structure and population of the user directory on the target deployment server.

5.1.2.1 Global JNDI Lookups and Oracle Application Server

When migrating an EJB (or any other object that performs JNDI lookups) from WebLogic Server to OC4J, global JNDI lookup needs to be enabled for OC4J if the EJB performs JNDI lookups to resources in other applications other than its own. JNDI lookups in WebLogic Server have a global context whereas the default configuration of OCJ4 allows lookups within the application scope only. Hence, if an EJB performs a lookup to another application in OC4J with default configuration, a `NameNotFoundException` is thrown.

To enable global JNDI lookups for an OC4J instance, set the `global-jndi-lookup-enabled` attribute to `true` in the `<application-server>` element of `server.xml`. For JNDI names to resolve properly to their bound classes, the target application's classes must be in the classpath of the application attempting the lookup. This can be accomplished in by performing either one of the following tasks:

- Put the target application's JAR archive in the `$ORACLE_HOME/j2ee/home/applib` common classes directory.
- Put the target application's JAR archive in a shared library location. A shared library can be defined in the `<shared-library>` element of `server.xml`. See *Oracle Containers for J2EE Configuration and Administration Guide* for more information on this element.

For further information on Oracle Application Server JNDI, see *Oracle Application Server Containers for J2EE Services Guide*.

5.1.2.2 WebLogic Server Caveats

The following are additional notes on the WebLogic Server EJB implementation:

- The WebLogic Server implementation of BMP security is not in total compliance with the J2EE specification. According to the specification, an exception needs to be thrown when there is a violation in a BMP security role permission. While OC4J throws an exception in compliance with the specification, WebLogic Server does not do so.
- Unlike WebLogic Server, OC4J does not make it necessary for the developers to create a proprietary XML file for EJB deployment such as the WebLogic Server `weblogic-ejb-jar.xml`. For OC4J, `orion-ejb-jar.xml` is created by the OC4J container for internal purposes. Developers have the ability to modify this file to provide their own EJB configuration data, if needed, but it is not necessary for developers to explicitly create this file. Any custom EJB configuration information in the `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml` files can be migrated to `orion-ejb-jar.xml` using XSLT or the Oracle JDeveloper Migration Assistant. See Section 5.2.3.1.

5.2 Migration Approach

The overall migration approach EJBs is outlined in this section. Session EJBs migrate easier than entity EJBs since there are no persistence-level issues to be considered. Standard J2EE components and deployment descriptors (`ejb-jar.xml`) require almost no modifications.

For both session and entity EJBs, implementation-specific dependencies require modification. These include (as mentioned earlier):

- Hard-coded JNDI context access and lookups.
- Data source JNDI names and lookups.
- Implementation-specific adaptations for O-R mapping, container class generation, and customization of deployment properties.
- Modification and regeneration of the EJB archive file (JAR).

5.2.1 Migrating Session EJBs

Migration of session beans involves the "generic steps" of EJB migration. These are:

1. Make any appropriate code changes as follows:
   - Remove and replace all proprietary APIs and flags.
   - Remove implementation-specific hard-coded JNDI and JDBC references.
2. Adjust deployment properties as follows:
   - Recreate XML deployment descriptors where needed. See Section 5.2.3.
   - Customize runtime properties for the OC4J environment.
   - Re-map EJB JNDI names if necessary.
3. Save the updated EJB archive files (`.jar` and `.ear`).
4. Deploy the archive file to OC4J.

5.2.2 Migrating Entity EJBs

The migration of entity EJBs involves the "generic steps" of session EJB migration (Section 5.2.1) and the following steps specific to entity EJB migration:

1. Remove and replace any implementation-specific JNDI or data source lookups.
2. Rewrite any specific database deployment descriptors:
   - `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml` (for container-managed persistence) to `orion-ejb-jar.xml`
3. Remove and replace all proprietary APIs and flags for transaction management, locking, and caching.

In addition to the above steps, take into consideration the support for different EJB specification levels from one container to another. Differences in the specifications may require code changes. For example, in EJB 1.1 (supported by WebLogic Server 7.0), the `ejbRemove()` method of local entity EJBs requires `javax.rmi.RemoteException` to be thrown. Whereas for EJB 2.0 (supported by Oracle Application Server), `ejbRemove()` should only throw `javax.ejb.EJBException`. If the local entity EJB is migrated from WebLogic Server 7.0 to Oracle Application Server 10*g* Release 3 (10.1.3) without changing the

exception thrown by `ejbRemove()`, a compilation exception occurs when the EJB is deployed.

5.2.2.1 EJBs with Bean-Managed Persistence (BMP)

The steps for migrating EJBs with BMP are:

1. Check code for specific JNDI references:
   - Re-map each JNDI name as appropriate.
   - Walk-through the code of the entity bean class to check the data access code for any implementation-specific dependencies such as hard-coded JNDI environment properties or implementation-dependent issues such as data source JNDI names. Modify and regenerate the code and EJB archive file as required.
2. Adjust deployment properties as required.

   If using Oracle JDeveloper, import the EJB archive into EJB Configurator. Then:
   - Adjust deployment properties as required.
   - Save the updated EJB archive file.
   - Deploy to OC4J.
   - Re-map the EJB's JNDI name.
3. Generate the low-level container classes for OC4J.
4. Customize deployment-time properties of the EJB(s) if required.

The main single point for concern is JNDI context access and the data source lookup procedure. It is, therefore, necessary and advisable to:

- Ensure that the code that retrieves the JNDI context does not pass any implementation-specific properties to the `InitialContext` class constructor.
- Modify code that uses any hard-coded references to data source JNDI names so that the data source JNDI name is obtained indirectly by looking up a specific environment entry for this EJB. Doing so will later make it straightforward to amend the data source JNDI name in the EJB's deployment descriptor when required.

5.2.2.2 EJBs with Container-Managed Persistence (CMP)

With CMP entity beans, the EJB container is responsible for managing the persistent state of an object using O-R mappings between the attributes of the object that need to be persisted and the corresponding columns of a database table that hold this object's attribute values.

Unfortunately, the EJB specification makes no provisions for a standard way to define O-R mappings. Therefore, it is left to EJB container vendors to store this information in the EJB archive file using a proprietary format.

Consequently, O-R mapping definitions stored in the EJB archive file are not compatible between EJB vendors, and mapping information must be regenerated as part of the migration process.

The overall tasks for migrating EJBs with CMP are:

- Since O-R mapping definitions are vendor dependent, the mapping definitions in `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml` need to be

re-created in `orion-ejb-jar.xml`. Procedures are provided in Section 5.2.3, "Migrating Deployment Descriptors".

- Resolve differences in container-managed relationships (CMR) mapping.
- Configure OC4J data sources for persistence.
- Remove WebLogic Server-specific container stub and skeleton classes, and generate the equivalent OC4J stub & skeleton classes.

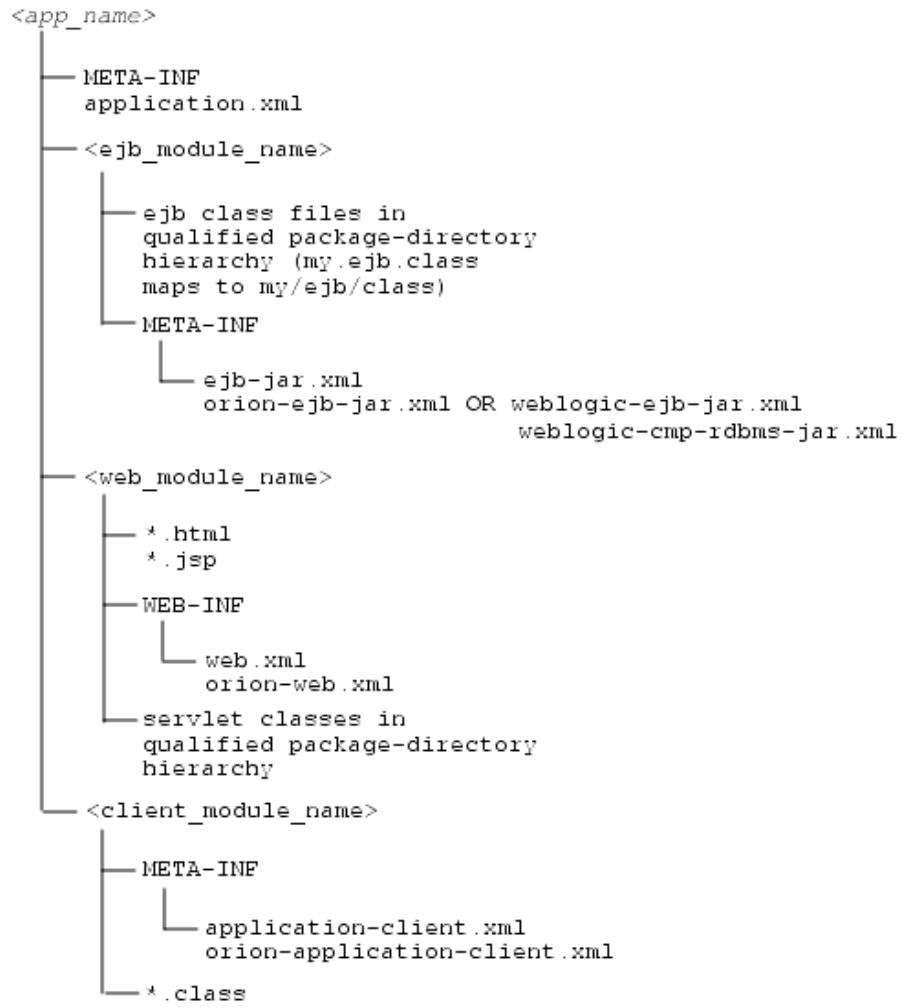
5.2.3 Migrating Deployment Descriptors

There are two deployment descriptors that are used to configure and deploy EJBs. The first deployment descriptor, `ejb-jar.xml`, is defined in the EJB specifications and provides a standardized format that describes an EJB application. The second deployment descriptor is a vendor-specific deployment descriptor that maps resources defined in the `ejb-jar.xml` file to resources in the application server. It is also used to define other aspects of the EJB container such as EJB behavior, caching, and vendor-specific features.

The WebLogic Server specific deployment descriptors are `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml`, and the OC4J-specific deployment descriptor is `orion-ejb-jar.xml`.

A typical J2EE application directory structure would look like this:

Figure 5–1 Directory Structure of a J2EE Application



The WebLogic Server-specific deployment descriptor, `weblogic-ejb-jar.xml`, defines EJB deployment descriptor DTDs which are unique to WebLogic Server. The DTD for `weblogic-ejb-jar.xml` includes elements for enabling stateful session EJB replication, configuring entity EJB locking behavior, and assigning JMS Queue and Topic names for message-driven beans

Elements configured in the EJB `weblogic-ejb-jar.xml` include:

- `weblogic-enterprise-bean`
  - `ejb-name`
  - `entity-descriptor`
  - `stateless-session-descriptor`
  - `stateful-session-descriptor`
  - `message-driven-descriptor`
  - `transaction-descriptor`
  - `reference-descriptor`
  - `enable-call-by-reference`
  - `jndi-name`

- `Security-role-assignment`
- `transaction-isolation`

The WebLogic Server-specific deployment descriptor, `weblogic-cmp-rdbms-jar.xml`, defines deployment properties for an entity EJB that uses WebLogic Server RDBMS-based persistence services.

Each `weblogic-cmp-rdbms-jar.xml` defines the following persistence options:

- EJB connection pools or data source for CMPs
- EJB field-to-database-element mappings
- Foreign key mappings for relationships
- WebLogic Server-specific deployment descriptors for queries

The OC4J-specific deployment descriptor, `orion-ejb-jar.xml`, contains extended deployment information for session beans, entity beans, message driven beans, and security.

An entity EJB can save its state in any transactional or non transactional persistent storage (bean-managed persistence), or it can ask the container to save its non-transient instance variables automatically (container-managed persistence). WebLogic Server and OC4J allow both choices and a mixture of the two.

In the case of an EJB that uses container-managed persistence, the `weblogic-ejb-jar.xml` or the `orion-ejb-jar.xml` deployment descriptor file specifies the type of persistence services that an EJB uses. In the case of WebLogic Server, the automatic persistence services requires the use of additional deployment files to specify their deployment descriptors, and to define entity EJB finder methods. WebLogic Server RDBMS-based persistence services obtain deployment descriptors and finder definitions from a particular bean using the bean's `weblogic-cmp-rdbms-jar.xml` file. This configuration file must be referenced in the `weblogic-ejb-jar.xml` file. In the case of OC4J, the type of persistence service as well as the details regarding the RDBMS-based persistence services are configured and obtained from the same deployment descriptor - `orion-ejb-jar.xml`.

Standard J2EE descriptors in `ejb-jar.xml` need little changes and should are migrated easily. The implementation-specific J2EE descriptors in the following files need to be modified:

- `weblogic-ejb-jar.xml`
- For CMP EJBs, descriptors in `weblogic-cmp-rdbms-jar.xml` need to be ported to `orion-ejb-jar.xml` (OC4J-specific).

Definitions of OC4J-specific XML descriptors can be found in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* or through online DTD files. These files are at:

```
http://xmlns.oracle.com/ias/dtds/orion-application.dtd
http://xmlns.oracle.com/ias/dtds/orion-application.dtd
http://xmlns.oracle.com/ias/dtds/orion-ejb-jar.dtd
http://xmlns.oracle.com/ias/dtds/orion-web.dtd
http://xmlns.oracle.com/ias/dtds/orion-application.dtd
```

Migration of deployment descriptors can be accomplished through one of the following ways:

- Manual creation of platform-specific deployment descriptor files using a text editor of your choice.

- Extensible Stylesheet Language Transformation (XSLT) - Oracle provides BEA, JBoss & Borland. Oracle's XSLT transformers are sample code and results will need some modification.
- Use Oracle JDeveloper to assist in the authoring of deployment descriptors. Reverse engineer the application to be migrated into Oracle JDeveloper. Appropriate descriptors will be automatically generated by Oracle JDeveloper.

> **Note:** For JBuilder, Oracle supplies an add-in which generates OC4J deployment descriptors.

5.2.3.1 Steps for Using Oracle JDeveloper 10*g* (10.1.3) to Convert `weblogic-ejb-jar.xml` to `orion-ejb-jar.xml`

The following steps convert most of the descriptor elements in `weblogic-ejb-jar.xml` and `weblogic-cmp-rdbms-jar.xml`:

1. In the File menu, select Import and specify the EAR file containing the WebLogic Server EJB's to be migrated.
2. Once the import operation has successfully completed, right-click `weblogic-ejb-jar.xml` and select "Export to OC4J."
3. Right-click the generated `orion-ejb-jar.xml`, and select "Properties."
4. Edit any mappings, if required.
5. Create a connection to an OC4J instance using the Connection Navigator. (Select "Connection Navigator" in the View menu.)
6. Create an EJB deployment profile. In the Application Navigator, right-click `ejb-jar.xml` and select "Create EJB Deployment Profile."
7. Deploy the EJBs to your OC4J instance. Right-click the deployment profile you created in step 6, and select your application server connection.

> **Note:** Use the above same steps to convert `weblogic-cmp-rdbms-jar.xml`. The same `orion-ejb-jar.xml` file is generated for both WebLogic Server descriptor files.

5.2.3.2 Using the Oracle Application Server TopLink Migration Tool to Convert `weblogic-cmp-rdbms-jar.xml` to `toplink-ejb-jar.xml`

The OracleAS TopLink migration tool takes the following files as input:

- `weblogic-ejb-jar.xml`
- `weblogic-cmp-rdbms-jar.xml`

It migrates as much WebLogic Server-specific persistence configuration as possible to a new `toplink-ejb-jar.xml` file and outputs the following new files in a target directory you specify:

- `orion-ejb-jar.xml`
- `toplink-ejb-jar.xml`
- OracleAS TopLink Mapping Workbench project file `TLCmpProject.mwp`

The input BEA WebLogic files may be in an EAR, JAR, or just standalone XML files. If you migrate from standalone XML files (rather than an EAR or JAR file), ensure that the domain classes are accessible and included in your classpath.

The OracleAS TopLink migration tool outputs a new `orion-ejb-jar.xml` and `toplink-ejb-jar.xml` file to the target directory you specify in the same format as it reads the original files. For example, if you specify an EAR file as input, then the OracleAS TopLink migration tool stages and creates a new EAR file that contains both the new `orion-ejb-jar.xml` and the new `toplink-ejb-jar.xml` file but is otherwise identical to the original.

The OracleAS TopLink Mapping Workbench project file is always output as a separate file.

Note: Oracle recommends that you make a backup copy of your `weblogic-ejb-jar.xml`, `weblogic-cmp-rdbms-jar.xml`, and `toplink-ejb-jar.xml` files before using the OracleAS TopLink migration tool.

As it operates, the OracleAS TopLink migration tool logs all errors and diagnostic output to a log file named wls_migration.log in the output directory. If you use the OracleAS TopLink migration tool from the OracleAS TopLink Mapping Workbench, also see the OracleAS TopLink Mapping Workbench log file `oracle.toplink.workbench.log` located in your user home directory (for example, in Windows, `C:\Documents and Settings\<username>`).

The OracleAS TopLink migration tool processes descriptor, mapping, and query information from the input files:

- It builds a OracleAS TopLink descriptor object for each entity bean and migrates native persistence metadata like mapped tables, primary keys, and mappings for CMP and CMR fields.
- It builds a OracleAS TopLink mapping object for every CMP and CMR field of an entity bean and migrates native persistence metadata like foreign key references.
- It builds a OracleAS TopLink query object for each finder or `ejbSelect` of an entity bean and migrates persistence metadata like customized query statements.

See Also: *Oracle Application Server TopLink Application Developer's Guide* for steps on using the OracleAS TopLink Migration Tool from OracleAS TopLink Mapping Workbench.

5.2.4 Generating and Deploying EJB Container Classes

The next step after compiling the EJB classes and adding the required XML deployment descriptors (the J2EE deployment descriptor as well as the vendor-specific deployment descriptors) is generation of the container classes that are used to access the EJB. The container classes include implementation of the external interfaces (home and remote) that clients use, as well as the classes that the application server uses, for the internal representation of the EJBs.

5.2.4.1 WebLogic Server

In WebLogic Server, you would have used the `appc` compiler to generate container classes according to the deployment properties specified in the WebLogic Server-specific XML deployment files. For example, if you indicate that your EJBs are

to be used in a cluster, `appc` creates special cluster-aware classes that will be used for deployment. You can also use `appc` directly from the command line by supplying the required options and arguments.

Once the container classes have been generated, you need to package the classes into a JAR or EAR file and deploy the classes using the console GUI.

5.2.4.2 OC4J

For OC4J, explicit compilation is not required. The EJB JAR file is packaged into a EAR file (together with a WAR file, if any). Then, you can use the Application Server Control Console GUI to specify the EAR file for deployment. The container classes are generated for OC4J and any J2EE Web application in the EAR file is bound to the OC4J container.

5.2.5 Loading EJB Classes in the Server

This section describes how each application server manages the loading of EJB classes.

5.2.5.1 WebLogic Server

The final step in deploying an EJB involves loading the generated container classes into WebLogic Server. However, you can prompt WebLogic Server to automatically load EJB classes by starting WebLogic Server. This places the EJB in the deployment directory where it is automatically deployed when the server is started.

5.2.5.2 OC4J

Similarly, you can specify classes belonging to an application to be loaded when OC4J starts by specifying the `auto-start="true"` parameter in the `<application>` tag in `server.xml`.

5.3 Migrating EJBs in a EAR or JAR File

EAR and JAR files containing EJBs which are deployed in WebLogic Server can be migrated to Oracle Application Server. However, you should unarchive and rearchive the EAR file to ensure its contents are complete and that the XML descriptors have the correct entries (using Oracle JDeveloper is another option). Use the following points as guidelines:

- Ensure that the EJB client XML descriptors specify the JNDI names of the EJB stubs. If the client is a Web application, the JNDI names should be specified in `web.xml`. If the client is standalone, the names should be specified in `application-client.xml`.
- For the case where the EJB client is standalone, the client classes and XML descriptor file, `application-client.jar`, should be archived into a JAR file, which in turn should be archived into the EAR file where the EJBs are.
- If the EJB(s) to be migrated from WebLogic Server are in a JAR file, you need to repackage them in a EAR file with the EAR's `application.xml`.
- Deploy the EAR file on Oracle Application Server using Application Server Control Console.
- You do not need to pre-compile EJB stubs using `appc`, `rmic`, or other such facilities into the client application. The OC4J EJB container generates EJB stubs on demand as it needs them.

5.4 Migrating an Exploded EJB Application

EJB applications can also be deployed as a collection of files that use a standard directory structure defined in the J2EE specification. This type of deployment deploys applications in an exploded directory format. Deploying an EJB application in exploded directory format is done most often whilst developing your application and only for standalone OC4J instances. This is because the exploded directory format is more suitable for developers to modify source files and test the application quickly. In Oracle Application Server production environments, however, the application should be packaged in a EAR file and deployed using Application Server Control Console.

When deploying an exploded directory structure to WebLogic Server, you would have copied the top level directory containing an EJB application in exploded directory format into the `mydomain/config/applications/` directory of your WebLogic Server distribution (where `mydomain` is the name of your WebLogic Server domain). Once copied, WebLogic Server automatically deploys the EJB application.

For OC4J, copy the top level directory containing the EJB application in exploded directory format into the following directory in your OC4J installation:

UNIX:
<*ORACLE_HOME*>`/j2ee/home/applications/`

Windows:
<*ORACLE_HOME*>`\j2ee\home\applications\`

Then, modify the default J2EE application deployment descriptor, `server.xml`, located in the <*ORACLE_HOME*>`/j2ee/home/config/` directory in UNIX, or <*ORACLE_HOME*>`\j2ee\home\config\` in Windows, to include your EJB module.

In WebLogic Server, if a file is modified using the administration console, or otherwise, it requires a server restart before the updated configuration is picked up. In the case of OC4J, the timestamp change for `server.xml` will cause OC4J to effect the changes in the XML file.

5.5 Writing Finders for RDBMS Persistence

For EJBs that use RDBMS persistence, WebLogic Server provides a way to write dynamic finders. The EJB provider writes the method signature of a finder in the `EJBHome` interface, and defines the finder's query expressions in the `ejb-jar.xml` deployment file. The `appc` compiler creates implementations of the finder methods at deployment time, using the queries in `ejb-jar.xml`.

The key components of a finder for RDBMS persistence are:

- The finder method signature in `EJBHome`
- A query stanza defined within `ejb-jar.xml`
- An optional WebLogic Server query stanza within `weblogic-cmp-rdbms-jar.xml`

OC4J simplifies the whole process by automatically generating the finder methods.

Specifying the `findByPrimaryKey` method is easy to do in OC4J. All the fields for defining a simple or complex primary key are specified within the `ejb-jar.xml` deployment descriptor. To define other finder methods in a CMP entity bean, do the following:

1. Add the finder method to the home interface

2. Add the finder method definition to the OC4J-specific deployment descriptor—the `orion-ejb-jar.xml` file

5.5.1 Migrating Finder Methods

The following considerations apply when migrating finder methods:

- Standard finder methods are automatically generated in OC4J; they do not need to be regenerated.
- Finder methods in WebLogic Server 5.1 and 6.0 use WLQL (WebLogic Query Language), a proprietary query language for specifying selection criteria. These need to be rewritten.
- OC4J uses standard SQL `WHERE` clauses for specifying selection criteria or EJB Query Language (EJB-QL).
- Any extensions of EJB-QL made by WebLogic Server need to be rewritten.

The EJB 1.1 specification does not fully address the particulars for custom finder methods, that is, the logic used within an EJB to find elements in a database. While the specification mandates that such methods be declared with names beginning with `find...()` or `findBy...()` in the home interface and bean class, it does not however provide a formal syntax to declare the underlying search logic. In other words, the way in which queries for custom finders are declared is not standardized, and is therefore dependent upon the EJB container. Additionally, custom finder methods may return either a single entity bean or a collection of entity beans, depending on the desired functionality.

Search logic in WebLogic Server is expressed using WLQL, which uses a syntax close to that of LISP; query operators and operands are presented in the form:

(operator operand1 operand2)

Search Logic in WebLogic Server

This language allows the definition of queries featuring multiple selection criteria (the equivalent of the `WHERE` clause in SQL) and optionally specifying a sorting clause (the equivalent of the `ORDER BY` clause in SQL).

Search Logic in Oracle Application Server

In contrast, search logic in Oracle Application Server is expressed using standard SQL `WHERE` clauses allowing multiple selection criteria.

For the `LIKE` operator with input parameters and `ORDER BY` clause, you can use Oracle Application Server 10*g* Release 3 (10.1.3) with OracleAS TopLink CMP that has support for EJB 2.1. If you do not use OracleAS TopLink, you can use a work around by modifying the SQL in '`query=""`' in `orion-ejb-jar.xml`.

5.6 WebLogic Query Language (WLQL) and EJB Query Language (EJB-QL)

In WebLogic Server 5.1 and 6.0, each finder query stanza in the `weblogic-cmp-rdbms-jar.xml` file had to include a WLQL string that defines the query used to return EJBs. These releases of WebLogic Server implemented an EJB 1.1 container and did not support standardized EJB-QL.

With the emergence of EJB Query Language, which is a standard based on the EJB 2.0 specification, use of WLQL is deprecated. With WebLogic Server 7.0 and 8.1, their EJB

containers are EJB 2.0 compliant and supports EJB-QL. These EJB containers additionally provide a WLQL extension to EJB-QL. This extension is proprietary to WebLogic Server.

Oracle Application Server provides complete support for EJB-QL including the following features:

- Automatic Code Generation: EJB-QL queries are defined in the deployment descriptor of the entity bean. When the EJBs are deployed to Oracle Application Server, the container automatically translates the queries into the SQL dialect of the target data store. Because of this translation, entity beans with container-managed persistence are portable -- their code is not tied to a specific type of data store.
- Optimized SQL Code Generation: Further, in generating the SQL code, Oracle Application Server makes several optimizations such as the use of bulk SQL and batched statement dispatch to make database access efficient.
- Support for Oracle and Non-Oracle Databases: Oracle Application Server provides the ability to execute EJB-QL against any database - Oracle, MS SQL-Server, IBM DB/2, Informix, and Sybase.
- CMP with Relationships: Oracle Application Server supports EJB-QL for both single entity beans and also with entity beans that have relationships, with support for any type of multiplicity and directionality.

For more information on EJB-QL in Oracle Application Server, refer to *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*.

5.7 Message Driven Beans

In WebLogic Server, in addition to the new `ejb-jar.xml` elements, the `weblogic-ejb-jar.xml` file includes only one new message-driven-descriptor stanza to associate the message-driven bean with an actual destination in WebLogic Server. The XML element is `destination-jndi-name`.

In OC4J, to create a message-driven bean, you perform the following steps:

1. Implement a message-driven bean as defined in the EJB specification
2. Create the message-driven bean deployment descriptors
3. Configure the JMS `Destination` type (queue or topic) in the OC4J JMS XML file, `jms.xml`.
4. Map the JMS `Destination` type to the message-driven bean in the OC4J-specific deployment descriptor, `orion-ejb-jar.xml`
5. If a database is involved in your message-driven bean application, configure the data source that represents your database in `data-sources.xml`.
6. Create an EJB JAR file containing the bean and the deployment descriptor; once created, configure the `application.xml` file, create an EAR file, and deploy the EJB in OC4J.

6

Migrating JDBC

This chapter provides the information you need to migrate database access code from WebLogic Server to Oracle Application Server. It addresses the migration of JDBC drivers, data sources, and connection pooling.

This chapter contains these topics:

6.1 Introduction

Migrating applications deployed on WebLogic Server that use JDBC, specifically WebLogic JDBC drivers, to OC4J and Oracle JDBC drivers can be straightforward, requiring little or no code changes to the applications migrated. All applications written to the standard JDBC specifications will work correctly and require minimal migration effort. The primary effort goes into configuring and deploying the applications in the new environment. Only in cases where proprietary extensions are used will the migration effort get complex.

6.1.1 Differences between WebLogic and Oracle Application Server Database Access Implementations

WebLogic Server 8.1 supports J2EE 1.3 (JDBC 2.0) and OC4J supports J2EE 1.4 (JDBC 2.1). The JDBC drivers from BEA as well as Oracle support the same version of the JDBC standard - version 2.0 specifications. Therefore, the differences between the two servers should be minimal, often differing primarily in the area of proprietary extensions. Before analyzing any differences, an overview of JDBC Drivers is apt.

6.1.1.1 Overview of JDBC Drivers

JDBC defines standard API calls to a specified JDBC driver, a piece of software that performs the actual data interface commands. The driver is considered the lower level JDBC API. The interfaces to the driver are database client calls, or database network protocol commands that are serviced by a database server.

Depending on the interface type, there are four types of JDBC drivers that translate JDBC API calls:

- **Type 1, JDBC-ODBC Bridge**—Translates calls into ODBC API calls.
- **Type 2, Native-API Driver**—Translates calls into database native API calls. As this driver uses native APIs, it is vendor dependent. The driver consists of two parts: a Java language part that performs the translation, and a set of native API libraries.
- **Type 3, Net-Protocol**—Translates calls into DBMS-independent network protocol calls. The database server interprets these network protocol calls into specific DBMS operations.
- **Type 4, Native-Protocol**—Translates calls into DBMS native network protocol calls. The database server converts these calls into DBMS operations.

BEA provides a variety of options for database access using the JDBC API specification. These options include WebLogic jDrivers for the Oracle, Microsoft SQL Server, and Informix database management systems (DBMS). In addition to the Type 2 WebLogic jDriver for Oracle, WebLogic provides a Type 2 driver for Oracle XA and three Type 3 drivers - RMI Driver, Pool Driver and JTS.

Similarly, Oracle Application Server provides a variety of options for database access, particularly the best JDBC drivers for the Oracle database, and JDBC drivers from partner Merant for accessing several other databases including DB2.

- **WebLogic jDriver for Oracle**—The WebLogic jDriver for Oracle provides connectivity to the Oracle database and requires an Oracle client installation since it is based on OCI (Oracle Call Interface API). The WebLogic jDriver for Oracle XA driver extends the WebLogic jDriver for Oracle for distributed transactions.

  The Oracle thick or JDBC OCI driver is the equivalent of WebLogic jDriver for Oracle as well as WebLogic jDriver for Oracle XA since the JDBC OCI driver provides XA functionality.
- **WebLogic Pool Driver**—The WebLogic Pool driver enables utilization of connection pools from server-side applications such as HTTP servlets or EJBs.
- **Oracle JDBC-OCI Driver**—The Oracle JDBC-OCI driver allows J2EE applications to use connection pools. This driver supports JDBC 2.0 connection pool features fully.
- **WebLogic RMI Driver**—The WebLogic RMI driver is a multitier, Type 3, Java Data Base Connectivity (JDBC) driver that runs in WebLogic Server and can be used with any two-tier JDBC driver to provide database access. Additionally, when configured in a cluster of WebLogic Servers, the WebLogic RMI driver can be used for clustered JDBC, allowing JDBC clients the benefits of load balancing and fail over provided by WebLogic Clusters.
- **WebLogic JTS Driver**—The WebLogic JTS driver is a multitier, Type 3, JDBC driver used in distributed transactions across multiple servers with one database instance. The JTS driver is more efficient than the WebLogic jDriver for Oracle XA driver when working with only one database instance because it avoids two-phase commit.
- **Oracle Thin Driver**—The two-tier Oracle Thin Type 4 driver provides connectivity from WebLogic Server to Oracle DBMS.

  If you are already using the Oracle OCI or Oracle thin JDBC drivers from your WebLogic Server, your code will not require any changes and you can move to the section on configuring data-sources in OC4J.

6.2 Migrating Data Sources

The JDBC 2.0 specification introduced the `java.sql.Datasource` class to make the JDBC program 100% portable. In this version, the vendor-specific connection URL and machine and port dependencies were removed. This version also discourages using `java.sql.DriverManager`, `Driver`, and `DriverPropertyInfo` classes. The data source facility provides a complete replacement for the previous JDBC `DriverManager` facility. Instead of explicitly loading the driver manager classes into the client applications runtime, the centralized JNDI service lookup obtains the `java.sql.Datasource` object. The `Datasource` object can also be used to connect to the database. According to the JDBC 2.0 API specification, a data source is registered under the JDBC subcontext or one of its child contexts. The JDBC context itself is registered under the root context. A `DataSource` object is a connection factory to a data source.

WebLogic and OC4J both support the JDBC 2.0 data source API. A J2EE server implicitly loads the driver based on the JDBC driver configuration, so no client-specific code is needed to load the driver. The JNDI (Java Naming and Directory Interface) tree provides the `DataSource` object reference.

6.2.1 Data Source Import Statements

`DataSource` objects, along with JNDI, provide access to connection pools for database connectivity. Each data source requires a separate `DataSource` object, which may be implemented as a `DataSource` class that supports either connection pooling or distributed transactions.

To use the `DataSource` objects, import the following classes in your client code:

```
import java.sql.*;
import java.util.*;
import javax.naming.*;
```

In the case of WebLogic Server, you would use the `weblogic.jdbc.*` packages and in the case of OC4J, you would use `oracle.jdbc.*` packages.

6.2.2 Configuring Data Sources in the Application Server

For Oracle Application Server, you configure data sources using the Application Server Control Console Web pages to specify the data source name, database name and JDBC URL string. You can also define multiple data sources to use a single connection pool, thereby allowing you to define both transaction and non-transaction-enabled `DataSource` objects that share the same database.

The best way to configure and define data sources is through Application Server Control Console. However, in this document we will examine the underlying infrastructure and focus on direct manipulation of the configuration files. OC4J uses flat files to configure data sources for all of its deployed applications. Data sources are specified in the <*ORACLE_HOME*>`/j2ee/home/config/data-sources.xml` file. Following is an sample data source configuration for an Oracle database. Each data source specified in `data-sources.xml` (`xa-location`, `ejb-location` and `pooled-location`) must be unique.

```
<data-source
class="com.evermind.sql.DriverManagerDataSource"
name="Oracle"
url="jdbc:oracle:thin@<database host name><database listener port
number>:<database SID>"
pooled-location="jdbc/OraclePoolDS"
```

```
xa-location="jdbc/xa/OracleXADS"
ejb-location="jdbc/OracleDS"
connection-driver="oracle.jdbc.driver.OracleDriver"
username="scott"
password="tiger"
url="jdbc:oracle:thin@<database host name><database listener port
number>:<database SID>"
schema="database-schemas/oracle.xml"
inactivity-timeout="30"
max-connections="20"
/>
```

Table 6–1 describes all of the configuration parameters in `data-sources.xml`. (Not all of the parameters are shown in the example above).

Table 6–1 Configuration Parameters in `data-sources.xml` File

Parameter	Description
`class`	Class name of the data source.
`connection-driver`	Class name of the JDBC.
`connection-retry-interval`	Number of seconds to wait before retrying a failed connection. Default value is 1 second.
`ejb-location`	JNDI path for binding an EJB-aware, pooled version of this data source; this version will participate in container-managed transactions. This is the type of data source to use from within EJBs and similar objects. This parameter only applies to a `ConnectionDataSource`.
`inactivity-timeout`	Number of seconds unused connections should be cached before being closed.
`location`	JNDI path for binding this data source.
`max-connect-attempts`	Number of times to retry a failed connection. Default is 3 times.
`max-connections`	Maximum number of open connections for pooling data sources.
`min-connections`	Minimum number of open connections for pooling data sources. The default is zero.
`name`	Displayed name of the data source.
`password`	User password for accessing the data source (optional).
`pooled-location`	JNDI path for binding a pooled version of this data source. This parameter only applies to a `ConnectionDataSource`. Relative or absolute path to a database-schema file for the database connection.
`source-location`	Underlying data source of this specialized data source.
`url`	JDBC URL for this data source (used by some data sources that deal with `java.sql.Connections`.
`username`	User name for accessing the data source (optional).
`wait-timeout`	Number of seconds to wait for a free connection if all connections are used. Default is 60.

Table 6–1 (Cont.) Configuration Parameters in `data-sources.xml` File

Parameter	Description
`xa-location`	JNDI path for binding a transactional version of this data source. This parameter only applies to a `ConnectionDataSource`.
`xa-source-location`	Underlying `XADataSource` of the specialized data source (used by `OrionCMTDataSource`).

6.2.3 Obtaining a Client Connection Using a Data Source Object

To obtain a connection from a JDBC client, you would use JNDI to look up and locate the `DataSource` object. This is illustrated in the following code fragment where you obtain a connection in WebLogic Server:

```
try
{
    java.util.Properties parms = new java.util.Properties();
    parms.setProperty(Context.INITIAL_CONTEXT_FACTORY,
    "weblogic.jndi.WLInitialContextFactory");

    javax.naming.Context ctx = new javax.naming.InitialContext(parms);
    javax.sql.DataSource ds = (javax.sql.DataSource)ctx.lookup("jdbc/SampleDB");
    java.sql.Connection conn = ds.getConnection();

    // process the results
    ...
}
```

To migrate the above code from WebLogic Server to OC4J, you need to change the class that implements the initial context factory (`Context.INITIAL_CONTEXT_FACTORY`) of the JNDI tree from `weblogic.jndi.WLInitialContextFactory`, which is the WebLogic-specific class, to `com.evermind.server.ApplicationClientInitialContextFactory`, which is the OC4J specific class.

With this change, your code is ready for deployment on OC4J and to use the Oracle JDBC drivers.

> **See Also:** Section 5.1.2.1, "Global JNDI Lookups and Oracle Application Server"

6.3 Migrating Connection Pools

Most Web-based resources, such as servlets and application servers, access information in a database. Each time a resource attempts to access a database, it must establish a connection to the database, consume system resources to create the connection, maintain it, and then release it when it is no longer in use. The resource overhead is particularly high for Web-based applications, because of the frequency and volume of Web users connecting and disconnecting. Often, more resources are consumed in connecting and disconnecting than in the interactions themselves.

Connection pooling enables you to control connection resource usage by spreading the connection overhead across many user requests. A connection pool is a cached set of connection objects that multiple clients can share when they need to access a database resource. The resources to create the connections in the pool are expended only once for a specified number of connections, which are left open and reused by many client requests, instead of each client using resources to create its own connection and closing

it after its database operation is complete. Connection pooling improves overall performance in the following ways:

- Reducing the load on the middle tier and server
- Minimizing resource usage by session create and session close operations
- Eliminating bottlenecks caused by socket and file descriptor limitations and 'n' user license limitations.

The JDBC 2.0 specification allows you to define a pool of JDBC database connections with the following objectives:

- Maximize the availability of connections to resources.
- Minimize the idle connections in the pool.
- Return orphan connections to the pool and make them available for reuse by other servlets or application servers.

To meet these objectives, you:

1. Set the maximum connection pool size property equal to the maximum number of concurrently active user requests expected.
2. Set the minimum connection pool size property equal to the minimum number of concurrently active user requests expected.

The connection pooling properties ensure that as the number of user requests decreases, connections are gradually removed from the pool. Likewise, as the number of user requests begins to grow, new connections are created. The balance of connections is maintained so that connection reuse is maximized and connection creation overhead minimized. You can also use connection pooling to control the number of concurrent database connections.

6.3.1 Overview of Connection Pools

Connection pools provide ready-to-use pools of connections to your DBMS. Since these database connections are already established when the connection pool starts up, the overhead of establishing database connections is eliminated. You can utilize connection pools from server-side applications such as HTTP servlets or EJBs using the pool driver or from standalone Java client applications.

One of the greatest advantages of connection pooling is that it saves valuable program execution time and has almost no or very low overhead. Making a DMBS connection is very slow. With connection pools, connections are established and available to users before they are needed. The alternative is for application code to make its own JDBC connections when needed. A DBMS runs faster with dedicated connections than if it has to handle incoming connection attempts at runtime.

6.3.2 How Connection Pools Enhance Performance

Establishing a JDBC connection with a DBMS can be very slow. If your application requires database connections that are repeatedly opened and closed, this can become a significant performance issue. WebLogic Server and Oracle Application Server connection pools offer an solution to this problem.

When WebLogic Server or Oracle Application Server starts, connections from the connection pools are opened and are available to all clients. When a client closes a connection from a connection pool, the connection is returned to the pool and becomes available for other clients; the connection itself is not closed. There is little cost to "open" and "close" pool connections.

How many connections should you create in the pool? A connection pool can grow and shrink according to configured parameters, between a minimum and a maximum number of connections. The best performance will always be when the connection pool has as many connections as there are concurrent users.

6.4 Overview of Clustered JDBC

Relevant only in multitier configurations, clustered JDBC allows external JDBC clients to reconnect and restart their JDBC connection without changing the connection parameters, in case a serving cluster member fails. For WebLogic, clustered JDBC requires data source objects and the WebLogic RMI driver to connect to the DBMS. Data source objects are defined for each WebLogic Server using the WebLogic Administration Console.

Oracle provides functionality that is similar to and more advanced than that provided by the clustered JDBC by leveraging the TAF capabilities of OCI.

6.5 Performance Tuning JDBC

Performance tuning your JDBC application in OC4J is similar to that for WebLogic Server. Connection pooling helps improve performance by avoiding the expensive operation of creating new database connections. The guidelines on writing efficient code hold true for Oracle Application Server and WebLogic Server.

A

Additional Features

This appendix provides additional comparative information between WebLogic Server 7.0 and Oracle Application Server 10*g*. This information consists of:

- Section A.1, "Migrating Web Services"
- Section A.2, "Java Messaging Service (JMS)"
- Section A.3, "Oracle TopLink"

A.1 Migrating Web Services

Before JAX-RPC was formalized as a mandatory J2EE 1.4 API, Web services were not standardized in the Java specifications. Each application server vendor had its own way of programming Web services (both server and client). Apache Axis, provided some portability between different vendors as it is a simple library that can run in any J2EE 1.3 compliant container. Axis is well adopted even though it lacks support for WS-* standards. If your WebLogic Web service is written using Apache Axis, you should be able to deploy it in Oracle Application Server 10*g* Release 3 (10.1.3) with little modification, if any.

Ideally, JAX-RPC as a standard should be portable. The implementation class, service endpoint interface, and the WSDL should be sufficient to deploy a Web service between different application servers. However, due to the complexity of the serialization/deserialization of SOAP, most of the time, a Web service needs to be "reassembled" from the top down. Oracle Application Server provides a utility to ease this task. The WebServicesAssembler utility accepts a WSDL and creates the requisite service endpoint interface (SEI) using its `genInterface` command. You can then fill in the implementation for the Web service for any required architecture, such as Java classes. To assemble the service, the utility is invoked with its `topDownAssemble` command. A EAR file is generated.

Even if a seamless migration is the ideal solution, "rewriting" Web services is not necessarily as complex as its appears. The critical parts to consider are the implementation class that contains the business logic and the WSDL that exposes this business logic as a Web service. Using the top down approach with the WebServicesAssembler utility allows a Web service's artifacts to be generated from a WSDL for Oracle Application Server.

> **See:** *Oracle Application Server Web Services Developer's Guide* for information on using the top down approach with WebServicesAssembler.

For migrating Web services that are compliant with J2EE 1.4 specifications to Oracle Application Server, only modifications to proprietary WebLogic Server API need to be

made. The SEI and `Impl` class can be used without modification. The WebLogic Server proprietary API are usually those that lookup data or objects, such as accessing a HTTP request from a Web service implementation object. The proprietary API would usually have package names starting with `weblogic.*` . They would have to be replaced with equivalent Oracle API.

A.2 Java Messaging Service (JMS)

Oracle Application Server 10*g* supports JMS 1.1 and WebLogic Server 7.0 supports JMS 1.0.2. Table A–1 highlights some of the key JMS features supported by both application servers.

Table A–1 JMS Feature Comparison Summary

Feature	Oracle Application Server 10*g*	WebLogic Server 7.0
Pluggable JMS Providers	Yes	Yes
Message Retention and Query Ability	Yes	Yes
Persistence of JMS Messages	Yes	Yes
Fail over of Persisted JMS Messages	Yes	No
Message Payloads: Structured Datatypes, Unstructured Datatypes, Relational Data, Text, XML, Objects, Multimedia Data	Yes	Yes
Message Transports: SOAP, Oracle Net	Yes	Yes
Secure Access	Yes	Yes
Abstraction of Business Logic, Rules, and Routing into Easily Maintainable Tables	Yes	No
Guaranteed Delivery	Yes	No
Ability to Cluster in a High Availability Configuration	Yes	No
Interfacing with Java and Non Java Clients	Yes	No

Oracle Application Server provides support for JMS in the following manner:

- *Fast, Lightweight, Compliant* - Oracle Application Server provides two out-of-the-box JMS implementations.

  The first is OracleJMS, which uses the Oracle databases integrated Advanced Queuing (AQ) to offer secure, transactional, recoverable, and guaranteed delivery of messages. Oracle Application Server also offers a fast and lightweight, in-memory JMS that can be used to pass messages between applications in the middle tier. In contrast, WebLogic provides a simple JMS implementation.

- *Pluggable JMS Providers* - Oracle Application Server J2EE applications can access queues and topics using the JMS API. They can use an Oracle Application Server

specific JNDI namespace to look up JMS `ConnectionFactories` and `Destinations`.

Oracle Application Server defines a `ResourceProvider` interface for plugging in message providers and provides the implementation classes for Oracle's Advanced Queuing and for third-party messaging systems such as MQSeries, SonicMQ and SwiftMQ. The `ResourceProvider` interface allows switching between message providers transparently to the JMS client. JMS clients can mix messages from multiple messaging systems in the same application, and switch between them by merely changing the JNDI mappings, and without any changes in the source code.

WebLogic has lesser support for plugging-in other JMS providers. Its approach to supporting IBM MQ Series is complicated. Developers need to use BEA WebLogic MQ Series JMS classes, a separate library of classes, to plug in MQ Series. Oracle Application Server, on the other hand, makes it extremely easy to plug in, almost as simple as a `DataSource`.

The following apply to Oracle Application Server 10*g* Release 3 (10.1.3):

- Supports JMS 1.1.
- Has generic JMS JCA 1.5 resource adaptor with:
  - support for WebSphereMQ, Tibco JMS, SonicMQ
  - full MDB support
  - full XA support
- JMX-based dynamic configuration:
  - no server restart for `Destination` and `ConnectionFactory` creation or deletion
  - no server restart for JMS server property changes
- JMS router in Oracle Application Server provides bridge for Oracle JMS, OracleAS JMS, WebSphereMQ, Tibco JMS, SonicMQ.
- Support for message filtering during routing of messages.

A.2.1 Oracle JMS (OJMS)

OJMS is the Java front-end for the Oracle database integrated Advanced Queuing (AQ), which offers secure, transactional, recoverable, guaranteed delivery of messages.

Advanced Queuing provides a number of important facilities. OJMS leverages the Oracle database robustness, query-ability and DML operations, scalability and high availability, and support for all data types in message payload, including relational data, text, XML, and multimedia.

The following general features are discussed:

- *Message Retention and Query Ability* - OJMS integrates a messaging system with the Oracle Database leveraging the databases robustness, and providing guaranteed message retention and auditing/tracking while eliminating the need for 2-PC operations between the messaging system and the database. Further, since the queues are stored in the Oracle Databases, they can be queried using standard SQL.
- *Message Payloads* - OJMS can support a variety of structured and unstructured datatypes as message payloads including relational data, text, XML, objects, and multimedia data.

- *Message Transports* - OJMS also provides support for reliable once-only, in-order delivery of messages over a variety of transports including SOAP, Oracle Net, and others. It can also use other messaging providers such as MQ-Series for transport.
- *Secure Access* - Finally, OJMS provides stringent access control on individual queues and messages using the databases ACL mechanisms.

WebLogic Server does not have the following set of capabilities that Oracle Messaging (JMS) provides:

- *Abstraction of Business Logic, Rules, and Routing into Easily Maintainable Tables* - OJMS has extensive rules functionality. Rules can be used for efficient routing of messages. You can specify rules as SQL expressions when defining your subscriptions. Rules engine performs efficient rules evaluation. These SQL expressions can contain any other PL/SQL, C or Java function. With the Oracle9*i* Database (Release 2), you can also organize your rules in rule-sets and use rules functionality independent of message queuing.
- *Guaranteed Delivery* - OJMS provides guaranteed once and only once delivery. This is a feature unique to OJMS. You can monitor messages in the queues using SQL views. You can write a single SQL statement to find out exact location of your message.
- *Ability to Cluster in a Highly Available Configuration* - With Oracle Advanced Queuing, you get benefits of the entire Oracle stack including persistence, high availability and scalability with Real Application Clusters (RAC).

  On the other hand, BEA WebLogic JMS clustering is not as robust. One of the key considerations in the choice of JMS is reliability. WebLogic Server does not have fail over of persisted messages pertaining to a server failure. Further, it provides fail over only for JMS destinations. However, these JMS destinations are not replicated. For example, you can deploy multiple persistent queues with the same JNDI name across all nodes of a cluster. A client will hit just one until that node fails, in which the cluster will transparently fail over the client to the next available node/queue.

  However, what's stored in the first queue remains in the first queue since it is persistent until someone manually brings that node up again. Or, you need to find some mechanism for retrieving the messages yourself. Oracle Application Server and Oracle AQ, on the other hand, can leverage the high availability capabilities of Oracle9*i* RAC to avoid this problem.
- *Interfacing with Java and Non Java Clients* - The WebLogic Server implementation of JMS does not allow sending messages to non Java clients. Oracle Application Server JMS, through Oracle AQ, has four APIs - PL/SQL, Java (JMS), C, and XML. This enables a message to be enqueued from any language and dequeued from any other language, thereby providing the flexibility to integrate with various heterogeneous systems including legacy systems.

A.3 Oracle TopLink

In an enterprise Java environment, one of the most formidable challenges is storing business objects and components in a relational database (RDB). Oracle TopLink makes application development more productive by offering an easy to use mapping workbench that maps the Java objects to relational databases and by simplifying one of the most difficult aspects of developing applications - persisting information to the database. Using Oracle TopLink, developers gain the flexibility to map objects and Enterprise Java Beans to a relational database schema with minimal impact on ideal application design or database integrity. The result: developers focused on addressing

business needs rather than building infrastructure. Oracle TopLink is built on JDBC and is portable across any JDBC-compliant database, including Oracles Database, DB2, SQL Server, Sybase, Informix, and Microsoft Access.

The Oracle TopLink solution offers three key benefits:

- *Mature Design, Flexibility and Performance* - Oracle TopLink provides a rich set of performance optimization and scalability features. Performance is addressed with caching techniques that minimize database and network traffic while always leveraging optimizations provided by JDBC and the databases.
- Simplified Application Development - Oracle TopLink makes application development more productive by offering an easy to use mapping workbench that maps the Java objects to relational databases and by simplifying one of the most difficult aspects of developing applications - persisting information to the database.
- Optimization of Resources - With Oracle TopLink, an application development team can focus on building the application rather than building infrastructure.

In essence, Oracle TopLink offers the best solution in the market to perform Java-to-relational database object-relational mapping. With Oracle TopLink, Oracle has blended the Java world and the relational database world in the best way possible, and solved one of the greatest challenges facing J2EE developers: productively mapping their Java objects and entity beans to a relational database.

With Oracle Application Server 10*g*, Oracle TopLink is an integrated component of Oracle Application Server. Specifically, the Oracle TopLink framework is integrated with the Oracle Containers for J2EE, and the Oracle TopLink Workbench is integrated with Oracle JDeveloper.

Index

O

P

R

S

T

W