Oracle® Application Server 10*g*

Migrating From WebLogic

10*g* (9.0.4)

Part No. B10425-01

November 2003

ORACLE®

Oracle Application Server 10*g* Migrating From WebLogic, 10*g* (9.0.4)

Part No. B10425-01

Primary Author: Kai Li

Contents

Send Us Your Comments vii

Preface ix

Intended Audience ix
Documentation Accessibility ix
Organization x
Related Documents xi
Conventions xii

1 Overview

Overview of J2EE 1-2
What is the J2EE Application Model? 1-2
What is the J2EE Platform? 1-3
What is an Application Server? 1-4
Overview of Oracle Application Server 1-5
J2EE Application Migration Challenges 1-5
J2EE Application Architecture 1-6
Migration Issues 1-7
Migration Approach 1-8
Migration Effort 1-8
Using This Guide 1-8

2 Comparison of Oracle Application Server and WebLogic Server

Application Server Product Offerings 2-1

WebLogic 2-1
WebLogic Server 2-2
WebLogic Enterprise 2-2
WebLogic Express 2-3
Oracle Application Server 2-3
Architecture Comparison 2-4
WebLogic Server 2-4
Oracle Application Server Components and Concepts 2-6
Oracle Application Server Instance 2-6
Oracle HTTP Server 2-7
OC4J Instances 2-8
Oracle Process Management Notification (OPMN) Server 2-8
Distributed Configuration Management (DCM) 2-8
Oracle Application Server Web Cache 2-9
Oracle Enterprise Manager Application Server Control 2-9
Oracle Application Server Infrastructure 2-10
Oracle Application Server Metadata Repository 2-11
Oracle Identity Management 2-13
High Availability and Load balancing 2-14
WebLogic Server Support for High Availability and Load Balancing 2-14
HTTP Session State Load Balancing and Failover (Servlet Clustering) 2-15
EJB and RMI Object Load Balancing and Failover 2-15
Oracle Application Server Support for High Availability and Load Balancing 2-16
Oracle Application Server Instance 2-16
Oracle Application Server Clusters (Middle Tier) 2-17
OC4J Islands 2-18
Stateful Session EJB High Availability Using EJB Clustering 2-19
JNDI Namespace Replication 2-20
Java Object Cache 2-20
Oracle Application Server Web Cache Clusters 2-21
OracleAS Infrastructure High Availability Solutions 2-22
Oracle Application Server Cold Failover Clusters 2-22
Oracle Application Server Active Clusters 2-22
J2EE Support Comparison 2-23
Java Development and Deployment Tools 2-24

WebLogic Development and Deployment Tools 2-24
WebLogic Server Workshop 2-24
WebLogic Server Administration Console 2-24
Oracle Application Server Development and Deployment Tools 2-25
Development Tools 2-25
Assembly Tools 2-26
Administration Tools 2-26

3 Migrating Java Servlets

Introduction 3-2
Differences Between WebLogic Server and Oracle Application Server Servlet Implementations 3-2
OC4J Key Servlet Container Features 3-2
Migrating a Simple Servlet 3-2
Migrating a WAR File 3-6
Migrating an Exploded Web Application 3-8
Migrating Configuration and Deployment Descriptors 3-9
Oracle Application Server 3-9
WebLogic Server 3-12
Migrating Cluster Aware Applications 3-13

4 Migrating JSP Pages

Introduction 4-2
Differences Between WebLogic Server and Oracle Application Server JSP Implementations 4-2
OC4J JSP Features 4-3
Edge Side Includes for Java (JESI) Tags 4-4
Web Object Cache Tags 4-4
Oracle JDeveloper and OC4J JSP Container 4-4
Migrating a Simple JSP Page 4-5
Migrating a Custom JSP Tag Library 4-7
Migrating from WebLogic Custom Tags 4-11
WebLogic Server `cache` Tag 4-12
WebLogic Server `process` Tag 4-12
WebLogic Server `repeat` Tag 4-12

Precompiling JSP Pages 4-13
Using the WebLogic Server JSP Compiler 4-13
Using the OC4J JSP Pre-translator 4-14
Standard JSP Pre-translation Without Execution (based on the JSP 1.1 specification) 4-15
Configure the JSP Container for Execution with Binary Files Only 4-16

5 Migrating Enterprise JavaBean Components

Introduction 5-2
Comparison of WebLogic Server and Oracle Application Server EJB Features 5-2
The following sections go into detail on some of the abovementioned features: 5-3
More Efficient Container Managed Persistence 5-3
Clustering Support 5-4
Scalability and Performance Enhancements 5-5
Security and LDAP Integration 5-6
WebLogic Server Caveats 5-6
EJB Migration Considerations 5-6
Migration Steps 5-7
Setting Deployment Properties 5-7
Vendor-specific Deployment Descriptors 5-8
WebLogic Server 5-8
OC4J 5-8
Generating and Deploying EJB Container Classes 5-8
WebLogic Server 5-9
OC4J 5-9
Loading EJB Classes in the Server 5-9
WebLogic Server 5-9
OC4J 5-9
Migrating EJBs in a EAR or JAR File 5-9
Migrating an Exploded EJB Application 5-10
Configuring EJBs using Deployment Descriptors 5-11
Writing Finders for RDBMS Persistence 5-14
WebLogic Query Language (WLQL) and EJB Query Language (EJB QL) 5-15
Message Driven Beans 5-15
Configuring Security 5-16
Migrating Cluster-Aware EJB Applications to OC4J 5-16

EJB Clustering in WebLogic Server 5-16
In-Memory Replication for Stateful Session EJBs 5-16
Requirements and Configuration 5-17
EJB Clustering in Oracle Application Server 5-18
Load Balancing 5-18
Static Discovery 5-19
Dynamic Discovery 5-19
Failover 5-19
Stateless Session EJBs 5-19
Stateful Session EJBs 5-19
Entity EJBs 5-20
JNDI Namespace Replication 5-20

6 Migrating JDBC

Introduction 6-2
Differences between WebLogic and Oracle Application Server Database Access Implementations 6-2
Overview of JDBC Drivers 6-2
Migrating Data Sources 6-4
Data Source Import Statements 6-4
Configuring Data Sources in the Application Server 6-4
Obtaining a Client Connection Using a Data Source Object 6-7
Migrating Connection Pools 6-7
Overview of Connection Pools 6-8
How Connection Pools Enhance Performance 6-9
Overview of Clustered JDBC 6-9
Performance Tuning JDBC 6-9

A Additional Feature Comparisons

Java Messaging Service (JMS) A-1
OracleJMS (OJMS) A-3
Java Object Cache A-5
Dynamic Monitoring System (DMS) A-6
Active Components for J2EE (AC4J) A-6

Oracle Application Server TopLink (OracleAS TopLink).. A-8

Index

Send Us Your Comments

Oracle Application Server 10*g* Migrating From WebLogic, 10*g* (9.0.4)

Part No. B10425-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX: (650) 506-7375   Attn: Oracle Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Application Server Documentation
  500 Oracle Parkway, Mailstop 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This preface contains these topics:

- Intended Audience
- Documentation Accessibility
- Organization
- Related Documents
- Conventions

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

The following chapters make up this guide:

Chapter 1, "Overview"

This chapter provides an overview of the issues involved in migrating J2EE web applications from WebLogic Server 7.0 to Oracle Application Server, and the effort required.

Chapter 2, "Comparison of Oracle Application Server and WebLogic Server"

This chapter provides a comparison between Oracle Corporation's implementation of Sun Microsystems' J2EE platform and component specifications and that of BEA Systems.

Chapter 3, "Migrating Java Servlets"

This chapter provides the information you need to migrate Java servlets from WebLogic Server 7.0 to Oracle Application Server. It addresses the migration of simple servlets, WAR files, and exploded Web applications.

Chapter 4, "Migrating JSP Pages"

This chapter provides the information you need to migrate JavaServer pages from WebLogic Server 7.0 to Oracle Application Server. It addresses the migration of simple JSP pages, custom JSP tag libraries, and WebLogic custom tags.

Chapter 5, "Migrating Enterprise JavaBean Components"

This chapter provides the information you need to migrate Enterprise JavaBeans from WebLogic Server 7.0 to Oracle Application Server. It addresses the migration of stateful and stateless session beans and container-managed persistence and bean-managed persistence entity beans.

Chapter 6, "Migrating JDBC"

This chapter provides the information you need to migrate database access code from WebLogic Server 7.0 to Oracle Application Server. It addresses the migration of JDBC drivers, data sources, and connection pooling.

Appendix A, "Additional Feature Comparisons"

This appendix summarizes additional features between Oracle Application Server and WebLogic Server.

Related Documents

For more information, see these Oracle resources:

- Oracle Application Server Documentation Library
- Oracle Application Server Platform-Specific Documentation on Oracle Application Server Disk 1

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation/`

For additional information, see:

- `http://ibm.com/` for more information on WebLogic Server
- `http://java.sun.com/` for more information on J2EE

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Microsoft Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		
		`[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example

`...`	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		`CREATE TABLE ... AS` *`subquery`*`;`
		
		`SELECT` *`col1`*`,` *`col2`*`, ... ,` *`coln`* `FROM employees;`

. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*
		`DB_NAME =` *`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`
	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job='SALESMAN' and sal<1600\"` `C:\>imp SYSTEM/`*`password`* `FROMUSER=scott TABLES=(emp, dept)`
`HOME_NAME`	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_ BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ `C:\orant` for Windows NT ■ `C:\orawin95` for Windows 95 ■ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install Oracle9*i* release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora90`. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Starting for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_ HOME*`\rdbms\admin` directory.

1 Overview

This chapter provides an overview of the issues involved in migrating J2EE Web applications from WebLogic Server 7.0 to Oracle Application Server 10*g* (9.0.4), and the effort required.

The chapter contains these topics:

- Overview of J2EE
- What is an Application Server?
- Overview of Oracle Application Server
- J2EE Application Architecture
- Migration Issues
- Migration Effort
- Using This Guide

Overview of J2EE

The application server market is evolving rapidly. In particular, the most significant development over the last few years is the emergence of Sun Microsystems' Java 2 Platform, Enterprise Edition (J2EE) Specification that promises to create a level of cross-vendor standardization.

The J2EE platform and component specifications define, among other things, a standard platform for developing and deploying multi-tier, web-based, enterprise applications.

J2EE provides a solution to the problems encountered by companies moving to a multi-tier computing model. The problems addressed include reliability, scalability, security, application deployment, transaction processing, web interface design, and timely software development. It builds upon the Java 2 Platform, Standard Edition (J2SE) to enable Sun Microsystems' "Write Once, Run Anywhere" paradigm for multi-tier computing.

J2EE consists of the components described in Table 1–1:

Table 1–1 J2EE Standard Architecture Components

Component	Description
J2EE Application Model	An application model for developing multi-tier, thin client services
J2EE Platform	A platform for hosting J2EE applications
J2EE Compatibility Test Suite	A compatibility test suite for verifying that a J2EE platform product meets the requirements set forth in the J2EE platform and component specifications
J2EE Reference Implementation	A reference implementation of the J2EE platform

What is the J2EE Application Model?

The J2EE application model is a multi-tier application model. Application components are managed in the middle tier by containers. A container is a standard runtime environment that provides services, including life cycle management, deployment, and security services, to application components. This container-based model separates business logic from system infrastructure.

What is the J2EE Platform?

The J2EE platform consists of a runtime environment and a standard set of services that provide the necessary functionality for developing multi-tiered, web-based, enterprise applications.

The J2EE platform consists of the components described in Table 1–2.

Table 1–2 J2EE Platform Components

Component	Description
J2EE runtime environment	
Application components	
Application clients	A Java program, typically used for a GUI, that executes on a desktop computer
Applets	A component of a Java program that typically executes in a web browser
Servlets and JSP pages	Servlet: A Java program, used to generate dynamic content, that executes on a web server
	JSP page: A technology used to return dynamic content to a client, typically a web browser
Enterprise JavaBeans (EJB)	An applications architecture for component-based distributed computing
Containers	An entity that provides services for application components, including life cycle management, deployment, and security services
Resource manager drivers	A system-level component that enables network connectivity to external data sources
Database	A set of related files used for the storage of business data and accessible through the JDBC API
J2EE standard services	
HTTP	The standard protocol used by the Internet to send and receive messages between web servers and browsers

Table 1–2 J2EE Platform Components (Cont.)

Component	Description
HTTPS	A protocol used by the Internet to send and receive messages *securely* between web servers and browsers
Java Transaction API (JTA)	An API that allows applications and application servers to access transactions
RMI-IIOP	RMI: A protocol that enables Java objects to communicate remotely with other Java objects IIOP: A protocol that enables browsers and servers to exchange things other than text RMI-IIOP is a version of RMI that uses the CORBA IIOP protocol
JavaIDL	A standard language for interface specification primarily used for CORBA object interface definition
JDBC	An API that provides connectivity between databases and the J2EE platform
Java Message Service (JMS)	An API that enables the use of enterprise messaging systems
Java Naming and Directory Interface (JNDI)	An API that provides directory and naming services
JavaMail	An API that provides the ability to send and receive e-mail
JavaBeans Activation Framework (JAF)	An API required by the JavaMail API

What is an Application Server?

An application server is software that runs between web-based client programs and back-end databases and legacy applications. It helps separate system complexity from business logic, enabling developers to focus on solving business problems. An application server helps reduce the size and complexity of client programs by enabling these programs to share capabilities and resources in an organized and efficient way.

Application servers provide benefits in the areas of usability, flexibility, scalability, maintainability, and interoperability.

Overview of Oracle Application Server

Oracle Application Server is a comprehensive, integrated application server that provides all of the infrastructure and functionality needed to run every successful e-business. All development teams face a similar set of challenges—the need to rapidly deliver web sites and applications that run fast over any network and on every device; while providing business intelligence to support operational adjustments and strategic decisions. Oracle Application Server enables teams to address all of these e-business challenges.

Oracle Application Server has generated a great deal of interest in the application server market, and many organizations are embracing it to deploy their web-based, enterprise applications.

Oracle Application Server offers the only integrated infrastructure to develop, deploy, and secure web sites and applications. It provides a complete J2EE platform for developing enterprise Java applications. Oracle Application Server enables developers to develop web applications in any language including Java, Perl, PL/SQL, XML, and Forms. It enables the reduction of development and deployment costs through a single, unified platform for Java, XML, and SQL.

The J2EE server implementation in Oracle Application Server is called Oracle Application Server Containers for J2EE (OC4J). OC4J runs on the standard JDK and is extremely lightweight, provides high performance and scalability, and is simple to deploy and manage. With Oracle Application Server 10*g* (9.0.4), OC4J supports J2EE 1.3 APIs.

This migration guide seeks to help you understand the migration challenges you may face when migrating your J2EE applications from WebLogic Server 7.0 to Oracle Application Server 10*g* (9.0.4).

Note: In this document, where WebLogic Server is mentioned without a version number, WebLogic Server 7.0 is implied. Otherwise, for other versions of WebLogic Server, a version number is specifically mentioned.

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 failover implementations among application servers; these differences are sparsely documented, and are thus an even bigger challenge to the migration process.

These challenges make the migration path daunting, uncertain, and difficult to reliably plan and schedule. This chapter addresses the challenges in migrating your applications from WebLogic Server to Oracle Application Server, providing an approach to migration with solutions based on the J2EE version 1.3 specification.

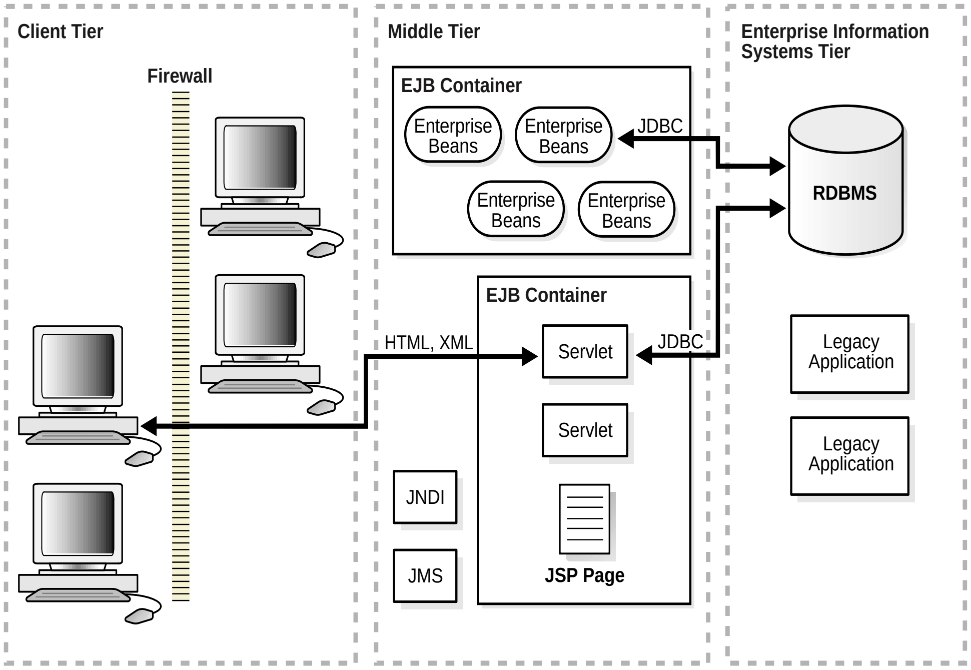
J2EE Application Architecture

The J2EE platform provides a multi-tiered, distributed application model. Central to the J2EE component-based development model is the notion of containers. Containers are standardized runtime environments that provide specific services to components. Thus, Enterprise JavaBeans (EJB) developed for a specific purpose in any organization can expect generic services such as transaction and EJB life cycle management to be available on any J2EE platform from any vendor.

Containers also provide standardized access to enterprise information systems; for example, providing RDBMS access through the JDBC API. Containers also provide a mechanism for selecting application behavior at assembly or deployment time.

As shown in Figure 1–1, the J2EE application architecture is a multi-tiered application model. In the middle tier, components are managed by containers; for example, J2EE web containers invoke servlet behavior, and EJB containers manage life cycle and transactions for EJBs. The container-based model separates business logic from system infrastructure.

Figure 1–1 J2EE Architecture



Migration Issues

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 specification.

  If a component is largely non-compliant with the J2EE specification, 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 1.3 (the version on which this guide is based), then the specification implementation differences will need to be addressed.

Migration Approach

The approach in developing this migration guide was to document our experience migrating web 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.

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.

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 features respectively, 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:

- Application Server Product Offerings
- Architecture Comparison
- High Availability and Load balancing
- J2EE Support Comparison
- Java Development and Deployment Tools

Application Server Product Offerings

WebLogic Server is a component of the WebLogic Platform, which includes several other WebLogic products mentioned below. Oracle Application Server also includes many component products, which are discussed below.

WebLogic

WebLogic Server is available in the following product configurations:

- WebLogic Server
- WebLogic Enterprise
- WebLogic Express

WebLogic Server

WebLogic Server provides the core services and infrastructure for J2EE applications. It supports J2EE 1.3 features. These J2EE 1.3 features include JSP 1.2, Servlets 2.3, EJB 2.0, and JCA 1.0.

WebLogic Server allows Java applications and components to be deployed as web services through SOAP, UDDI, and WSDL. CORBA support is available through BEA Tuxedo.

Each WebLogic Server can be configured as a web server utilizing its own HTTP listener, which supports HTTP 1.1. Alternatively, Apache, Microsoft IIS, and Netscape web servers can also be used. This web server configuration allows WebLogic Server to service requests for static HTML content in addition to dynamic content generated by servlets or JSPs.

A WebLogic Server node can be deployed as an administration server. This node provides administrative services to other WebLogic Servers, called managed servers, in the WebLogic domain. A WebLogic domain is a set of WebLogic Servers and clusters of WebLogic Servers managed by an administration server, inclusive of the latter. The administration server provides a web-based GUI for management of the entire domain. In each domain, WebLogic Servers can be clustered together or are standalone. Refer to "Oracle Application Server Support for High Availability and Load Balancing" for more clustering information.

Note: For a list of J2EE 1.3 APIs supported by WebLogic Server, refer to the section "J2EE Support Comparison" in this chapter.

WebLogic Enterprise

WebLogic Enterprise consists of WebLogic Server and BEA Tuxedo. Tuxedo is a distributed transaction management platform that enables distributed transactions across multiple databases. Tuxedo integrates with WebLogic Server through the latter's connector architecture.

WebLogic Enterprise supports multiple application environments including Java, C++, C, and COBOL. WebLogic Enterprise also supports CORBA applications and allows single sign-on to disparate application environments. Additionally, through Tuxedo, WebLogic Enterprise supports industry standard SNMP MIBS allowing WebLogic Server to be monitored by third-party tools.

WebLogic Express

WebLogic Express is a "lightweight"version of WebLogic Server. It is not J2EE compliant as it does not have support for EJBs and JMS. It does support JSPs, servlets, JDBC, and RMI, and it also includes a web server. Hence, WebLogic Express can be used to build rudimentary web applications with simple database access using JDBC (no support for two-phase transactions).

Oracle Application Server

Oracle Application Server is a platform-independent J2EE application server that can host multi-tier, web-enabled enterprise applications for the Internet and intranets, and which is accessible from browsers and standalone clients. It includes Oracle Application Server Containers for J2EE (OC4J) a lightweight, scalable J2EE container written in Java, and is J2EE 1.3 certified. Hence, OC4J provides support for the following J2EE 1.3 APIs:

- Servlets 2.3
- JSP 1.2
- EJB 2.0
- JNDI 1.2
- JavaMail 1.1.2
- JAF 1.0
- JAXP 1.1
- JCA 1.0
- JAAS 1.0
- JMS 1.0
- JTA 1.0
- JDBC 2.0 Extension

Oracle Application Server is designed specifically for running large-scale, highly available distributed Java enterprise applications, including Internet commerce sites, enterprise portals and high volume transactional applications. It adds considerable value beyond the J2EE standards in areas critical to the implementation of real world applications, providing an entire suite of integrated solutions that encompass:

- Web services

- Business intelligence
- Management and security
- E-business integration
- Support for wireless clients
- Enterprise portals
- Performance caching

To enable these solutions to be implemented in a reliable and scalable infrastructure, Oracle Application Server can be deployed in a redundant architecture using clustering mechanisms and several high availability solutions.

The sections "Architecture Comparison" and "Oracle Application Server Support for High Availability and Load Balancing" in this chapter detail the components and characteristics of Oracle Application Server.

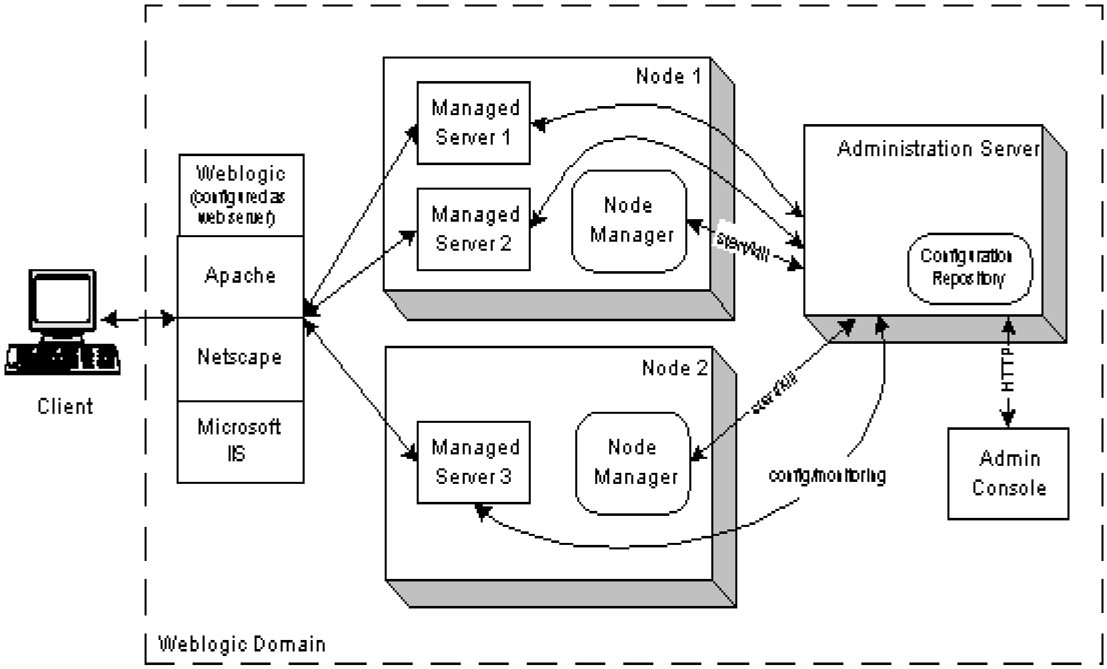
Architecture Comparison

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

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. 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 filesystem 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 failover 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.

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 10g Concepts,

Oracle Application Server 10g Administrator's Guide,

Oracle Application Server 10g High Availability Guide

Oracle Application Server Containers for J2EE User's Guide.

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, which can be a middle tier instance or Infrastructure instance. 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
- Distributed Configuration Management (DCM))

- Oracle Application Server Web Cache
- Oracle Enterprise Manager Application Server Control
- Oracle Application Server Infrastructure

Oracle HTTP Server

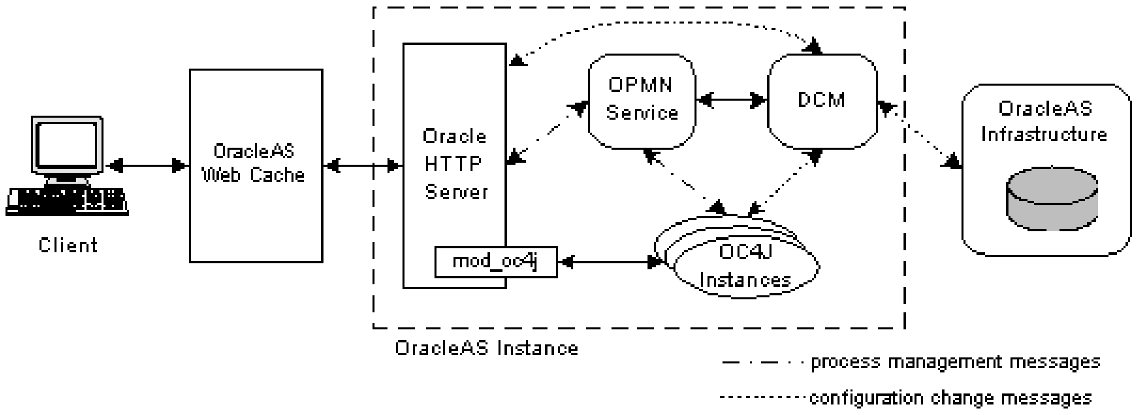
Oracle Application Server contains two listeners: Oracle HTTP Server (based on the Apache open source project) and the listener that is part of OC4J, which runs in a separate thread of execution. Each OracleAS instance has one Oracle HTTP Server.

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 for a J2EE and Web Cache installation type.

Figure 2–2 Components of an OracleAS Instance



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) complete and written entirely in Java. It executes on the standard Java Development Kit (JDK) 1.4 Java Virtual Machine, which is installed with Oracle Application Server (JDK 1.3 is supported). 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 Application Server Control GUI.

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 the section "Oracle Application Server Support for High Availability and Load Balancing" below.

Oracle Process Management Notification (OPMN) Server

Each OracleAS instance has an OPMN component, which performs monitoring and process management functions within that instance. This service communicates messages between the components in an OracleAS instance to enable startup, death-detection and recovery of components. This communication extends to other OPMN services in other OracleAS instances belonging to the same OracleAS Cluster as well, thereby allowing other instances in a cluster to be aware of active OC4J and Oracle HTTP server processes in other OracleAS instances (in the same cluster).

The OPMN service also communicates and interfaces with Application Server Control to provide a consolidated interface for monitoring, configurating, and managing Oracle Application Server. Oracle Application Server components, Oracle HTTP Server, OC4J instances, and Distributed Configuration Management (described below), use a subscribe-publish messaging mechanism to communicate with the OPMN server. For failover and availability, the process that implements the OPMN server has a shadow process that restarts the OPMN process if it fails.

Distributed Configuration Management (DCM)

In order to manage and track configuration changes in the various components in each OracleAS instance, a DCM process exists in each OracleAS instance to perform those tasks. Each configuration change made to any of the components in a OracleAS instance is communicated to the DCM. DCM in turn takes note of the change and records it in the Oracle Application Server Metadata Repository in the Infrastructure database. This repository contains the configuration information for all the OracleAS instances connected to it. All OracleAS instances connecting to the same Oracle Application Server Metadata Repository in this way belong to the same OracleAS Farm. If any of the OracleAS instances fail, the configuration information can be retrieved from the Metadata Repository for the purpose of restarting these instances.

Each DCM also communicates with the OPMN in their respective instances to send notification events on changes in repository data. This allows OPMN to make the corresponding adjustments to the Oracle Application Server components.

Oracle Application Server Web Cache

Oracle Application Server provides a caching solution with the unique capability to cache both static and dynamically generated web content. Oracle Application Server Web Cache (OracleAS Web Cache) significantly improves the performance and scalability of heavily loaded Oracle Application Server hosted web sites by reducing the number of round trips to the Oracle HTTP Server. In addition, OracleAS Web Cache provides a number of features to ensure consistent and predictable responses. These features include page fragment caching, dynamic content assembly, web server load balancing, OracleAS Web Cache clustering, and failover. OracleAS Web Cache can be used as a load balancer for OracleAS instances in a cluster. OracleAS Web Cache can itself be deployed in its own cluster. Refer to the *Oracle Application Server Web Cache Administrator's Guide*.

Oracle Enterprise Manager Application Server Control

Oracle Enterprise Manager Application Server Control (Application Server Control) provides a web-based interface for managing Oracle Application Server components and applications. Using the Application Server Control, you can do the following:

- monitor OracleAS components, OracleAS middle tier and Infrastructure instances, OracleAS middle tier clusters, 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

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 and instructions on how to use it.

Oracle Application Server Infrastructure

Oracle Application Server provides a completely integrated infrastructure and framework for development and deployment of enterprise applications. An Oracle Application Server Infrastructure installation type provides centralized product metadata, security and management services, and configuration information and data repositories for the Oracle Application Server middle tier. By integrating the Infrastructure services required by the middle tier, time and effort required to develop enterprise applications are reduced. In turn, the total cost of developing and deploying these applications is reduced, and the deployed applications are more reliable.

The Oracle Application Server Infrastructure provides the following overall services:

- **Product Metadata Service**

  Oracle Application Server Infrastructure stores all application server metadata required by Oracle Application Server middle tier instances. This data is stored in an Oracle9*i* database, thereby leveraging the robustness of the database to provide a reliable, scalable, and easy-to-manage metadata repository.

- **Security Service**

  The security service provides a consistent security model and identity management for all applications deployed on Oracle Application Server. The service enables centralized authentication using single sign-on, Web-based administration through the Oracle Delegated Administration Services, and centralized storage of user authentication credentials. The Oracle Internet Directory is used as the underlying repository for this service.

- **Management Service**

  This service is used by Distributed Configuration Management to manage and administer Oracle Application Server middle tier instances and the Oracle Application Server Infrastructure. It is also used to administer clustering services for the middle tier. Application Server Control reduces the total administrative cost by centralizing the management of deployed J2EE applications.

The components in OracleAS Infrastructure which implement the above services are:

- Oracle Application Server Metadata Repository
- Oracle Identity Management

Oracle Application Server Metadata Repository Oracle Application Server Metadata Repository is an Oracle9*i* Enterprise Edition database server and stores component-specific information that is accessed by the Oracle Application Server middle tier or Infrastructure components as part of their application deployment. The end user or the client application does not access this data directly. For example, a Portal application on the middle tier accesses the Portal metadata as part of the Portal page assembly aggregation. Metadata also includes demo data for many Oracle Application Server components, such as data used by the Order Management Demo for BC4J.

Oracle Application Server metadata and customer or application data can co-exist in the Oracle Application Server Metadata Repository, the difference is in which applications are allowed to access them.

The Oracle Application Server Metadata Repository stores three main types of metadata corresponding to the three main Infrastructure services described in the section "Oracle Application Server Infrastructure". These types of metadata are:

- product metadata
- identity management metadata
- management metadata

Table 2–1 shows the Oracle Application Server components that store and use these types of metadata during application deployment.

Table 2–1 Metadata and Infrastructure Components

Type of Metadata	Infrastructure Components Involved
Product metadata (includes demo data)	Oracle Application Server Metadata Repository
Identity Management metadata	OracleAS Single Sign-On, Oracle Internet Directory, Oracle Application Server Certificate Authority

Table 2–1 Metadata and Infrastructure Components

Type of Metadata	**Infrastructure Components Involved**
Management metadata	Distributed Configuration Management, Oracle Enterprise Manager

Oracle Application Server Metadata Repository (OracleAS Metadata Repository) is needed for all application deployments except for those using the J2EE and Web Cache installation type. Oracle Application Server provides three middle tier installation options:

- **J2EE and Web Cache:** Installs Oracle HTTP Server, Oracle Application Server Containers for J2EE (OC4J), Oracle Application Server Web Cache (OracleAS Web Cache), Web Services, Oracle Business Components for Java (BC4J), and Application Server Control.
- **Portal and Wireless:** Installs all components of J2EE and OracleAS Web Cache, plus UDDI, Oracle Application Server Portal (OracleAS Portal), Oracle Application Server Syndication Services (OracleAS Syndication Services), Oracle Ultra Search, and Oracle Application Server Wireless (OracleAS Wireless).
- **Business Intelligence and Forms:** Installs all components of J2EE and OracleAS Web Cache, OracleAS Portal and Oracle Application Server Wireless, plus Oracle Application Server Forms Services, Oracle Application Server Reports Services, Oracle Application Server Discoverer, and Oracle Application Server Personalization.

Integration components, such as Oracle Application Server ProcessConnect, Oracle Application Server InterConnect, and Oracle Workflow are installed on top of any of these middle tier install options.

The Distributed Configuration Management (DCM) component enables middle tier management, and stores its metadata in the OracleAS Metadata Repository for both the Portal and Wireless, and the Business Intelligence and Forms install options. For the J2EE and Web Cache installation type, by default, DCM uses a file-based repository. If you choose to associate the J2EE and Web Cache installation type with an Infrastructure, the file-based repository is moved into the OracleAS Metadata Repository.

> **See Also:** *Oracle Application Server Installation Guide* for information on the OracleAS installation details.

Oracle Identity Management The Oracle Identity Management components provide an infrastructure for the security lifecycle of applications and entities in OracleAS. The components that make up Identity Management are:

- *Oracle Internet Directory*

  Oracle Internet Directory is Oracle's implementation of a directory service using the Lightweight Directory Access Protocol (LDAP) version 3. It runs as an application on the Oracle9*i* database and utilizes the database's high performance, scalability, and high availability.

  Oracle Internet Directory provides a centralized repository for creating and managing users for the rest of the Oracle Application Server components such as OC4J, Oracle Application Server Portal, or Oracle Application Server Wireless. Central management of user authorization and authentication enables users to be defined centrally in Oracle Internet Directory and shared across all Oracle Application Server components.

  Oracle Internet Directory is provided with a Java-based management tool (Oracle Directory Manager), a Web-based administration tool (Oracle Delegated Administration Services) for trusted proxy-based administration, and several command-line tools. Oracle Delegated Administration Services provide a means of provisioning end users in the Oracle Application Server environment by delegated administrators who are not the Oracle Internet Directory administrator. It also allows end users to modify their own attributes.

  Oracle Internet Directory also enables Oracle Application Server components to synchronize data about users and group events, so that those components can update any user information stored in their local application instances.

  See Also: *Oracle Internet Directory Administrator's Guide*

- *OracleAS Single Sign-On*

  OracleAS Single Sign-On is a multi-part environment which is made up of both middle tier and database functions allowing for a single user authentication across partner applications. A partner application can be achieved either by using the SSOSDK or via the Apache `mod_osso` module. This module allows Apache (and subsequently URLS) to be made partner applications.

  OracleAS Single Sign-On is fully integrated with Oracle Internet Directory, which stores user information. It supports LDAP-based user and password management through Oracle Internet Directory.

OracleAS Single Sign-On supports Public Key Infrastructure (PKI) client authentication, which enables PKI authentication to a wide range of Web applications. Additionally, it supports the use of X.509 digital client certificates and Kerberos Security Tickets for user authentication.

By means of an API, OracleAS Single Sign-On can integrate with third-party authentication mechanisms such as Netegrity Site Minder.

> **See Also:** *Oracle Application Server Single Sign-On Administrator's Guide*

- *OracleAS Certificate Authority*

  OracleAS Certificate Authority (OCA) is a component of the Oracle public key infrastructure (PKI) offering that allows you to create and manage X.509v3 digital certificates for use in Oracle or third-party software. OCA is fully standards-compliant, and is fully integrated with OracleAS Single Sign-On and Oracle Internet Directory. OracleAS Certificate Authority provides web-based certificate management and administration, as well as XML-based configuration. It leverages the Identity Management infrastructure, high availability, and scalability of the Oracle9*i* platform.

  > **See Also:** *Oracle Application Server Certificate Authority Administrator's Guide*

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.

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 failover 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 are round-robin, weight-based, and parameter-based.

HTTP Session State Load Balancing and Failover (Servlet Clustering)

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 failover 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 failover is not automatic for these replication methods.

EJB and RMI Object Load Balancing and Failover

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

Each WebLogic Server in 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 reflect this. 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. The server also selects another server to replicate the state to since the original server is down. Failover of stateful objects is achieved this way.

Failover 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.

Oracle Application Server Support for High Availability and Load Balancing

Oracle Application Server is designed with several high availability and load balancing mechanisms. These mechanisms ensure that failover and scalability are achieved at the Infrastructure and middle tier levels. For failover, clusters of similar OracleAS components can be created. These clusters offer redundancy for similar components.

This section describes the clustering and load balancing concepts and capabilities of applicable components in Oracle Application Server.

See Also: *Oracle Application Server 10g High Availability Guide*

Oracle Application Server Instance

The Oracle Application Server architecture supports high availability in the middle tier that in many cases can prevent unplanned down time for deployed applications. This section provides an overview of the architecture of an Oracle Application Server instance and shows some of the mid-tier high availability features.

Within each Oracle Application Server instance, the following features provide high availability within the instance, and for any clusters that the instance is a part of:

- Process Monitoring – Using the Oracle Process Manager and Notification Server system provides for process death detection and process restarting in the event that problems are detected for monitored processes.

- Configuration Cloning – Using the Distributed Configuration Management features that uses a Oracle Application Server Metadata Repository for configuration information provides distributed and managed configuration for Oracle Application Server instances and for Oracle Application Server instances that are part of a cluster.
- Data Replication – Using OC4J instances with OC4J islands that provide Web application level stateful session replication, and using EJB sessions, data is replicated across processes within an Oracle Application Server instance and across different Oracle Application Server instances that may reside on different hosts when using Oracle Application Server Clusters. This allows stateful session based applications to remain available even when processes within an Oracle Application Server instance become unavailable or fail.
- 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.

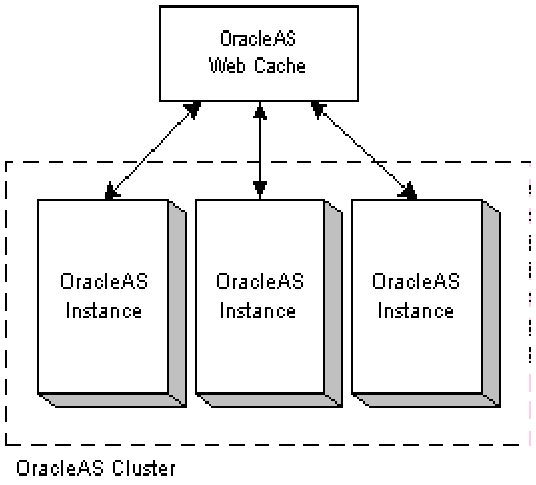
Oracle Application Server Clusters (Middle Tier)

An Oracle Application Server Cluster (OracleAS Cluster) is made up of one or more OracleAS instances (see Figure 2–3). All OracleAS instances in the cluster have the same configuration. The first OracleAS instance to join a cluster has its configuration replicated to the second and later instances when they join. In addition to the configuration, deployed OC4J applications are also replicated to the newer instances. Information for the replicated configuration and applications is retrieved from the OracleAS Metadata Repository used by the cluster.

Within each cluster, there is no mechanism to load balance or failover the OracleAS instances. That is, there is no internal mechanism in the cluster to load balance or failover 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 failover the Oracle HTTP Server instances in the cluster.

Several OracleAS Clusters and standalone OracleAS instances can be further grouped into an OracleAS Farm. The clusters and instances in this farm share the same OracleAS Metadata Repository. For further information on OracleAS Farms, refer to the *Oracle Application Server 10g Administrator's Guide*.

Figure 2–3 An OracleAS Cluster Using OracleAS Web Cache for Load Balancing

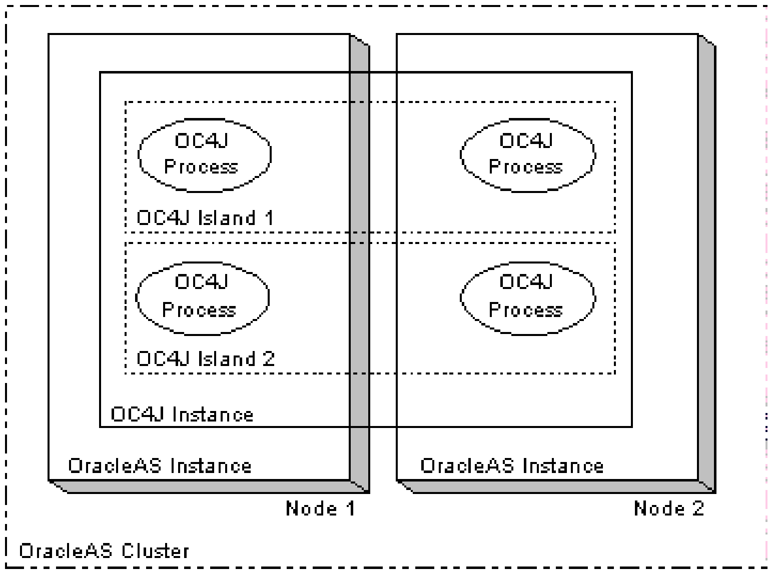


OC4J Islands

An important function of clustering technology in Oracle Application Server is that of reducing multicast traffic. With every server sharing its session state with every other server in the cluster, a lot of CPU cycles is consumed as overhead to replicate the session state across all nodes in the cluster. Oracle Application Server solves this problem by introducing the concept of OC4J islands, where OC4J processes (JVMs) in an OracleAS Cluster can be sub-grouped into islands. Session state of applications is replicated only to OC4J processes belonging to the same island rather than all OC4J processes in the OracleAS Cluster. Hence, state is replicated to a smaller number of processes. OC4J islands are typically configured to span across physical nodes, thereby allowing failover of application state if a node goes down.

Consider an OracleAS Cluster with four OC4J processes running in two nodes, two processes per node (see Figure 2–4). When the state of an application changes, which could occur at every request from the same client, multicast messages are sent between all four processes to update the state of that application in each process. If these four processes were to be divided into two islands of two processes across two nodes, state replication of the application would only have to occur between processes within the same island. Multicast messages would be required only between the two processes in the island instead of four, reducing replication overhead by half. As a result, network traffic and CPU cycles are reduced.

Figure 2–4 OC4J Islands



When configuring OC4J islands, you can specify the number of OC4J processes for each node that belong to each island. By doing so, you can increase or decrease the number of processes based on the capabilities of the hardware and operating system of each node. For instructions on how to configure OracleAS Clusters and OC4J islands, refer to *Oracle Application Server 10g High Availability Guide*.

Stateful Session EJB High Availability Using EJB Clustering

Using OC4J, stateful session EJBs can be configured to provide state replication across OC4J processes running within an application server instance or across an OracleAS Cluster. This EJB replication configuration provides high availability for stateful session EJBs by using multiple OC4J processes to run instances of the same stateful session EJB.

Note: Use of EJB replication (EJB clusters) for high availability is independent of OracleAS Clusters and can involve multiple application server instances installed across nodes that are or are not part of OracleAS Clusters.

EJB clusters provide high availability for stateful session EJBs. They allow for failover of these EJBs across multiple OC4J processes that communicate over the same multicast address. Thus, when stateful session EJBs use replication, this can protect against process and node failures and can provide for high availability of stateful session EJBs running on Oracle Application Server.

See Also:

- *Oracle Application Server 10g High Availability Guide*
- *Oracle Application Server Containers for J2EE User's Guide*
- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

JNDI Namespace Replication When EJB clustering is enabled, JNDI namespace replication is also enabled between the OC4J instances in an OracleAS Cluster. New bindings to the JNDI namespace in one OC4J instance are propagated to other OC4J instances in the OracleAS Cluster. Rebindings and unbindings are not replicated.

The replication is done outside the scope of OC4J islands. In other words, multiple islands in an OC4J instance have visibility into the same replicated JNDI namespace.

See Also: *Oracle Application Server Containers for J2EE Services Guide*

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 island, 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.

Oracle Application Server Web Cache Clusters

Two or more OracleAS Web Cache instances can be clustered together to create a single logical cache. Physically, the cache can be distributed amongst several nodes. If one node fails, a remaining node in the same cluster can fulfill the requests serviced by the failed node. The failure is detected by the remaining nodes in the cluster who take over ownership of the cacheable content of the failed member. The load balancing mechanism in front of the OracleAS Web Cache cluster, for example, a hardware load balancing appliance, redirects the requests to the live OracleAS Web Cache nodes.

OracleAS Web Cache clusters also add to the availability of OracleAS instances. By caching static and dynamic content in front of the OracleAS instances, requests can be serviced by OracleAS Web Cache reducing the need for the requests to be fulfilled by OracleAS instances, particularly for Oracle HTTP Servers. The load and stress on OracleAS instances is reduced, thereby increasing availability of the components in the instances.

Oracle Application Server Web Cache can also perform a stateless or stateful load balancing role for Oracle HTTP Servers. Load balancing is done based on the percentage of the available capacity of each Oracle HTTP Server, or, in other words, the weighted available capacity of each Oracle HTTP Server. If the weighted available capacity is equal for several Oracle HTTP Servers, OracleAS Web Cache uses round robin to distribute the load. Refer to *Oracle Application Server Web Cache Administrator's Guide* for the formula to calculate weighted available capacity.

In the case of failure of a Oracle HTTP Server, OracleAS Web Cache redistributes the load to the remaining Oracle HTTP Servers and polls the failed server intermittently until it comes back online. Thereafter, OracleAS Web Cache recalculates the load distribution with the revived Oracle HTTP Server in scope.

See Also: *Oracle Application Server Web Cache Administrator's Guide*

OracleAS Infrastructure High Availability Solutions

Several solutions exist to enable high availability for the OracleAS Infrastructure. These solutions allow for intrasite failover. They are:

Oracle Application Server Cold Failover Clusters The cold failover cluster solution offers a two-node hardware cluster, which are identically configured. One node is active whilst the other is passive. A hardware interconnect exists between both nodes, which run with an operating system that has clustering features. Both of these nodes access a common shared storage. A single logical IP address is also shared between the two nodes. (A unique physical IP address also exists for each node. But only the single logical IP address is visible and used by the middle tier to access the Infrastructure on the cold failover cluster.

During OracleAS Infrastructure installation, the "Oracle Home" for the installation is installed on the shared storage together with the database files. During operation, only one node is mounted on the shared storage at any one time. In the event that the active node fails, the clustering software of the passive node detects the failure and "takes over" the logical IP address. The passive node becomes the active node, mounts the shared storage, and services requests from the middle tier.

The cold failover cluster nodes can also be installed with the middle tier. In this scenario, the nodes are active-active for the middle tier and active-passive for the Infrastructure.

See Also: *Oracle Application Server 10g High Availability Guide*

Oracle Application Server Active Clusters Whilst the cold failover cluster offers an active-passive availability configuration for the Infrastructure, the Oracle Application Server Active Clusters (OracleAS Active Clusters) solution offers active-active availability. The OracleAS Active Clusters solution is based on Oracle9*i* Real Application Clusters technology. It allows more than two nodes to be active in a cluster. The underlying hardware used for each node also utilizes hardware cluster technology. But the IP address take over mechanism is not used. Instead, a hardware load balancer appliance is configured in front of the OracleAS Active Clusters nodes to load balance requests to them. This load balancer has a logical IP name and address, which is/are used by the middle tier to access the Infrastructure. Oracle Net connections bypass this hardware load balancer by using an address list of nodes in the cluster. Both the hardware load balancer appliance and Oracle Net manage the failover of requests to active nodes if a node fails.

See Also: *Oracle Application Server 10g High Availability Guide*

J2EE Support Comparison

This section outlines the differences in the level of support of J2EE specifications between WebLogic Server 7.0 and Oracle Application Server 10*g* (9.0.4).

Oracle Application Server OC4J is fully certified with J2EE 1.3, having passed Sun Microsystems' Certification Test Suite (CTS). The CTS includes over 5,000 tests designed to assess application portability and the overall quality of a J2EE implementation. WebLogic Server is also J2EE 1.3 certified.

Table 2–2 lists the J2EE technologies and the level of support provided by Oracle Application Server and WebLogic Server:

Table 2–2 J2EE Support

J2EE Technology	Version Supported by WebLogic Server 7.0	Version Supported by Oracle Application Server 10*g* (9.0.4)
JDK	1.3	1.4 and 1.3
Servlets	2.3	2.3
JSPs	1.2	1.2
EJBs	2.0	2.0
JDBC	2.0	2.0 Extension
JNDI	1.2	1.2
JTA	1.0.1	1.0.1
JMS	1.0.2	1.0.2
JavaMail	1.1	1.1.2
JAF	None	1.0.1
JAXP	1.1	1.1
JCA	1.0	1.0
JAAS	1.0	1.0

Note: Oracle Application Server OC4J is installed with JDK 1.4.1. However, OC4J can also work with JDK 1.3.x for this version, 10*g* (9.0.4), of Oracle Application Server.

In addition to supporting these standards, Oracle Application Server provides a well thought out, integrated architecture for building real world J2EE applications, including implementation of standard deployment archives: JAR files for EJBs, Web Archives (WARs) for servlets and JSPs, and Enterprise Archives (EARs) for applications. This ensures smooth interoperability with other standards-compliant application servers.

Java Development and Deployment Tools

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

WebLogic Development and Deployment Tools

The WebLogic development environment and Administration Console are described below.

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.

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.

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.

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 multi-tier 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 multi-tier Java applications, JDeveloper has the following features:

- Oracle Business Components for Java (BC4J)
- 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 1.3 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, 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*.

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.

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 Application Server Control, 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 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:

- Introduction
- Migrating a Simple Servlet
- Migrating Configuration and Deployment Descriptors
- Migrating a WAR File
- Migrating an Exploded Web Application
- Migrating Cluster Aware Applications

Introduction

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

Both application servers are fully compliant with Sun Microsystem's J2EE Servlet specification, version 2.3. All servlets written to the standard specification will work correctly 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.

Differences Between WebLogic Server and Oracle Application Server Servlet Implementations

Oracle Application Server and WebLogic Server both support the Servlet 2.3 specification. Hence, migrating a servlet from WebLogic Server to Oracle Application Server is straightforward.

OC4J Key Servlet Container Features

One of the key distinguishing features of OC4J is the seamless integration with Single Sign-On (SSO) and Oracle Internet Directory (OID). This is achieved through Oracle's implementation of the Java Authentication and Authorization Service (JAAS) standard - JAAS provider is integrated with OC4J.

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 Application Server Control or the `dcmctl` command line utility. The manual processes described in this chapter of editing XML files and starting OC4J at the command line using the `java` command should be used for development purposes only and is for discussion purposes only.

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. 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 Migrating a Simple Servlet (Cont.)

Step	Description	Process

2	Copy the servlet class file to the appropriate directory	After running the samples that came with WebLogic, 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`*`>/weblogic700/samples/server/config/examples/applications/examplesWebApp/WEB-INF/classes/examples/servlets/`
		
		to:
		
		`<`*`ORACLE_HOME`*`>/j2ee/home/default-web-app/WEB-INF/classes/examples/servlets/`
		
		In Windows, from:
		
		`<`*`BEA_HOME`*`>\weblogic700\samples\server\config\examples\applications\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 (for example, `examples/servlets/`).

3	Restart the `home` OC4J instance, or start it if it is not currently running	Use the Oracle Enterprise Manager Application Server Control administration web pages or the following `dcmctl` command: `dcmctl start|restart -i <`*`appsvr_instance_name`*`> -ct oc4j -co home` where `<`*`appsvr_instance_name`*`>` is the name of your Oracle Application Server instance
4	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.

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 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 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 or the `dcmctl` utility. During the development of a web application, it may be faster to deploy and test edited code using an exploded directory format.

Table 3–3 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 an equivalent):
		
		`<`*`BEA_HOME`*`>/weblogic700/samples/server/src/examples/webapp/cookie`
		
		In this directory, build the application by typing `ant`
		
		When built, a WAR file for this application is created in the following directory:
		
		`<`*`BEA_HOME`*`>/samples/server/config/examples/applications/`

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

Step	Description	Process
2	Deploy the sample application.	**1.** On the machine where the `cookie.war` file is located, open a browser and go to the Application Server Control URL. For example: `http://<hostname>:1810` **2.** Enter your administrator username and password if prompted. Click the name of the Oracle Application Server instance you want to deploy your application to. **3.** Click the `home` OC4J component, which brings up its settings page. **4.** Click "Applications". In the applications page of the `home` OC4J instance, click "Deploy WAR file". The "Deploy Web Application" page appears. **5.** Click the "Browse" button and enter the location of the `cookie.war` file. **6.** In the "Application Name" and "Map to URL" text boxes, enter "cookie" and "/cookie" respectively. Click "Deploy". **7.** 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 sample application.

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

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 10g Administrator's Guide* for detailed information on using the Oracle Enterprise Manager administration 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.

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

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

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.)

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 website by adding an entry in the following website XML file (or the corresponding XML file if a non-default website is used):

(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.

Migrating Configuration and Deployment Descriptors

Since WebLogic Server and Oracle Application Server fully support J2EE 1.3, 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:

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, etc.

- **`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 website 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 website 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.

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.

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, filesystem 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 failover 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 Application Server Control 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 10g High Availability Guide*

Each OracleAS Farm consists of multiple OC4J islands and each island can consist of multiple applications. The sharing of session state for failover is within a particular island.

For instructions on how to set up OracleAS Clusters and OC4J islands, refer to the *Oracle Application Server 10g High Availability 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:

- Introduction
- Migrating a Simple JSP Page
- Migrating a Custom JSP Tag Library
- Precompiling JSP Pages

Introduction

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

Both application servers are fully compliant with Sun Microsystem's JavaServer Page specifications, version 1.1 and 1.2. All JSP pages written to the standard specification will work correctly 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".

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	WebLogic Server
JSP Version Support	1.2	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 Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

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 and SQLJ. 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 connection pooling tags, XML tags, EJB tags, file access tags, email tags, caching tags, OracleAS Personalization tags, OracleAS Ultrasearch tags, and a custom tag library for SQL functionality. 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 Application Server 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.

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.

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.

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.

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 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.

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.	Go to `http://`*`<hostname>`*`:1810` and select the OC4J instance you want to start (*`<hostname>`* is the name of your Oracle Application Server host). Or, use the following `dcmctl` command: `dcmctl start -i <appsvr_instance_name> -ct oc4j -co home` where *`<appsvr_instance_name>`* is the name of your Oracle Application Server instance.

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

Step	Description	Process

2	Copy the JSP page to the appropriate directory	Copy `HelloWorld.jsp` from its directory in your WebLogic Server installation to the appropriate directory in `Oracle Application Server` as follows:
		In UNIX, from:
		`<`*`BEA_HOME`*`>/weblogic700/samples/server/src/examples/jsp/`
		to:
		`<`*`ORACLE_HOME`*`>/j2ee/home/default-web-app/`
		
		In Windows, from:
		`<`*`BEA_HOME`*`>\weblogic700\samples\server\src\examples\jsp\`
		to:
		`<`*`ORACLE_HOME`*`>\j2ee\home\default-web-app\`

3	Copy any files required by the JSP page	Copy `BEA_Button_Final_web.gif` from its directory in your WebLogic Server installation to the appropriate directory in Oracle Application Server as follows:
		In UNIX, from:
		`<`*`BEA_HOME`*`>/weblogic700/samples/server/src/examples/images/`
		to:
		`<`*`ORACLE_HOME`*`>/j2ee/home/default-web-app/images/`
		
		In Windows, from:
		`<`*`BEA_HOME`*`>\weblogic700\samples\server\src\examples\images\`
		to:
		`<`*`ORACLE_HOME`*`>\j2ee\home\default-web-app\images\`
		Note: You may have to create the `images` directory

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

Step	Description	Process
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` where *`<hostname>`* is the Oracle Application Server host you copied the JSP file to.

See Also: *Oracle Application Server 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.

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 to OC4J.

Table 4–3 Migrating a Custom JSP Tag Library

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 Migrating a Custom JSP Tag Library (Cont.)

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\`
		
		of your OC4J installation
		
		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).

Table 4–3 Migrating a Custom JSP Tag Library (Cont.)

Step	Description	Process

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

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>`

Table 4–3 Migrating a Custom JSP Tag Library (Cont.)

Step	Description	Process
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 `dcmctl` command: `dcmctl restart|start -i <appsvr_instance_name> -ct oc4j -co home` where `<appsvr_instance_name>` and `<hostname>` are the names of your Oracle Application Server instance and host respectively.
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 Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* for detailed information on configuring and deploying JSP pages.
- *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on custom JSP tag libraries.

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`.

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 Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on Web Object Cache tags and JESI tags.

WebLogic Server `process` Tag

OC4J does not have an exact equivalent for the `process` tag. The closest option is to use the `jml:useForm` and `jml:if` tags. from Oracle's JSP Markup Language (JML).

> **See Also:** Bean Binding Tag Descriptions and Logic and Flow Control Tag Descriptions subsections in the JSP Markup Language (JML) Tag Descriptions section of Chapter 3 of *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on these JML tags.

Alternatively, you could write Java code to implement the tag.

WebLogic Server `repeat` Tag

The OC4J equivalent for this tag is the `jml:foreach` tag. This tag provides the ability to iterate over a homogeneous set of values. The body of the tag is executed once per element in the set. This tag currently supports iterations over the following types of data structures:

- Java array
- `java.util.Enumeration`
- `java.util.Vector`

However, these tags do not cover data structures such as Iterators, Collections, and the keys of a hashtable.

> **See Also:** The Logic and Flow Control Tag Descriptions subsection in the JSP Markup Language (JML) Tag Descriptions section of Chapter 3 of *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on this JML tag.

For `ResultSets` and `ResultSetMetaData`, OC4J provides tags called the SQL Tags for Data Access. These tags provide functionality very similar to that provided by the WebLogic Server `repeat` tag. The `dbNextRow` tag is the tag that you are likely to be most interested in. This tag can be used to process each row of a result set obtained in a `dbQuery` tag and associated with the specified `queryId`. Place the processing code in the tag body, between the `dbNextRow` start and end tags. The code in the body is executed for each row of the result set.

> **See Also:**
>
> - The Custom Data-Access Tag Library subsection in the SQL Tags for Data Access section of Chapter 4 of *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference* for detailed information on these JML tags.
> - *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* for detailed information about the standard JSP tag library framework and tag-extra-info classes.

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.

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
```

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

or

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

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 Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*.

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.

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 Application Server 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 simple EJB JARs, as well as J2EE web applications in the form of EAR files or in an exploded directory format.

This chapter contains these topics:

- Introduction
- Migration Steps
- Migrating EJBs in a EAR or JAR File
- Migrating an Exploded EJB Application
- Configuring EJBs using Deployment Descriptors
- Writing Finders for RDBMS Persistence
- WebLogic Query Language (WLQL) and EJB Query Language (EJB QL)
- Message Driven Beans
- Configuring Security
- Migrating Cluster-Aware EJB Applications to OC4J

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.

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.

Comparison of WebLogic Server and Oracle Application Server EJB Features

Since both WebLogic Server and Oracle Application Server Containers for J2EE (OC4J) have implemented the same versions of the Enterprise JavaBeans specifications, there are no differences between the two in the core areas. 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*	WebLogic Server 7.0
Session Beans	Available	Available
Container Managed Persistence Entity Beans (CMP)	Available	Available
Bean Managed Persistence Entity Beans (BMP)	Available	Available
Message Driven Beans	Available	Available
JTA Transactions	Available	Available
JCA Enterprise Connectivity	Available	Available
IMS Messaging	Available	Available
Dynamic EJB Stub Generation	Available	Available
Full EAR File Based Deployment	Available	Available

Table 5–1 Comparison of EJB features(Cont.)

Feature	Oracle Application Server 10*g*	WebLogic Server 7.0
Automatic Deployment of EJB Applications	Available	Available
Stateless and Stateful EJB Clustering	Available	Available
Local Interfaces for Enterprise JavaBeans	Available	Available
EJB Query Language (EJBQL) - Automatic Code Generation - Oracle and Non Oracle Database Support - CMP with Relationships	Available	Available
RMI-over-IIOP Support	Available	Available
CMP with Relationships	Available	Available
Concurrency Control - Read-Only Locking - Pessimistic Locking - Optimistic Locking	Available	Available

The following sections go into detail on some of the abovementioned features:

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 does an `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 do the `ejbStore` operation.
- **Simple and Complex Database mapping for CMP**—When using 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). For instance, it is difficult to qualify a `where` clause string in WebLogic Server and this results in doing unnecessary full table scans.

Clustering Support

Application server clustering essentially means the use of a group of application servers that coordinate their actions in order to provide scalable, highly available services in a transparent manner.

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

- **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 and 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) to configure various OracleAS instances to belong to a single cluster / island whether they are multiple servers with load balancing on a single machine or multiple servers with load balancing on different 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 will be used in a cluster, then you need to specify it during the time the EJB stubs are created using `ejbc`, which then results in the creation of special cluster-aware classes that will be 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.
- **Stateless Session Bean Clustering**—Oracle Application Server supports clustering of stateless session beans.
- **Stateful Session Bean and Entity Bean Clustering**—Oracle Application Server supports clustering of stateful session beans and entity beans. Two 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. In delivering clustered EJBs, Oracle is working on optimizing the EJB clustering implementation to avoid introducing performance penalties.
  - Programmatic Simplicity—Additionally, unlike servlets which have a natural session boundary at which to failover 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.

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 will be 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 will depend on the isolation mode of the bean ("*Non- Repeatable-Reads*" or "*Serializable*") and the order in which clients are updating the rows.

WebLogic Server provides a similar set of features.

Security and LDAP Integration

One of the key distinguishing features of OC4J is the seamless integration with Single Sign On (SSO) and Oracle Internet Directory (OID). This is achieved through Oracle's implementation of the Java Authentication and Authorization Service (JAAS) standard. See *Oracle Application Server Containers for J2EE User's Guide* and *Oracle Application Server 10g Security Guide*.

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, if needed, but it is not necessary for developers to create this file. Hence, OC4J has a simpler process for deployment of EJBs.

EJB Migration Considerations

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 a binary level, to ensure portability of compiled, 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 platform to another. Typically, an EJB component requires low level interfaces with the container in the form of stub and skeleton classes that will 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 descriptor that describes generic component properties such as JNDI names and transactional attributes.

- Security attributes.

Implementation-specific elements include:

- Low level helper implementation classes (stubs and skeletons) to 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.

Migration Steps

The tasks involved in migrating EJBs are best analyzed by looking at the steps required for deploying EJBs to an EJB container:

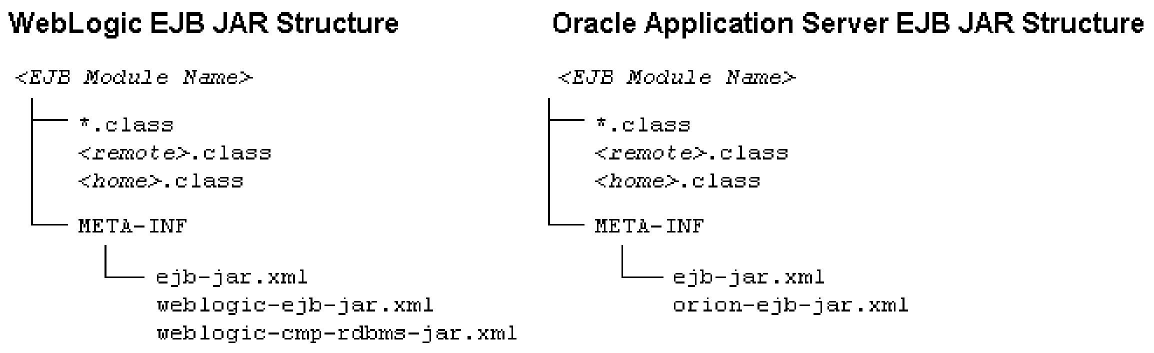
- Setting the EJB deployment descriptors, particularly the vendor-specific deployment descriptors
- Generating EJB container classes
- Loading EJB classes in the server
- Deploying the EJBs in the form of an EAR file or in an exploded directory format
- Configuring the EJBs for deployment at startup

We can address the migration tasks along the same lines.

Setting Deployment Properties

The deployment process starts with a JAR file or a J2EE standard deployment directory that contains the compiled EJB interfaces and implementation classes created by the EJB provider. There should also be an EJB-compliant `ejb-jar.xml` file that describes the bundled EJB(s). The `ejb-jar.xml` file and other required XML deployment files, typically the vendor-specific deployment descriptors, must reside in a top level `META-INF` directory of the JAR file or deployment directory as follows:

Figure 5–1 EJB JAR File Contents and Structure



Vendor-specific Deployment Descriptors

WebLogic Server You would have first created and configured the WebLogic Server-specific and mandatory deployment descriptor, `weblogic-ejb-jar.xml`, and then added the file to the deployment file or directory. The `weblogic-ejb-jar.xml` file is used for specifying caching, clustering, and performance behavior.

If you were deploying an entity EJB that used container managed persistence, you would have also included an additional deployment file for specifying the O-R mapping details, or, in other words, the RDBMS-based persistence services in a file called `weblogic-cmp-rdbms-jar.xml`. A separate file would have been required for each bean that used RDBMS persistence.

OC4J In the case of OC4J, only one file is required. Via Application Server Control, the OC4J-specific and mandatory deployment descriptor, `orion-ejb-jar.xml`, is created and added to the deployment file or directory. The `orion-ejb-jar.xml` file is used for defining caching, clustering, and performance behavior. The details on O-R mapping or the RDBMS-based persistence services are also specified in the `orion-ejb-jar.xml` file. This is different from WebLogic Server where two separate files were required.

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.

WebLogic Server In WebLogic Server, you would have used the `ejbc` 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, `ejbc` creates special cluster-aware classes that will be used for deployment. You can also use `ejbc` 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.

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 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. In lieu of the Application Server Control, you can also use the `dcmctl` command to deploy the EAR file. Refer to *Distributed Configuration Management Reference Guide* for more information.

Loading EJB Classes in the Server

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.

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`.

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. Use the following points as a guideline:

- 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 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 or `dcmctl`.
- You do not need to pre-compile EJB stubs using `ejbc`, `rmic`, or other such facilities into the client application. The OC4J EJB container generates EJB stubs on demand as it needs them.

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 or `dcmctl`.

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.

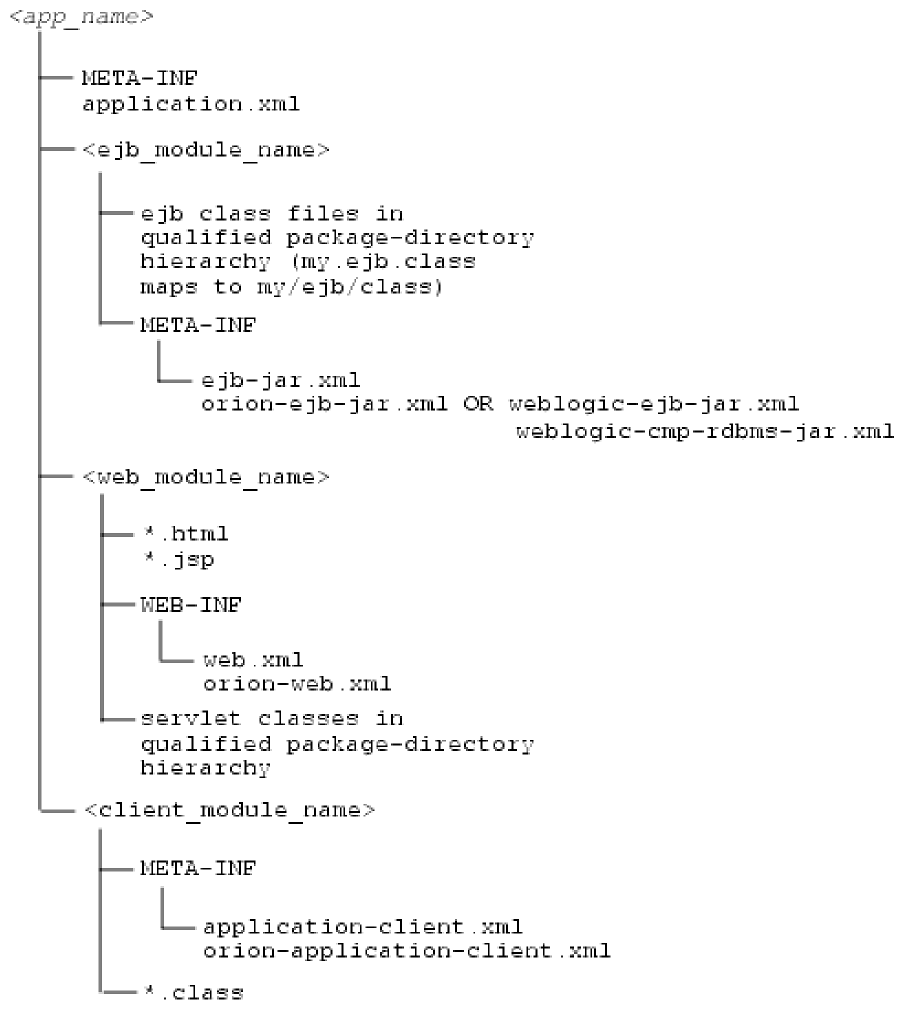
Configuring EJBs using Deployment Descriptors

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 5–2 Directory Structure of a J2EE Application


<app_name>
META-INF
application.xml
<ejb_module_name>
ejb class files in qualified package-directory hierarchy (my.ejb.class maps to my/ejb/class)
META-INF
ejb-jar.xml
orion-ejb-jar.xml OR weblogic-ejb-jar.xml
weblogic-cmp-rdbms-jar.xml
<web_module_name>
*.html
*.jsp
WEB-INF
web.xml
orion-web.xml
servlet classes in qualified package-directory hierarchy
<client_module_name>
META-INF
application-client.xml
orion-application-client.xml
*.class

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`.

Some of the attributes such as *Development Mode* are unique to OC4J.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for more information on the attributes.

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 `ejbc` 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

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, its EJB container is EJB 2.0 compliant and supports EJB QL. This EJB container additionally provides 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*.

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.

Configuring Security

Security can be handled by the application server, or it can be incorporated programmatically into your EJB classes. Both WebLogic Server and OC4J provide similar support for security such as authentication, authorization, and digital certificates.

See Also: The configuring security chapter in *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*.

Migrating Cluster-Aware EJB Applications to OC4J

Oracle Application Server provides clustering features that are superior to WebLogic Server in performance as well as ease of use. Further, migrating cluster-aware applications from WebLogic Server to OC4J is straightforward.

EJB Clustering in WebLogic Server

In-Memory Replication for Stateful Session EJBs

The WebLogic Server EJB container can replicate the state of an EJB across clustered WebLogic Server instances.

Replication support for stateful session EJBs is transparent to clients of the EJB. When a stateful session EJB is deployed, WebLogic Server creates a cluster-aware `EJBHome` stub and a replica-aware `EJBObject` stub for the stateful session EJB. The `EJBObject` stub maintains a list of the primary WebLogic Server instance on which the EJB instance runs and the name of a secondary WebLogic Server to use for replicating the bean's state.

Each time a client of the EJB commits a transaction that modifies the EJB's state, WebLogic Server replicates the bean's state to the secondary server instance. Replication of the bean's state occurs directly in memory, for best performance in a clustered environment.

Should the primary server instance fail, the client's next method invocation is automatically transferred to the EJB instance on the secondary server. The secondary server becomes the primary WebLogic Server for the EJB instance, and a new secondary server is used to account for the possibility of additional failovers. Should the EJB's secondary server fail, WebLogic Server enlists a new secondary server instance from the cluster.

By replicating the state of a stateful session EJB, clients are generally guaranteed to have the last committed state of the EJB, even if the primary WebLogic Server instance fails. However, in certain rare failover scenarios, the last committed state may not be available. This can happen when:

- A client commits a transaction involving a stateful EJB, but the primary WebLogic Server fails before the EJB's state is replicated. In this scenario, the client's next method invocation will work against the previous committed state, if available.
- A client creates an instance of a stateful session EJB and commits an initial transaction, but the primary WebLogic Server fails before the EJB's initial state can be replicated. In this scenario the client's next method invocation will fail to locate the bean instance, because the initial state could not be replicated. The client would need to recreate the EJB instance using the clustered `EJBHome` stub and restart the transaction.
- Both the primary and secondary servers fail. In this scenario the client would need to recreate the EJB instance and restart the transaction.

Requirements and Configuration

To replicate the state of a stateful session EJB in a WebLogic Server cluster, ensure that the cluster is homogeneous for the EJB class. In other words, deploy the same EJB class to every WebLogic Server instance in the cluster, using the same deployment descriptors. In-memory replication is not supported for heterogeneous clusters.

By default, WebLogic Server does not replicate the state of stateful session EJB instances in a cluster. To enable replication, set the replication type deployment parameter to `InMemory` in the `weblogic-ejb-jar.xml` deployment file. For example:

```
<stateful-session-clustering>
...
...
...
<replication-type>InMemory</replication-type>
</stateful-session-clustering>
```

EJB Clustering in Oracle Application Server

EJB clustering in Oracle Application Server provides EJB load balancing and failover. For Oracle Application Server, the mechanisms used to achieve these are different from HTTP session load balancing and failover. For EJBs, load balancing redirection is performed by the EJB client stubs and state replication for failover is done without using cluster islands (a future release of Oracle Application Server will implement cluster islands for EJBs).

To create an EJB cluster, you need to specify which OC4J nodes are part of the cluster and configure each of them with the same multicast address, username, and password. The EJBs to be clustered can then be deployed to each of these nodes. Configuring all nodes in the cluster with the same multicast username and password allows authentication to all nodes with a single username/password combination. If you use a different username/password combination with the same multicast address, another cluster is actually defined. The Application Server Control provides a user interface to specify the multicast username and password.

Load Balancing

Load balancing for EJBs is performed at the EJB client end. The client stubs obtain the addresses of nodes in the cluster in one of two ways: static discovery or dynamic discovery. Once all nodes in the same cluster are known, a client stub selects one at random. Load balancing is performed using a random methodology.

Static and dynamic discovery is performed as follows:

Static Discovery At lookup time, the JNDI addresses of all nodes in the cluster are provided in the lookup URL property. This requires knowledge of the node name and `ormi` port for each node. For example:

```
java.naming.provider.url = ormi://serverA:23791/ejb, ormi://serverB:23792/ejb,
                           ormi://serverC:23791/ejb;
```

Dynamic Discovery For dynamic discovery, at the first lookup made, the first node that is contacted communicates with the other nodes with the same multicast address and username/password. The `ormi` addresses of these nodes are retrieved and returned to the client stubs, which select one of the addresses at random. To enable dynamic discovery, "`lookup:`" is inserted before the `ormi` URL:

```
ic.lookup("lookup:ormi://serverA:23791/ejb");
```

Failover

Depending on the type of EJB that is clustered, failover in an EJB cluster is achieved by request redirection and state replication.

Stateless Session EJBs Load balancing and failover for stateless session EJBs is performed by EJB client stubs by redirecting requests to randomly picked nodes after the nodes have been discovered statically or dynamically. Because of the stateless nature of the EJBs, replication of bean state is not required.

Stateful Session EJBs Load balancing for stateful session EJBs is the same as for stateless session EJBs. For failover, state replication is required, and by default, is replicated to all nodes in the cluster at the end of every method call to each EJB instance. Though reliable, this obviously incurs a significant amount of CPU overhead in all the nodes and degrades performance. Hence, two more replication modes are provided to allow replication without compromising performance significantly: JVM termination and stateful session context replication modes.

The JVM termination mode replicates the state of all stateful session EJBs to one other node when the JVM executing these EJBs terminates gracefully. The replication logic uses JDK termination hooks (JDK 1.3 or later is required). This mode is the most performant among all because replication is done only once. However, reliability is not the best as it is dependent on the JVM's ability to shutdown properly.

Stateful session context mode replicates state programatically. An OC4J-proprietary class, `com.evermind.server.ejb.statefulSessionContext`, is provided to allow you to specify the information to be replicated. By setting this information as parameters for the `setAttribute` method, this information can be replicated to all nodes in the EJB cluster. Hence, EJB providers have more control on when and what to replicate.

Entity EJBs Replication for entity EJBs allows EJB state to be stored in a database. Each time the state of an entity EJB changes, it is updated in the database. The entity EJB that changes the state notifies the other nodes in the cluster that their equivalent entity EJBs are out-of-date. If the node hosting the "up-to-date" EJB fails, the client stub redirects to another node and the out-of-date entity EJB in that node resynchronizes its state with the information in the database.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for information on how to configure EJB clustering.

JNDI Namespace Replication

When EJB clustering is enabled, JNDI namespace replication is also enabled between the OC4J instances that have a role in the EJB cluster. New bindings to the JNDI namespace in one OC4J instance are propagated to other OC4J instances that are participating in the EJB cluster. Rebindings and unbindings are not replicated.

JNDI replication is completed outside the scope of OC4J islands. In other words, multiple islands in an OC4J instance have visibility into the same replicated JNDI namespace. For more information see the *Oracle Application Server Containers for J2EE Services Guide*.

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:

- Introduction
- Migrating Data Sources
- Migrating Connection Pools
- Overview of Clustered JDBC
- Performance Tuning JDBC

Introduction

Migrating applications deployed on WebLogic Server that use JDBC, specifically WebLogic JDBC drivers, to OC4J and Oracle JDBC drivers is can be straightforward, requiring little or no code changes to the applications migrated. Both application servers support the same API levels for the JDBC API - full support for version 2.0 of the specification. 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.

Differences between WebLogic and Oracle Application Server Database Access Implementations

Both WebLogic Server and OC4J have fully J2EE 1.3 compliant containers that permit the usage of all types of JDBC drivers to access several different databases. Further, 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 are minimal, often differing primarily in the area of proprietary extensions. Before analyzing any differences, an overview of JDBC Drivers is apt.

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.

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.

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.

Configuring Data Sources in the Application Server

For Oracle Application Server, you configure data sources using the Application Server Control 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. 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`.

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

Parameter	Description
`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.
`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`).

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.

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 re-used 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 re-use is maximized and connection creation overhead minimized. You can also use connection pooling to control the number of concurrent database connections.

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 stand-alone 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.

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.

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.

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 Feature Comparisons

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

- Java Messaging Service (JMS)
- Java Object Cache
- Dynamic Monitoring System (DMS)
- Active Components for J2EE (AC4J)
- Oracle Application Server TopLink (OracleAS TopLink)

Java Messaging Service (JMS)

Oracle Application Server 10*g* and WebLogic Server 7.0 both support JMS 1.0.2. Table F–2 highlights some of the key JMS features supported by both application servers.

Table F–2 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
Failover of Persisted JMS Messages	Yes	No

Table F–2 JMS Feature Comparison Summary(Cont.)

Feature	Oracle Application Server 10*g*	WebLogic Server 7.0
Message Payloads: Structured Datatypes, Unstructured Datatypes, Relational Data, Text, XML, Objects, Multimedia Data	Yes	Yes
Message Transports: SOAP, Net8	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`.

OracleJMS (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, Net8, 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 failover of persisted messages pertaining to a server failure. Further, it provides failover 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 failover 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.

Java Object Cache

The Java Object Cache is designed to improve access performance for shared Java objects and to reduce SQL-to-Java overhead (database access overhead). The Java Object Cache is distributed and can be accessed in-process, across process boundaries on a single machine, and across processor / machine boundaries.

To improve performance, the creation of Java objects is distributed to avoid bottlenecks. The cache is configurable, allowing objects to be grouped, pooled, pinned, and paged as necessary. Finally, the Java Object Cache is clusterable with updates that can be synchronized across clusters.

The goal of the HTTP-level (for servlets), and EJB clusters is to provide a load balanced and fault tolerant system. The Java Object Cache focuses on providing better performance by distributing pre-computed objects across the Oracle Application Server instances so that expensive computation is done only once.

The Java Object Cache is an in-process cache of Java objects that can be used on any Java platform and by any Java application. It allows applications to share objects across requests, across users, and coordinates the life cycle of the objects across processes. The Java Object Cache enables data replication amongst processes even if they have no island, instance, or cluster relationship amongst each other. This replication provides performance improvement by caching expensive (shared) Java objects no matter which application produced them, and also provides availability improvement in case the sources (for example, database, an external application) required to re-create the Java objects are down. Further, if an object is updated or invalidated in one process, it is also updated or invalidated in all other associated processes. This distributed management allows a system of processes to stay synchronized, without the overhead of centralized control. Moreover, the Java Object Cache also supports versioning of objects, thus allowing different applications to have different versions of an object, which is especially useful during application upgrades.

The Java Object Cache supports two modes of operation: local mode and distributed mode. Using local mode, objects are isolated to a single Java VM process and are not shared. Using distributed mode, the Java Object Cache can propagate object changes - including invalidations, destroys, and replaces- through the cache’s messaging system to other communicating caches running either on a single system or across a network (the Java Object Cache messaging system is built on top of TCP / IP).

Thus, while the primary focus of the Java Object Cache is performance improvement, it does indeed have the side effect of improved scalability and availability, since the contents of the cache are available for results computation even when the back-end server is down.

WebLogic Server does not have any feature comparable to the Java Object Cache.

Dynamic Monitoring System (DMS)

Gathering meaningful performance metrics on a deployed application server is essential for an administrator to troubleshoot bottlenecks, identify resource availability issues, and tune the application server. Oracle Application Server provides such a monitoring framework called Dynamic Monitoring Service.

DMS has been instrumented into a number of Oracle Application Server components including the Oracle HTTP Server, OC4J, OracleAS Portal, Oracle SOAP, and JServ. By publishing key metrics within these components and others, DMS provides a comprehensive view into the performance of applications managed by Oracle Application Server.

For instance, metrics such as the number of currently active requests in the Oracle HTTP Server, milliseconds required to parse the incoming request in the OC4J Servlet Engine, total free memory in the JVM, and numerous operating system metrics are published by DMS and displayed through Application Server Control. Since clients such as OEM retrieve DMS data via HTTP connections, these metrics can also be displayed through a browser.

While DMS is built into Oracle Application Server, it is also available as a monitoring framework for developers to utilize when building their own applications. DMS enables application developers to measure and export customized application specific performance metrics, thus making the application easier to support by administrators, developers, and support analysts.

There is a Java API called Java Management Extensions (JMX) that is in some ways similar to DMS and in other ways complimentary to DMS. WebLogic Server has been providing some support for JMX. Both DMS and JMX are being reviewed by the Java Community for possible inclusion into specifications.

Active Components for J2EE (AC4J)

The existing J2EE framework provides a very productive and scalable environment to author and deploy transaction-based, short-lived applications. However, it does not yet fully address emerging E-business requirements of long-lived interactions between autonomous applications. Further, as Web Services proliferate within organizations, Web Services will need to communicate with other Web Services in a loosely coupled environment, over a long period of time, without limiting resources, and surviving abnormal system crashes. Long-lived interactions between autonomous application components and Web Services require that all participants act as peers and communicate in an asynchronous and reliable fashion. To address these issues, Oracle Application Server introduces Active Components for J2EE (AC4J), a programming framework for developing loosely coupled applications, which are consistent, scalable, and recoverable.

AC4J introduces the concept of loosely coupled beans, known as Active Enterprise Java Beans. An AC4J is a standard EJB with the following important features:

- *Autonomous Peer Model* - With AC4J, each application, when interacting with another application, exists as an autonomous peer. The responding application may choose to ignore the request, or to execute one or more functions on behalf of the requestor (possibly different than the one that the requestor asked for), before responding to the initiating application. As peers, both applications can make requests to each other, but neither can require submission from the other. Neither application can assume control over the resources that its peer application owns.

  To support this programming model, Oracle Application Server provides reliable asynchronous, disconnected, one way, or request/response type interactions between Active EJBs.

- *Declarative Specification* - AC4J makes this entire paradigm easy to program allowing users to declaratively specify a number of pieces of information by simply adding attributes to the EJB deployment descriptor. When deployed, Oracle Application Server provides a runtime environment that automatically provides a number of services for these components.

  AC4J hides queues/topics and related JMS constructs from applications, provides automatic definition of communication message formats, and packs/unpacks messages, automatic routing of service requests to the appropriate service provider, automatic security context propagation, authorization and identity impersonation, automatic exception routing and handling, which is integrated in the EJB framework and automatic tracking and documenting of the computation progress of Active EJBs.

- *Transactional Data Driven Execution of EJB Applications* - AC4J also provides composite matching on available data based on specified rules, which describe under which conditions these data can fire which EJB method. Coupled with the transparent scheduling and activation of EJBs and execution of their methods that this provides, AC4J enables transactional data driven execution of EJBs.

- *Long-Running Transactions* - Since the communication between the various services and the associated tasks being performed can take a very long period of time, AC4J provides automatic durability of J2EE computational state in a portable way, automatically passivating and activating state as necessary.
- *Sophisticated Capabilities* - Further, with AC4J, Oracle Application Server automatically manages load balancing and scalability transparently to the applications, provides conversational state and globally visible identity (ActiveHandle); supports parallel and incremental invocation of Active EJB methods and synchronization on their results (fork and join operations); and supports restart-ability of Active EJB business method(s) in the case of system or application failures while monitoring and enforcement of application consistency.

There is no AC4J-equivalent feature available in WebLogic Server.

Oracle Application Server TopLink (OracleAS TopLink)

In an enterprise Java environment, one of the most formidable challenges is storing business objects and components in a relational database (RDB). OracleAS 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 OracleAS 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. OracleAS 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 OracleAS TopLink solution offers three key benefits:

- *Mature Design, Flexibility and Performance* - OracleAS 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 - OracleAS 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 OracleAS TopLink, an application development team can focus on building the application rather than building infrastructure.

In essence, OracleAS TopLink offers the best solution in the market to perform Java-to-relational database object-relational mapping. With OracleAS 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*, OracleAS TopLink is an integrated component of Oracle Application Server. Specifically, the OracleAS TopLink framework is integrated with the Oracle Containers for J2EE, and the OracleAS TopLink mapping Workbench is integrated with JDeveloper.

Index

A

Apache, 2-2, 2-7
  JServ Protocol, 2-7
`application.xml`, 3-9, 3-10, 5-16
authentication, 5-16, 5-18
authentication credentials, 2-10
authorization, 5-16

B

BEA Tuxedo, 2-2
Business Intelligence and Forms, 2-12

C

centralized repository, 2-13
client stubs, 2-15
clustering
  JDBC, 6-9
  servlets, 3-13
  servlets and JSPs, 3-13
concurrent users, 6-8
configuration cloning, 2-17
connection pool, 6-7
console GUI, 5-9
CORBA, 2-2
CPU cycles, 2-18

D

data replication, 2-17
data sources, 6-4
`data-sources.xml`, 3-10, 5-16, 6-5
DCM, 2-10, 2-12, 2-17
  file-based repository, 2-12
`dcmctl`, 3-3, 3-5, 3-6, 4-5, 4-11, 5-10
digital certificates, 5-16
directory service, 2-13
Distributed Configuration Manager (DCM), 2-8

E

EAR file, 2-26, 3-3, 4-2, 5-2, 5-7, 5-9, 5-16
Edge Side Includes (ESI), 4-12
Edge Side Includes for Java (JESI), 4-3, 4-12
EJB
  cluster, 2-20
  replication, 2-20
  stateful session, 2-19
EJB session, 2-17
`ejbc`, 5-4, 5-9
`ejb-jar.xml`, 3-9, 5-7, 5-11, 5-14, 5-15
Enterprise JavaBeans, 5-2
  clustering, 5-4
    stateful session bean, 5-4
    stateless session bean, 5-4
  load balancing, 5-18
  Query Language, 5-15
  stateful session
    replication, 5-17
entity EJB
  replication, 5-20
  simple and complex DB mapping, 5-3

F

failover, 2-16, 2-18, 5-19

file-based repository, 2-12
finder method, 5-14

G

global-web-application.xml, 3-11

H

HelloWorld, 3-3
high availability, 2-8
HTTP
  1.1, 2-2
  Apache, 2-2
  listener, 3-13
  Microsoft IIS, 2-2
  Netscape, 2-2
  session state, 2-16

I

identity management metadata, 2-11
in-memory replication, 5-17
installation type
  Business Intelligence and Forms, 2-12
  J2EE and Web Cache, 2-12
  Portal and Wireless, 2-12
intelligent routing, 2-17

J

J2EE
  1.3, 2-1, 2-2, 2-23
    supported component specifications, 2-3
  application architecture, 1-6
  application model, 1-2
  Certification Test Suite, 2-23
  components, 1-2
  containers, 2-8
  platform, 1-3
J2EE and Web Cache, 2-12
JAAS, 2-3, 3-2
JAF, 2-3
JAR file, 2-26, 5-7, 5-9
Java Virtual Machine, 2-8, 2-18
JavaBeans, 2-25, 4-3, 4-4
JavaMail, 2-3
JAXP, 2-3
JCA, 2-3
JDBC, 2-3, 6-4
  clustering, 6-9
  DriverManager, 6-4
  drivers, 6-2
JMS, 2-3
jms.xml, 5-16
JNDI, 2-3, 2-15, 5-6, 5-19, 6-4, 6-7
  INITIAL_CONTEXT_FACTORY, 6-7
JNDI namespace
  replication, 2-20
JSP custom tags, 4-2, 4-7, 4-11
JSP pre-translation, 4-15
JTA, 2-3
JVM, 5-19

K

Kerberos Security Tickets, 2-14

L

LDAP, 2-13
load balancer, 2-9, 2-15, 2-17
load balancing
  parameter-based, 2-14, 2-15
  random, 2-15
  round-robin, 2-14, 2-15
  weight-based, 2-14

M

management metadata, 2-11
management service, 2-10
message-driven bean, 5-15
metadata
  identity management, 2-11
  management, 2-11
  product, 2-11
Microsoft IIS, 2-2
migration challenges, 1-5
mod_oc4j, 2-17

mod_oc4j, 2-7
multicast, 2-18, 2-20, 5-18

N

Netegrity Site Minder, 2-14
Netscape, 2-2

O

object-relational mapping, 5-7, 5-8
OC4J
  container, 2-26, 5-9
  failover, 2-18
  instance, 2-17, 2-20
  instances, 2-7
  island, 2-17, 2-18, 3-14
  processes, 2-18
  what is, 1-5
OC4j
  island, 5-18
`oc4j-connectors.xml`, 3-12
`ojspc`, 4-14, 4-16
OPMN, 2-17
Oracle
  Business Components for Java, 2-25
  HTTP Server, 2-7
  Internet Developer Suite, 2-25
  JDeveloper, 2-25
Oracle Application Server
  Certificate Authority, 2-11
  Cluster, 2-16, 2-17, 2-19
  components, 2-6
    Oracle HTTP Server, 2-7
  Farm, 2-17
  Infrastructure, 2-10
  installation, 2-6
  instance, 2-6, 2-16, 2-17, 2-21
  Integration, 2-12
  InterConnect, 2-12
  JSP Markup Language (JML), 4-3, 4-12
  JSP pre-translator, 4-14
  Metadata Repository, 2-12, 2-17
  ProcessConnect, 2-12
  Single Sign-On, 2-11, 3-2
    third party authentication, 2-14
  Web Cache, 2-9, 2-17
    JESI, 4-3
Oracle Delegated Administration Services, 2-10, 2-13
Oracle Developer Suite, 2-25
Oracle Directory Manager, 2-13
Oracle Enterprise Manager, 2-12, 2-26, 5-9, 5-10, 6-4
  Application Server Control, 2-8
Oracle HTTP Server, 2-17
  stateful load balancing, 2-21
  stateless load balancing, 2-21
Oracle Internet Directory, 2-10, 3-2
Oracle JDeveloper, 4-4
Oracle OCI driver, 6-3
Oracle Process Management Notification (OPMN), 2-8
Oracle Workflow, 2-12
Oracle XA drivers, 6-3
Oracle9*i*, 2-10, 2-11
OracleJSP, 4-3
Orion JSP container, 4-3
`orion-application.xml`, 3-10
`orion-ejb-jar.xml`, 3-11, 5-7, 5-11, 5-13, 5-14, 5-16
`orion-web.xml`, 3-11
ormi, 5-19

P

portability, 1-7
Portal and Wireless, 2-12
precompiling, 4-13
`principals.xml`, 3-10
process monitoring, 2-16
product metadata, 2-11
product metadata service, 2-10
proprietary extensions, 1-7
Public Key Infrastructure, 2-14

R

RMI, 2-14, 2-15, 3-13
round-robin, 2-15

S

scalability, 2-8, 2-16
security service, 2-10
`server.xml`, 5-9, 5-11
session state, 2-14, 2-15, 2-16, 2-18, 3-13, 3-14
single sign-on, 2-2, 2-10
skeleton classes, 5-6
smart routing, 2-17
SNMP, 2-2
SOAP, 2-2
SQLJ, 4-3
SSOSDK, 2-13
state replication
  database, 3-13
  filesystem, 3-13
  in-memory, 3-13
stateful session replication, 2-17
stub classes, 5-7, 5-18
stubs, 2-15

T

tag library, 2-26
  custom, 4-7
transactions
  two-phase, 2-3

U

UDDI, 2-2

W

WAR file, 2-26, 3-1, 3-2, 3-3, 3-6
Web Cache, 2-9
WebLogic Express, 2-2, 2-3
WebLogic Server, 1-8
  administration console, 2-24
  Administration Server, 2-4
  cluster, 2-14
  clustering
    servlets and JSPs, 3-13
  components, 2-4
  `config.xml`, 3-12
  console GUI, 2-5, 6-9
  domain, 2-2, 2-5
  Enterprise JavaBeans
    field-to-database-element mapping, 5-13
    in-memory replication, 5-18
  failover, 2-15
  htmlKona, 3-2
  JDBC drivers, 6-2
  jDriver, 6-3
  JSP compiler, 4-13
  JSP custom tags, 4-2, 4-11
  load balancing, 2-15
    parameter-based, 2-15
    random, 2-15
    round-robin, 2-15
    weight-based, 2-15
  Managed Server, 2-4
  proxy plug-in, 2-15
  round-robin, 2-15
  session state, 2-15
  state replication, 2-15, 3-13
  `weblogic-ejb-jar.xml`, 3-13
  `weblogic.xml`, 3-12
`weblogic-cmp-rdbms-jar.xml`, 5-7, 5-11, 5-14
`weblogic-ejb-jar.xml`, 5-7, 5-11, 5-13, 5-15, 5-18
`web.xml`, 3-3, 3-9
WSDL, 2-2

X

X.509 certificate, 2-14