Oracle® Application Server Containers for J2EE

User's Guide

10*g* (9.0.4)

Part No. B10322-01

September 2003

ORACLE

Oracle Application Server Containers for J2EE User's Guide, 10*g* (9.0.4)

Part No. B10322-01

Primary Author: Sheryl Maring

Contributing Author: Brian Wright, Timothy Smith

Contents

3 Advanced Configuration, Development, and Deployment

4 Data Sources Primer

5 Servlet Primer

6 JSP Primer

A Additional Information

B Third Party Licenses

Index

Send Us Your Comments

Oracle Application Server Containers for J2EE User's Guide, 10*g* (9.0.4)

Part No. B10322-01

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 - 650-506-7225. Attn: Java Platform Group, Information Development Manager
- Postal service:
  Oracle Corporation
  Java Platform Group, Information Development Manager
  500 Oracle Parkway, Mailstop 4op9
  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 introduces you to the *Oracle Application Server Containers for J2EE User's Guide*, discussing the intended audience, structure, and conventions of this document. It also provides a list of related Oracle documents.

Intended Audience

This manual is intended for anyone who is interested in using Oracle Application Server Containers for J2EE (OC4J), assuming you have basic knowledge of the following:

- Java and J2EE
- XML
- JDBC

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.

Structure

The *Oracle Application Server Containers for J2EE User's Guide* contains the following chapters and appendices:

Chapter 1, "OC4J Overview"

This chapter describes OC4J primary features, an overview of J2EE APIs and OC4J support, and tunneling and performance services provided by OC4J.

Chapter 2, "Configuration and Deployment"

This chapter discusses how to install OC4J, how to configure and deploy the FAQ application.

Chapter 3, "Advanced Configuration, Development, and Deployment"

This chapter covers advanced OC4J information. It includes an overview of OC4J XML configuration files, how they relate to each other, what happens when you deploy an application, some tips on manual XML configuration file editing for applications, when OC4J automatic deployment for applications occurs, and building and deploying within a directory.

Chapter 4, "Data Sources Primer"

This chapter documents how to use data sources and the JDBC driver.

Chapter 5, "Servlet Primer"

This chapter instructs how to create and use a servlet in OC4J.

Chapter 6, "JSP Primer"

This chapter instructs how to create and use a JSP page in OC4J.

Chapter 7, "EJB Primer"

This chapter instructs how to create and use an EJB in OC4J.

Chapter 8, "OC4J Clustering"

This chapter describes how to cluster application server instances, Oracle HTTP Servers, and OC4J instances.

Appendix A, "Additional Information"

This appendix describes the elements of the server XML configuration files, OC4J command-line tool options, and provides configuration and deployment examples.

Appendix B, "Third Party Licenses"

This appendix lists the Java plug-in partners, third party tool support and third party licences.

Related Documents

For more information on OC4J, see the following documentation available from other OC4J manuals:

- *Oracle Application Server Containers for J2EE Services Guide*
- *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*
- *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*
- *Oracle Application Server Containers for J2EE Servlet Developer's Guide*
- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*
- *Oracle Application Server Containers for J2EE Security Guide*

The following documentation may also be helpful in understanding OC4J:

- *Oracle Application Server 10g Administrator's Guide*
- *Oracle Application Server 10g Performance Guide*
- *Oracle Application Server 10g High Availability Guide*

- *Oracle9i JDBC Developer's Guide and Reference*
- *Oracle9i SQLJ Developer's Guide and Reference*
- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server 10g DMS API Reference*

Conventions

The following conventions are used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1

OC4J Overview

This chapter describes OC4J and demonstrates how to install OC4J with the Oracle Application Server installation.

This chapter includes the following topics:

- Introduction to OC4J
- JDK 1.4 Considerations: Cannot Invoke Classes Not in Packages
- Navigating the OC4J Documentation Set
- OC4J Installation
- Using OC4J in an Enterprise or Standalone Environment
- OC4J Communication

Introduction to OC4J

Oracle Application Server provides a complete Java 2 Enterprise Edition (J2EE) environment written entirely in Java that executes on the Java virtual machine (JVM) of the standard Java Development Kit (JDK). You can run Oracle Application Server Containers for J2EE (OC4J) on the standard JDK that exists on your operating system. Refer to the certification matrix on `http://otn.oracle.com`.

OC4J is J2EE certified and provides all the containers, APIs, an services that J2EE specifies. OC4J is based on technology licensed from Ironflare Corporation, which develops the Orion server—one of the leading J2EE containers. Although OC4J is integrated with the Oracle Application Server infrastructure, the product and some of the documentation still contains some reference to the Orion server.

OC4J supports and is certified for the standard J2EE APIs, as listed in Table 1–1.

Table 1–1 Oracle Application Server J2EE Support

J2EE 1.3 Standard APIs	Version Supported
JavaServer Pages (JSP)	1.2
Servlets	2.3
Enterprise JavaBeans (EJB)	2.0
Java Transaction API (JTA)	1.0
Java Message Service (JMS)	1.0
Java Naming and Directory Interface (JNDI)	1.2
Java Mail	1.1.2
Java Database Connectivity (JDBC)	2.0 Extension
Oracle Application Server Java Authentication and Authorization Service	1.0
J2EE Connector Architecture (JCA)	1.0
JAXP	1.1

The OC4J documentation assumes that you have a basic understanding of Java programming, J2EE technology, and Web and EJB application technology. This includes deployment conventions such as the `WEB-INF` and `META-INF` directories.

Examples in each of the primers assume the following:

- You have a working JDK. The JDK version 1.4.1 is installed with Oracle Application Server. However, OC4J can also work with JDK version 1.3.x.
- You have installed Oracle Application Server, including the OC4J software.
- You have started the OC4J instance.

For Web applications, when specifying a URL to execute an application in Oracle Application Server, note that by default in Oracle Application Server, you should use port 7777 to go through the Oracle HTTP Server, with OracleAS Web Cache enabled.

Examples also use standard J2EE configuration files such as `web.xml` and `application.xml`.

JDK 1.4 Considerations: Cannot Invoke Classes Not in Packages

Among the migration considerations in moving to a Sun Microsystems JDK 1.4 environment, which is the environment that is shipped with Oracle Application Server 10*g*, there is one of particular importance to servlet and JSP developers.

As stated by Sun Microsystems, "The compiler now rejects import statements that import a type from the unnamed namespace." (This was to address security concerns and ambiguities with previous JDK versions.) Essentially, this means that you cannot invoke a class (a method of a class) that is not within a package. Any attempt to do so will result in a fatal error at compilation time.

This especially affects JSP developers who invoke JavaBeans from their JSP pages, as such beans are often outside of any package (although the JSP 2.0 specification now requires beans to be within packages, in order to satisfy the new compiler requirements). Where JavaBeans outside of packages are invoked, JSP applications that were built and executed in an OC4J 9.0.3 / JDK 1.3.1 environment will no longer work in an OC4J 9.0.4 / JDK 1.4 environment.

Until you update your application so that all JavaBeans and other invoked classes are within packages, you have the alternative of reverting back to a JDK 1.3.1 environment to avoid this issue.

Notes:

- The `javac -source` compiler option is intended to allow JDK 1.3.1 code to be processed seamlessly by the JDK 1.4 compiler, but this option does not account for the "classes not in packages" issue.
- Only the JDK 1.3.1 and JDK 1.4 compilers are supported and certified by OC4J. It is possible to specify an alternative compiler by adding a `<java-compiler>` element to the `server.xml` file, and this might provide a workaround for the "classes not in packages" issue, but no other compilers are certified or supported by Oracle for use with OC4J. (Furthermore, do *not* update the `server.xml` file directly in an Oracle Application Server environment. Use the Oracle Enterprise Manager.)

For more information about the "classes not in packages" issue and other JDK 1.4 compatibility issues, refer to the following Web site:

`http://java.sun.com/j2se/1.4/compatibility.html`

In particular, click the link "Incompatibilities Between Java 2 Platform, Standard Edition, v1.4.0 and v1.3".

Navigating the OC4J Documentation Set

Most of the location of J2EE subject matter is obvious. For example, you can find out how to implement and use servlets within the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*. Table 1–2 shows each J2EE subject matter and where you can find the information in the OC4J documentation set.

Table 1–2 Location of Information for J2EE Subjects

J2EE Subject	The Subject is Documented in this OC4J Documentation Book
JSP	*Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*
JSP Tag Libraries	*Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*
Servlet	*Oracle Application Server Containers for J2EE Servlet Developer's Guide*

Table 1–2 Location of Information for J2EE Subjects

J2EE Subject	The Subject is Documented in this OC4J Documentation Book
EJB	*Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*
JTA	*Oracle Application Server Containers for J2EE Services Guide*
Data Source	*Oracle Application Server Containers for J2EE Services Guide*
JNDI	*Oracle Application Server Containers for J2EE Services Guide*
JMS	*Oracle Application Server Containers for J2EE Services Guide*
RMI and RMI/IIOP	*Oracle Application Server Containers for J2EE Services Guide*
Security	*Oracle Application Server Containers for J2EE Security Guide*
CSiV2	*Oracle Application Server Containers for J2EE Security Guide*
JCA	*Oracle Application Server Containers for J2EE Services Guide*
Java Object Cache	*Oracle Application Server Containers for J2EE Services Guide*
OracleAS Web Services	*Oracle Application Server Web Services Developer's Guide*
HTTPS	*Oracle Application Server Containers for J2EE Services Guide*

OC4J Installation

OC4J is a lightweight container that is J2EE-compliant. It is configured with powerful and practical defaults and is ready to execute after installation. OC4J is installed with Oracle Application Server; therefore, see the *Oracle Application Server 10g Installation Guide* for details on OC4J installation.

Using OC4J in an Enterprise or Standalone Environment

OC4J is installed within Oracle Application Server with the goal of managing J2EE enterprise systems. Oracle Application Server can manage multiple clustered OC4J processes. Oracle Application Server, which includes OC4J, is managed and configured through the Oracle Enterprise Manager, which can manage and configure your OC4J processes across multiple application server instances and hosts. Thus, you cannot locally manage your OC4J process using the `admin.jar` tool or by hand editing a single OC4J process' configuration files. This undermines the enterprise management provided by the Oracle Enterprise Manager.

You can still execute OC4J as you have in the past. For those who want a single OC4J instance for development environments or simple business needs, you can download OC4J in standalone mode—complete with documentation.

This following sections discusses both management options in the following sections:

- Managing Multiple OC4J Instances in an Enterprise Environment
- Managing a Single OC4J Instance

Also, the following section describes how to understand the OC4J documentation set:

- OC4J Documentation Set Assumptions

Managing Multiple OC4J Instances in an Enterprise Environment

You manage Oracle Application Server, including OC4J, using Oracle Enterprise Manager within an enterprise system. This includes clustering, high availability, load balancing, and failover.

You configure each OC4J instance and its properties—within the context of an application server instance—using Oracle Enterprise Manager. After configuration, you start, manage, and control all OC4J instances through Oracle Enterprise Manager. You can group several OC4J processes in a cluster. You must use either the Oracle Enterprise Manager management tool or its command-line tools for starting, stopping, restarting, configuring, and deploying applications.

Note: You cannot use the OC4J standalone tool—`admin.jar`—for managing OC4J instances created in an application server instance.

You can modify the XML files locally. If you do so, you must notify Oracle Enterprise Manager that these files have been hand edited through the Distributed Configuration Management (DCM) component tool—`dcmctl`. The following is the command that you execute after hand editing an XML file:

```
dcmctl updateconfig -ct oc4j
```

DCM controls and manages configuration for Oracle Application Server instances and its Oracle HTTP Server and OC4J components. For more information on DCM, see the *Distributed Configuration Management Reference Guide*.

This book discusses how to start, stop, manage, and configure OC4J in an enterprise environment.

Managing a Single OC4J Instance

You can still use a single OC4J—outside of the Oracle Application Server environment. After downloading OC4J in `oc4j_extended.zip` from OTN, you can start, manage, and control all OC4J instances through `oc4j.jar` and the `admin.jar` command-line tool. You configure either through the `admin.jar` command or by modifying the XML files by hand.

Any standalone OC4J process is not managed by Oracle Enterprise Manager and cannot be used within an Oracle Application Server enterprise environment. Typically, you would use standalone for development or for a simple single OC4J instance Web solution.

Download the *OC4J Standalone User's Guide* for information on how to start, stop, configure, and manage your standalone process.

OC4J Documentation Set Assumptions

Aside from this book, the rest of the OC4J documentation set was written with a standalone mindset. These other books may refer to modifying XML files by hand for managing the instance. This book provides a good overview and familiarization of the Oracle Enterprise Manager configuration pages. It also guides you to understand the relationship of each Oracle Enterprise Manager page to its XML counterpart. Use the familiarity of the Oracle Enterprise Manager when reading the other OC4J books. You should be able to look at an XML representation and match it to the relevant Oracle Enterprise Manager field names.

Also, the Distributed Configuration Management (DCM) utility, `dcmctl`, provides a command-line alternative to using Oracle Enterprise Manager for some management tasks. The `dcmctl` tool uses the same distributed architecture and synchronization features as Oracle Enterprise Manager, thereby providing identical functionality in a format that is ideal for scripting and automation.

The following functions can be managed through DCM:

- administration
- managing application server instances
- managing components
- managing clusters

- deploying applications

For other DCM commands that relate to OC4J, see the *Distributed Configuration Management Reference Guide*.

OC4J Communication

For HTTP applications, OC4J is preconfigured to execute behind the Oracle HTTP Server (OHS). You use the Oracle HTTP Server as a front-end listener and OC4J as the back-end J2EE application server.

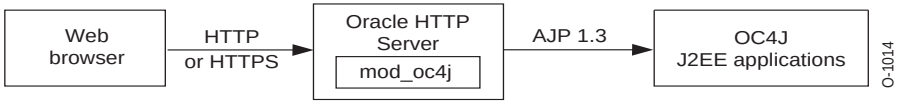
However, for RMI-based applications—such as EJB and JMS—clients should send their requests directly to OC4J. See "Understanding and Configuring OC4J Listeners" on page 3-31 for directions.

HTTP Communication

For all incoming HTTP communication within the application server environment, you use the OHS as a front-end listener and OC4J as the back-end J2EE application server. Figure 1–1 illustrates this as follows:

1. A browser accesses the OHS listener for all HTTP requests. The Oracle HTTP Server is an Apache server. The default port number is 7777.
2. OHS, through the `mod_oc4j` module, passes the request to the OC4J server. The connection between the OHS and OC4J uses the Apache JServ Protocol (AJP) on a port number negotiated during OC4J startup. AJP is faster than HTTP, through the use of binary formats and efficient processing of message headers.

Figure 1–1 HTTP Application Listener



The `mod_oc4j` module is preconfigured to direct all incoming HTTP requests under the `j2ee/` Web context to OC4J. This is to separate incoming requests for JServ from those directed to OC4J. Thus, if you want to use the default routing, you can deploy your Web application into a servlet context that includes as its prefix `j2ee/`. However, any URL mapping you provide in the deployment wizard is automatically added to the `mod_oc4j` module. See "Configuring Oracle HTTP

Server With Another Web Context" on page 3-32 for information on what is added to `mod_oc4j` for you during deployment. For additional information on the `mod_oc4j` module, see the *Oracle HTTP Server Administrator's Guide*.

Notes: In Oracle9*i*AS version 1.0.2.2, the default OC4J Web site did not use the Oracle HTTP Server as a front-end, and it listened using the HTTP protocol on port 8888.

Requirements

For optimum performance, run OC4J with the JDK that is installed with Oracle Application Server Release 2, which is JDK 1.3.x.

It is not necessary to add anything to your `CLASSPATH` to run OC4J, because it loads the Java JAR and class files directly from the installation directory, from the `lib/` subdirectory, and from the deployed application EAR files.

2

Configuration and Deployment

This chapter demonstrates how to configure and execute OC4J as simply and quickly as possible. You installed OC4J with the Oracle Application Server installation.

Within OC4J, you can execute servlets, JSP pages, EJBs, and SQLJ. As an example of deploying an application to OC4J, this chapter describes how to configure the FAQ application demo.

This chapter includes the following topics:

- OC4J Home Page Overview
- Starting and Stopping OC4J
- Creating the Development Directory
- Configuring the FAQ Application Demo
- Deploying Applications
- Recovering From Deployment Errors
- Undeploying Web Applications

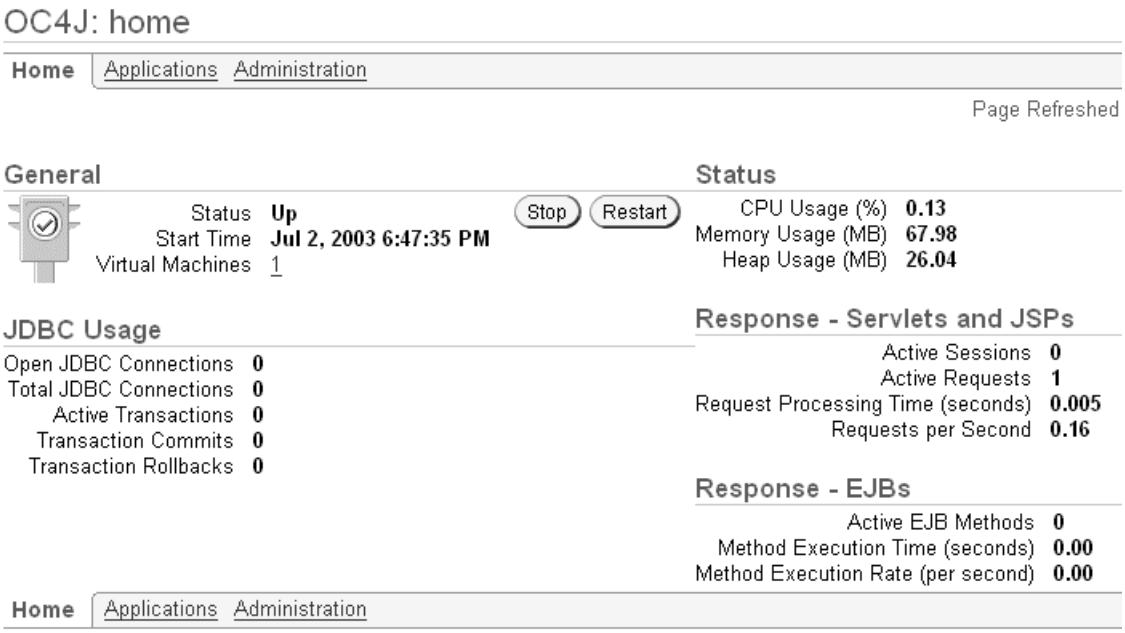
OC4J Home Page Overview

Most of the configuration and management of your OC4J instance occurs off its OC4J Home Page. When you create an OC4J instance off of the Oracle Application Server Instance Home Page, it creates an OC4J Home Page for configuration and management of your OC4J instance. Each OC4J instance has its own OC4J Home Page.

Off the Application Server Control, you can drill down to any of the running OC4J instances by selecting the name of the instance (`home`, for example) in the System Components table. The Application Server Control displays the OC4J Home Page for that instance.

Figure 2–1 shows portions of the OC4J Home Page for the `home` instance.

Figure 2–1 Application Server Control OC4J Home Page



The OC4J Home Page shows metrics on your OC4J instance and its applications. In addition, you can start, stop, and restart all OC4J processes configured to this instance.

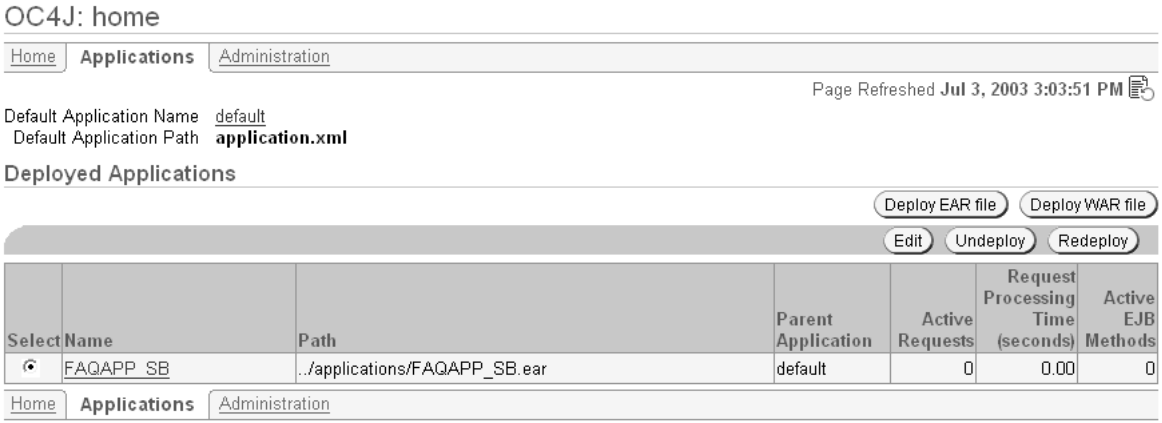
From the OC4J Home Page, you can navigate to the following pages:

- Click **Applications** to access the Application Server Control Applications Page. See "Applications Page" on page 2-3 for more information.
- Click **Administration** to access the Application Server Control Administration Page. See "Administration Page" on page 2-4 for more information.

Applications Page

Figure 2–2 shows the Deployed Applications section. In this section, you can deploy applications using the **Deploy EAR file** or **Deploy WAR file** buttons. After deployment, you can modify configuration for each application. See "Deploying Applications" on page 2-14 for more information.

Figure 2–2 Deployed Applications



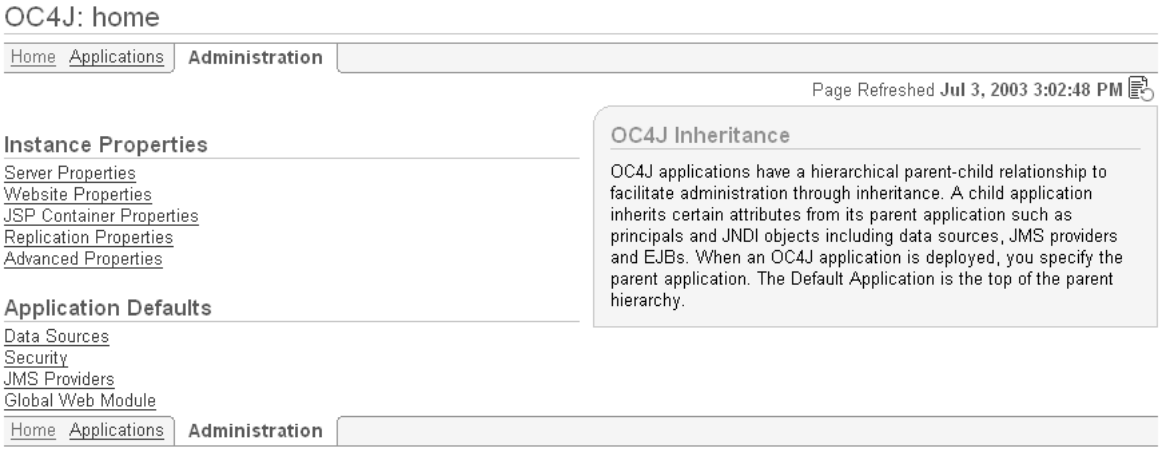
As an example, you can see how the FAQ application demo is configured and deployed to OC4J in "Configuring the FAQ Application Demo" on page 2-7.

Administration Page

Figure 2–3 shows the Administration page. You can modify the following:

- Instance Properties, which are global configuration values for a specific OC4J instance. This includes configuration of OC4J services, such as RMI, JMS, and Web sites.
- Application defaults, including default data sources and security that can be used by all deployed applications in this OC4J instance.

Figure 2–3 Administration Section



Details for each of these options are covered in "Configuring OC4J Using Oracle Enterprise Manager" on page 3-2.

Starting and Stopping OC4J

OC4J is installed with a default configuration that includes a default Web site and a default application. Therefore, you can start OC4J immediately without any additional configuration.

From the Oracle Enterprise Manager Web site, you can start, stop, and restart OC4J with one of the following methods:

- Drill down to the Oracle Application Server Instance Home Page, start the entire Oracle Application Server instance, which includes any configured OC4J

instances, by clicking the **Start All** button in the General section. In addition, **Stop All** and **Restart All** buttons are included for these purposes.

- Drill down to the Oracle Application Server Instance Home Page, start a specific OC4J instance by selecting the radio button next to the OC4J instance. Click the **Start** button. Click **Stop**, **Restart**, or **Delete** to stop, restart, or delete the specified OC4J instance.
- From the Oracle Application Server Instance Home Page, drill down to the OC4J Home Page. Click the **Start** button in the General section on this page. In addition, **Stop** and **Restart** buttons are included for these purposes. Figure 2–1 displays the General section of the OC4J Home Page.

OC4J automatically detects changes made to deployed applications and reloads these applications automatically. Therefore, you do not need to restart the server when redeploying an application. However, you may have to restart OC4J if you modify fields in any of the options off of the Administration page.

You can also start, stop, and restart using the DCM control command. See the *Distributed Configuration Management Reference Guide* for directions.

Testing the Default Configuration

Start OC4J with the defaults, as follows:

1. From the Oracle Application Server Instance Page, start either the whole Oracle Application Server instance or—at least—the Oracle HTTP Server and OC4J components. To start, click the **Start All** button for the Oracle Application Server instance or select the components and click the **Start** button.
2. Test OC4J by specifying the following from a Web browser:

   ```
   http://<ohs_host>:7777/j2ee/j2ee-index.html
   ```

   Substitute the name of the host where the OHS is installed for `<ohs_host>`.
3. Test a servlet deployed in OC4J during installation by specifying the following in a Web browser:

   ```
   http://<ohs_host>:7777/j2ee/servlet/HelloWorldServlet
   ```

   This command returns a "`Hello World`" page. The `HelloWorldServlet` is automatically deployed with the OC4J installation.

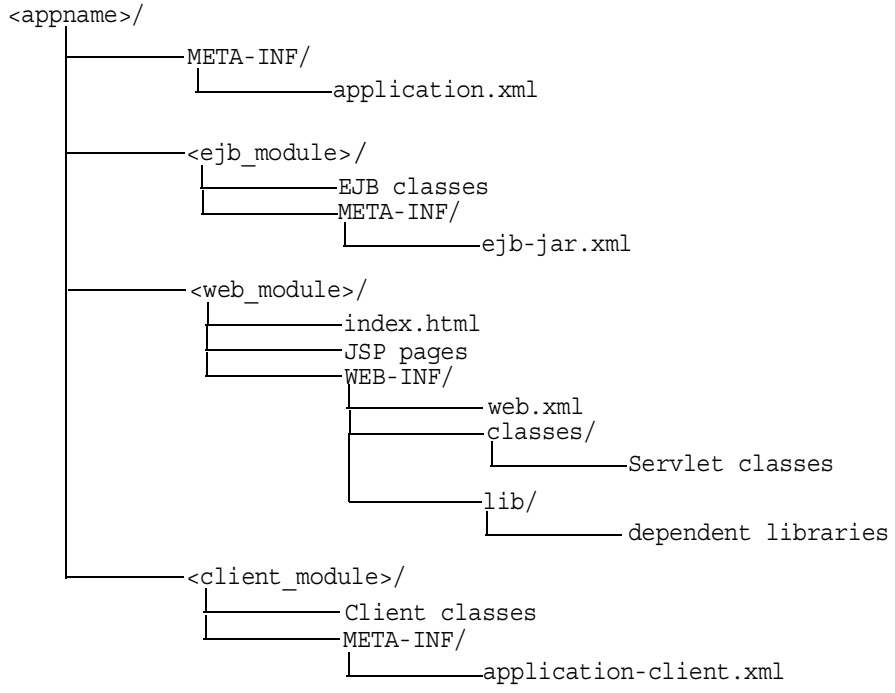
Note: The examples and URLs in this guide use port 7777, which is the default port for the OHS Web listener. If you change the default port number of the OHS, then specify the new port number after the hostname, as follows:

```
http://<ohs_host>:<ohs_port>/j2ee/
```

Creating the Development Directory

When developing your application, Oracle recommends that you use consistent and meaningful naming conventions. As an example, you could develop your application as modules within a directory named after your application. All the subdirectories under this directory could be consistent with the structure for creating JAR, WAR, and EAR archives. Thus, when you have to archive the source, it is already in the required archive format. Figure 2–4 demonstrates this structure.

Figure 2–4 Development Application Directory Structure



Consider the following points regarding Figure 2–4:

- You cannot change the following directory names and XML filenames: `META-INF`, `WEB-INF`, `application.xml`, `ejb-jar.xml`, `web.xml`, and `application-client.xml`.
- Separate directories clearly distinguish modules of the enterprise Java application from each other. The `application.xml` file, which acts as the standard J2EE application descriptor file, defines these modules.
- The directories containing the separate modules (`<ejb_module>`, `<web_module>`, and `<client_module>`) can have arbitrary names. However, these names must match the values in the standard J2EE application descriptor file—the local `application.xml` file.
- The top of the module represents the start of a search path for classes. As a result, classes belonging to packages are expected to be located in a nested directory structure beneath this point. For example, a reference to an EJB package class `'myapp.ejb.Demo'` is expected to be located in `<appname>/<ejb_module>/myapp/ejb/Demo.class`.

Configuring the FAQ Application Demo

This section describes how to configure the FAQ J2EE demo application, which provides support for managing Frequently Asked Questions (FAQs) and storing/retrieving these FAQs from an Oracle database. You must have a working Oracle database and an OC4J installation. You should use this installation for demonstration purposes only and not in a production environment.

FAQs are broadly categorized into *Specialization Areas*. Each Specialization Area is further sub-categorized into *Topics*. Each FAQ can be associated with multiple Specialization Areas, where each area has one or more Topics associated with them.

You can generate a list of FAQs (in HTML format) for a given Specialization Area for internal or external publication.

- Internal: FAQs that are published for internal users only. These include all external and internal FAQs.
- External: FAQs that are published on external forums.

Within the demo, Areas, Topics, and FAQs are entered or updated in the database through Input/Update screens or through a Web service interface. Each Area, Topic and FAQ is uniquely identified by a primary key, which is automatically generated by the system.

This application is a J2EE 1.3 compliant application, developed utilizing the following technologies:

- HTML (including MS-HTML for creating a Rich-Text Editor)
- JavaScript
- Cascade Style Sheets
- Java Server Pages 1.2
- Servlet 2.3
- JSP Standard Tag Library (JSTL) 1.0
- Oracle JSP 1.2 Utility Tag Libraries
- Enterprise JavaBeans 2.0 (using Local Interfaces, Abstract Classes, CMR and EJB-QL)
  - Entity Bean (CMP)
  - Session (Facade) Bean (stateless)
- Oracle Application Server Java Authentication and Authorization Service
- Oracle Application Server Web Services

The following sections detail how to configure and deploy the FAQ demo application. In addition, the last section demonstrates how these steps relate to any application that you may wish to configure and deploy to OC4J:

- Environment Setup for FAQ Demo
- OC4J System Configuration for FAQ Demo
- Deploy the FAQ Demo
- Deployment Details Explained

Environment Setup for FAQ Demo

In order to execute the FAQ demo, you must modify the back-end database to contain tables that the demo uses.

Oracle Database

Add the FAQ user and tables, as follows:

1. Add the `faq` user with password of `faq` in your database.

2. Create the database tables for the FAQ demo by executing the SQL table creation script `CreateTables.sql` script, which is located at `<FAQApp_home>/faq/sql/CreateTables.sql` or can be downloaded with the rest of the FAQ application from OTN at `http://otn.oracle.com/tech/java/oc4j/demos/` in the `FAQApp.zip` file.

   In an Oracle database environment, you can execute the SQL script through SQL*Plus, connecting to the database and schema where you want the tables to be installed and executing `@CreateTables`. Please refer to the Oracle database documentation for further instructions on how to use SQL*Plus, running install scripts, creating database users/schemas, and so on.

OC4J System Configuration for FAQ Demo

In order for the FAQ demo to execute properly, the following system modification must be implemented:

- Modify the default data source, `OracleDS`, to point to the back-end database.
- Add the FAQ user to the `jazn.com` realm and assign it to the `users` role.

The directions for each of these steps are covered in the following sections:

- Data Source Configuration
- Security Configuration

Data Source Configuration

In order to execute the FAQ application, you must have an Oracle database with the corresponding FAQ application database schema installed on it. The FAQ Application uses the default global data source named `OracleDS` that ships with the application server, which must be configured so that it connects to the database in which you created the FAQ tables.

Note: An I/O exception is thrown if you do not update the global `OracleDS` data source appropriately.

---

1. Navigate to the OC4J Home Page on the Oracle Enterprise Manager Web site.
2. Select the Administration tab at the top of the page.

3. Select **Data Sources** under the Application Defaults section. The default application is the automatic parent of each application and it holds global configuration for all deployed applications, such as the data sources used. You are going to modify the default data source that the FAQ application uses.
4. For the `OracleDS` data source, click the **Edit** button. This brings up the configuration information for this data source. Modify the JDBC URL, driver, username, and password to point to your back-end database.

   When finished, click the **Apply** button.

If your back-end database uses the thin JDBC driver, is located at `myhost:1521:ORCL`, and uses the username/password of `faq/faq`, then the `j2ee/home/config/data-sources.xml` file is modified to point to the database at the URL of `jdbc:oracle:thin:@myhost:1521:ORCL`, as follows:

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="faq"
  password="faq"
  url="jdbc:oracle:thin:@myhost:1521:ORCL"
  inactivity-timeout="30"
/>
```

See Chapter 4, "Data Sources Primer" for more information on data sources.

Security Configuration

The FAQ demo uses Oracle Application Server Java Authentication and Authorization Service for authentication and user access control capabilities.

1. Select the Administration tab at the top of the OC4J Home Page.
2. Select **Security** under the Application Defaults section. The default configures global configuration for all deployed applications. You are going to add the user that the FAQ application uses to the `jazn.com` realm.
3. On the Security page, scroll down to the Users section
4. Click **Add User**. A configuration screen appears that allows you to add information about the new user. Supply the following information:

- Name and password of the user
- Check the checkbox next to the `jazn.com/users` realm

When finished, click the **Apply** button.

Alternatively, an application user is added to the default `jazn.com` realm through the `jazn.jar` command line tool, as follows:

```
> java -jar jazn.jar -adduser jazn.com <username> <passwd>
> java -jar jazn.jar -grantrole users jazn.com <username>
```

The previous adds your user (given the username and password) to the `jazn.com` realm and then grants the `users` role to the new user. See the *Oracle Application Server Containers for J2EE Security Guide* for complete information on using OracleAS JAAS Provider as your security provider.

Deploy the FAQ Demo

Download the FAQ demo application from OTN at `http://otn.oracle.com/tech/java/oc4j/demos/` in the `FAQApp.zip` file.

1. Unzip this file to a working directory, which is referred to as `<FAQApp_Home>`.
2. Navigate to the OC4J Home Page on the Oracle Enterprise Manager Web site.
3. Select the Applications tab at the top of the screen.
4. Click on the **Deploy EAR File** button. This starts the application deployment wizard.
5. Provide the EAR file and the name of your application in the Select Application page. Click the **Browse** button to find the `FAQApp.ear` file that you unzipped on your system. Type "`FAQApp`" in the application name field. Click the **Continue** button.
6. Provide the URL mappings for the servlet context on all Web modules in the FAQ application. The FAQApp demo contains a Web module, which should be mapped to the `/FAQApp` servlet context. Type "`/FAQApp`" in the URL mapping field and click the **Next** button.
7. At this point, the FAQApp demo does not need any additional configuration through the wizard. You can jump to the Summary page by clicking **Finish**.
8. Read the summary of the FAQApp application deployment. Click the **Deploy** button to complete the application deployment.

9. On the OC4J Home Page, select "FAQApp" in the Name column of the Applications section. This shows the configuration and all deployed modules of the FAQApp demo application. If the OC4J server is started, the application is automatically started.
10. Execute the FAQApp application in your browser by accessing the OHS, where the default port is 7777.

    ```
    http://<ohs_host>:7777/FAQApp
    ```

    The FAQApp screen appears.

Deployment Details Explained

Although the development of J2EE applications is standardized and portable, the non-application (server) configuration is not. The necessary server configuration depends on the services that your application uses. For example, if your application uses a database, you must configure its `DataSource` object.

For basic applications, such as the FAQ demo, you would configure the following:

- `META-INF/application.xml`—The standard J2EE application descriptor for the application is contained within the `application.xml` file. This file must be properly configured and included within the J2EE EAR file that is to be deployed.
- `DataSource` objects—You must configure the `DataSource` object in the Data Source configuration page for each database used within the application.
- When you deploy any application, you configure the name of the application, the Web contexts, and any application-level data sources or security, as well as other services.

To create and deploy simple J2EE applications, perform the following basic steps:

Basic Step	FAQ Application Step Description
1. Create or obtain the application.	Download the `FAQApp.zip` from OTN
2. Make any necessary server environment changes.	Set the `JAVA_HOME` variable
3. Modify any global configuration modifications.	Modified the default data source, `OracleDS`, and added a user to the default OracleAS JAAS Provider security.

Basic Step	FAQ Application Step Description
4. Provide the application deployment descriptors.	The deployment descriptors, such as `web.xml` and `ejb-jar.xml`, are provided in the `FAQApp.EAR` file. For your application, you may have to create these XML files.
5. Update the application standard J2EE application descriptor file.	The `application.xml` file is included in the `FAQApp.EAR` file. For your application, you may have to create this XML file.
6. Build an EAR file including the application—if one does not already exist.	If you want to modify the FAQ demo, modify within the `src` directory, and use ANT to build an EAR file.
7. Register the application in the appropriate server XML files.	The application is registered when you deploy the application through the deployment wizard.

The following steps describe what modifications to make to deploy the FAQ demo application into OC4J.

> **Note:** Displays of the screens for each step of the deployment wizard are shown in "Deploying Applications" on page 2-14.

1. We asked you to download the FAQ demo application from the Oracle OTN site.
2. Make any necessary server environment changes. You must set the `JAVA_HOME` variable to the base directory of the Java 2 SDK.
3. Modify the global configuration of Oracle Application Server:
   a. Configure the OC4J `DataSource` for an Oracle database. Modify the default data source, `OracleDS`, to point to your back-end database, with the correct URL, username, and password.
   b. Add the user to the `jazn.com` realm.
4. You can modify the FAQ demo application and rebuild it using the ANT command. You must install ANT as it is not provided with OC4J. To rebuild the FAQ demo, execute the following in the `<FAQAPP_Home>` directory:

   ```
   ant all
   ```

The ANT `build.xml` is included in the FAQ ZIP download. To learn more about the ANT file, go to the following Jakarta site:

`http://jakarta.apache.org/ant/`

5. Deploy the FAQApp application with the Oracle Enterprise Manager deployment wizard. When you deploy, the wizard asks for information that is necessary to register the application, such as the name of the application and the Web context.
6. Using the Oracle Enterprise Manager, start OC4J. For a complete description of all the OC4J starting options, see "Starting and Stopping OC4J" on page 2-4.
7. Open your Web browser and then specify the following URL:

   `http://`*`oc4j_host`*`:7777/FAQApp`

Deploying Applications

This section describes how to deploy a J2EE application to the OC4J server. When you deploy an application using the deployment wizard, the application is deployed to the OC4J instance and any Web application is bound to a URL context so that you can access the application from OC4J.

To deploy your application, drill down to the OC4J Home Page and scroll to the Deployed Applications section. Figure 2–2 shows this section.

Note: You can also deploy simple applications with the `dcmctl` command. See the *Distributed Configuration Management Reference Guide* for directions.

Basic Deployment

Your J2EE application can contain the following modules:

- Web applications

  The Web applications module (WAR files) includes servlets and JSP pages.

- EJB applications

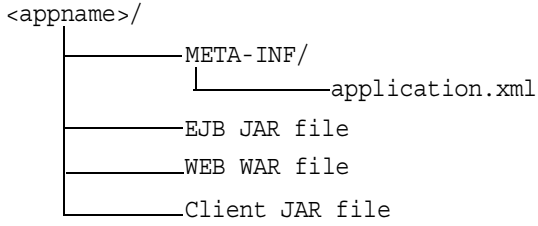
  The EJB applications module (EJB JAR files) includes Enterprise JavaBeans (EJBs).

- Client application contained within a JAR file

Archive the JAR and WAR files that belong to an enterprise Java application into an EAR file for deployment to OC4J. The J2EE specifications define the layout for an EAR file.

The internal layout of an EAR file should be as follows:

Figure 2–5 Archive Directory Format



Archive these files using the JAR command in the `<appname>` directory, as follows:

```
% jar cvfM <appname>.ear .
```

Note that the `application.xml` file acts as a standard J2EE application descriptor file.

- To deploy a J2EE application packaged within an EAR file, click the **Deploy Ear File** button on the Applications page.
- To deploy a J2EE Web application packaged within a WAR file, click the **Deploy WAR File** button on the Applications page.

Both of these buttons start an application deployment wizard, which guides you through deploying an application. In the case of the WAR file, the `application.xml` file is created for the Web application. Whereas, you must create the `application.xml` file within the EAR file. Thus, deploying a WAR file is an easier method for deploying a Web application.

> **Note:** You must still provide configuration for J2EE services, such as data source and security configuration.

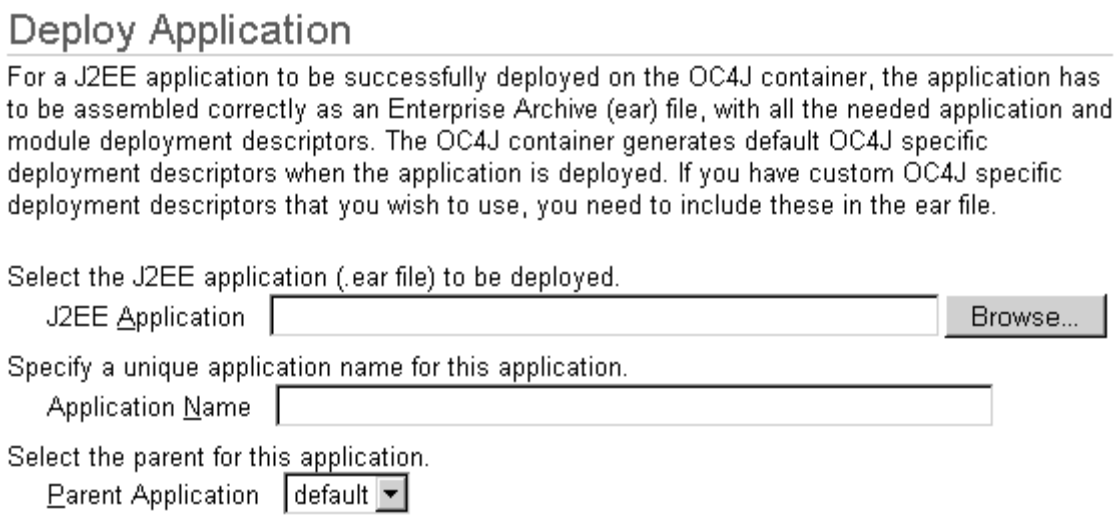
Select Application

Figure 2–6 shows the first page, which enables you to provide the following:

- Browse your system for the EAR file to be deployed.

- Provide a name to be identified with this application. The application name is user-created and will be the identifier for the application in the Applications page.
- Designate a parent application. A child application can see the namespace of its parent application. Thus, setting up an application as a parent is used to share services among children. The default parent is the global application. The parent application must already be deployed to be seen in the pull-down menu.

Figure 2–6 Designate EAR File



When the application is deployed, the information in this step enables the following:

1. Copies the EAR file to the `/applications` directory.
2. Creates a new entry in `server.xml` for the application, as follows:

   ```
   <application name=<app_name> parent="applicationWithCommonClasses"
   path=<path_EARfile> auto-start="true" />
   ```

   where

   - The `name` variable is the name of the application you provided.

- The `parent` is the name of the optional parent application. The default is the global application. Children see the namespace of its parent application. This setting is used to share services, such as EJBs among multiple applications.
- The `path` indicates the directory and filename where the EAR file is deployed.
- The `auto-start` variable indicates if this application should be automatically restarted each time OC4J is restarted.

For a description of the elements in `server.xml`, see "Elements in the server.xml File" on page A-8.

Click the **Continue** button to go to the next step in the wizard deployment process. The wizard uploads your EAR file and examines the application. Based on your configuration files and deployment descriptors, the wizard dynamically presents only the configuration screens your application needs. These screens are a subset of the steps presented in the following sections:

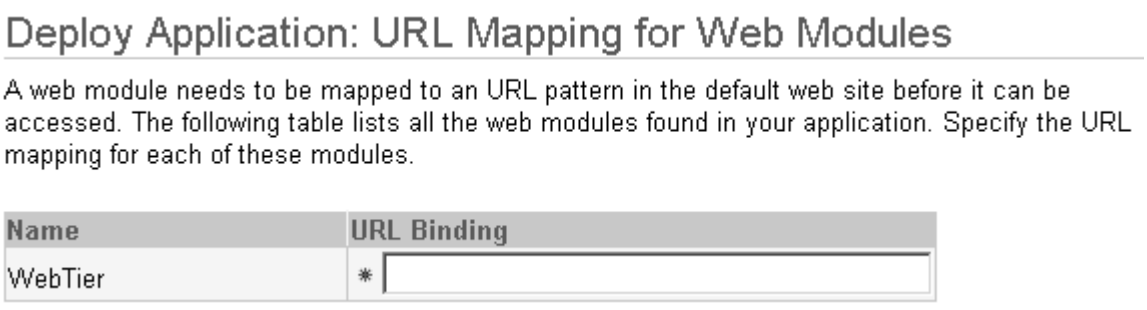
- Provide The URL Mappings For All Web Modules
- IIOP Stub Generation
- Provide Any Resource Reference Mappings
- Specify Any User Manager
- Publish OracleAS Web Services
- Deployment Review

Provide The URL Mappings For All Web Modules

Map any Web modules in your application to a specific URL for its servlet context. When you try to access any Web applications, you provide the host, port, and Web context.

For all Web modules, your URL mapping for this module includes the URL you bind in this screen. Thus, for the URL `http://<host>:<port>/url_name`, provide `/url_name` in the URL mapping screen of the wizard.

Figure 2–7 Configure URL Mapping for Web Modules

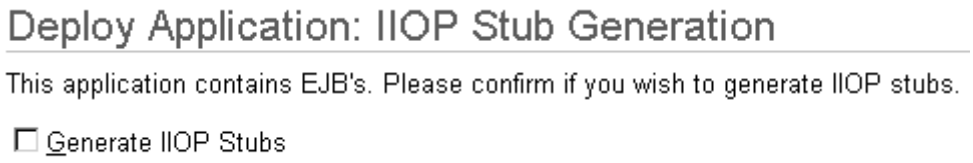


Click the **Next** button to go to the next step in the wizard deployment process.

IIOP Stub Generation

The EJBs in your application can have IIOP stubs generated by selecting **Generate IIOP Stubs**. See the "Interoperability and RMI Tunneling" chapter in the *Oracle Application Server Containers for J2EE Services Guide* for information on IIOP stubs.

Figure 2–8 Generate IIOP Stubs



Click the **Next** button to go to the next step in the wizard deployment process.

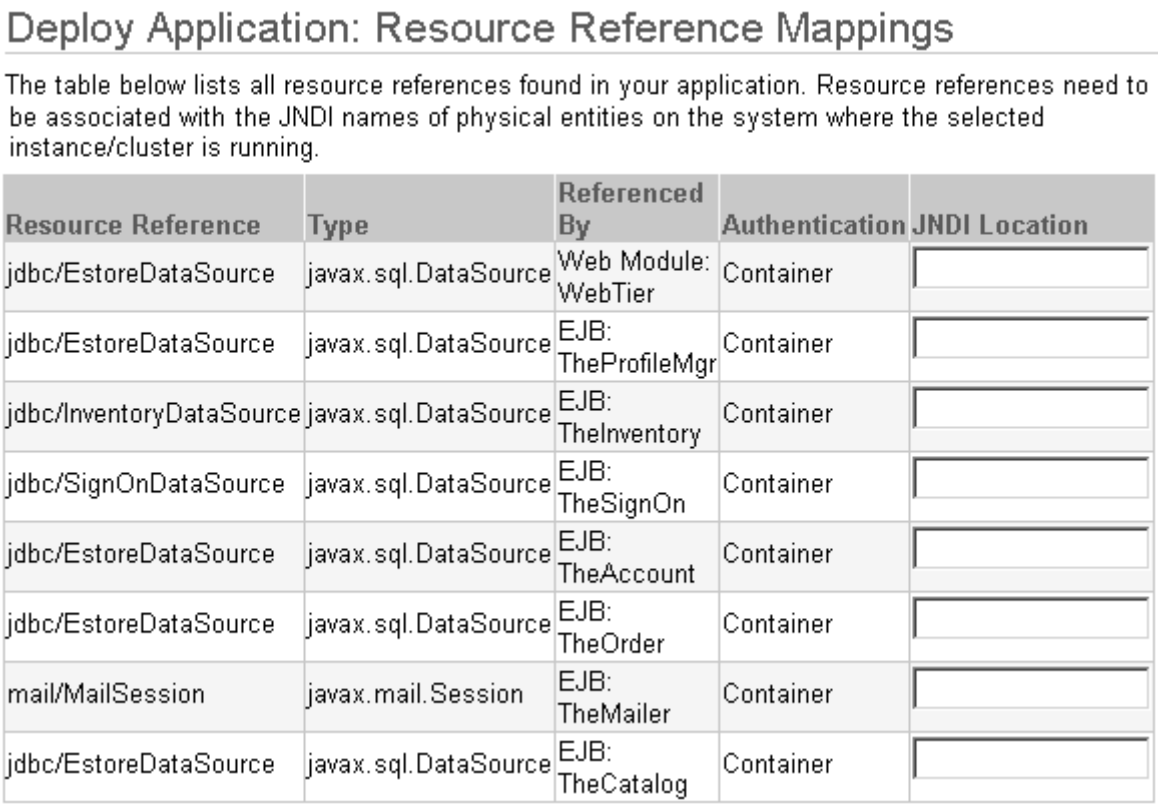
Provide Any Resource Reference Mappings

Map any references resources in your application, such as data sources or mail queues, to physical entities currently present on the OC4J container. Note that if you need a specific resource, you must have already added this to the OC4J container before you deploy your application in order for you to match them in this step.

For most applications, the resource reference you must designate is the data source JNDI name. This screen does not configure the data source information, it only designates an already configured data source or a data source that you will be

configuring later. Designate the JNDI location name of the data source that the application will be using.

Figure 2–9 Map Resource References



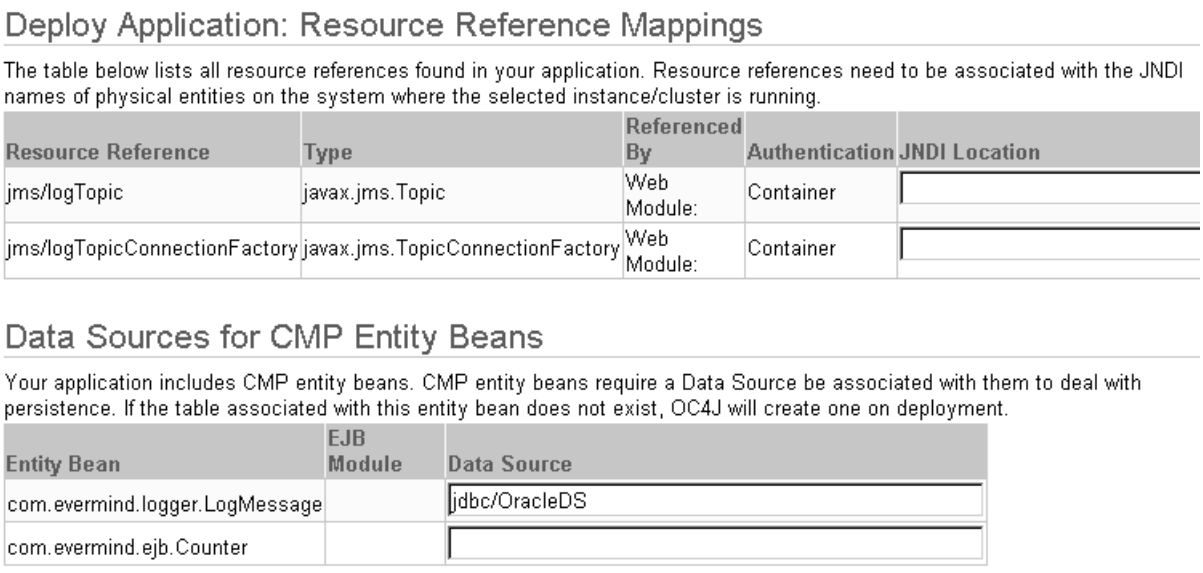
Deploy Application: Resource Reference Mappings

The table below lists all resource references found in your application. Resource references need to be associated with the JNDI names of physical entities on the system where the selected instance/cluster is running.

Resource Reference	Type	Referenced By	Authentication	JNDI Location
jdbc/EstoreDataSource	javax.sql.DataSource	Web Module: WebTier	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheProfileMgr	Container	
jdbc/InventoryDataSource	javax.sql.DataSource	EJB: TheInventory	Container	
jdbc/SignOnDataSource	javax.sql.DataSource	EJB: TheSignOn	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheAccount	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheOrder	Container	
mail/MailSession	javax.mail.Session	EJB: TheMailer	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheCatalog	Container	

If you have any MDBs in your EAR file, you may be required to add information about the subscriptions or topics. If you are defining `DataSource` objects for CMP entity beans, you are given the option to add a JNDI location for those `DataSource` objects.

Figure 2–10 Map Resource References for Entity Beans and MDBs



Deploy Application: Resource Reference Mappings

The table below lists all resource references found in your application. Resource references need to be associated with the JNDI names of physical entities on the system where the selected instance/cluster is running.

Resource Reference	Type	Referenced By	Authentication	JNDI Location
jms/logTopic	javax.jms.Topic	Web Module:	Container	
jms/logTopicConnectionFactory	javax.jms.TopicConnectionFactory	Web Module:	Container	

Data Sources for CMP Entity Beans

Your application includes CMP entity beans. CMP entity beans require a Data Source be associated with them to deal with persistence. If the table associated with this entity bean does not exist, OC4J will create one on deployment.

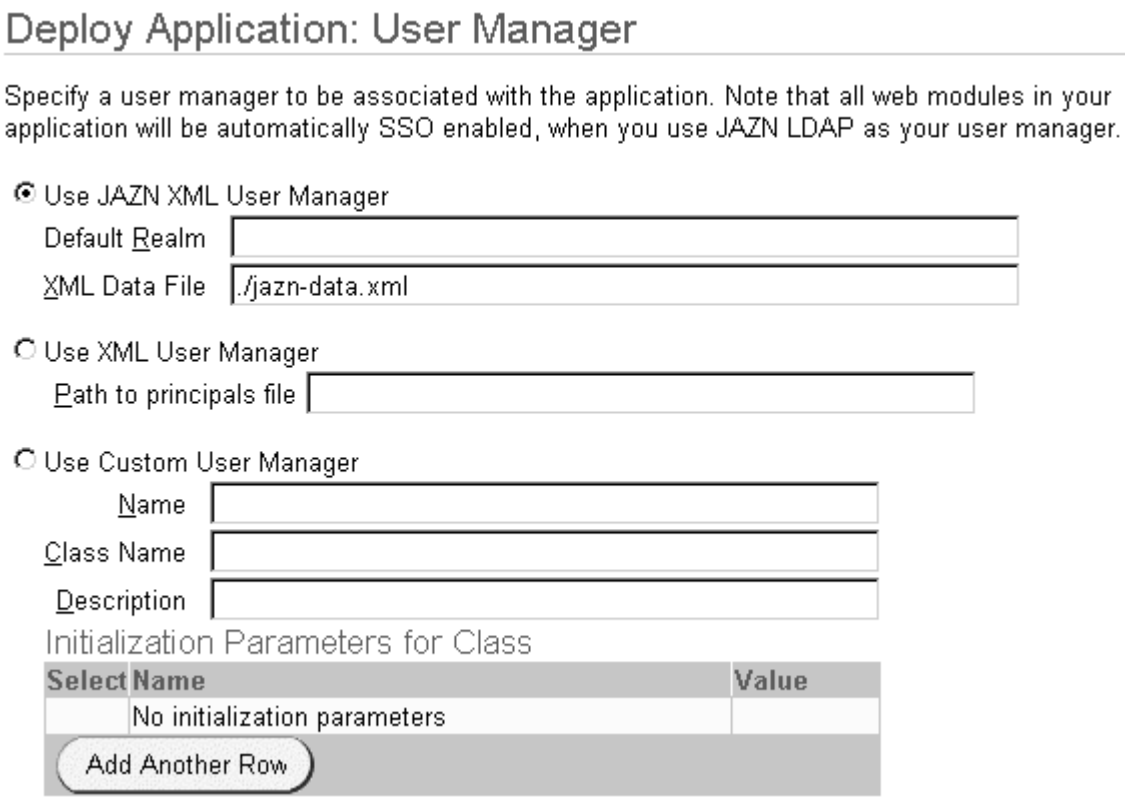
Entity Bean	EJB Module	Data Source
com.evermind.logger.LogMessage		jdbc/OracleDS
com.evermind.ejb.Counter		

Click the **Next** button to go to the next step in the wizard deployment process.

Specify Any User Manager

You can specify what User Manager to use for security. For complete security, we recommend that you choose the OracleAS JAAS Provider XML User Manager.

Figure 2–11 User Manager Choices



As Figure 2–11 demonstrates, you must already have your User Manager set up and configured. Most of the entries requires an XML file that designates the security roles, users, and groups for your security mappings.

- OracleAS JAAS Provider XML User Manager—This is the recommended User Manager. It requires a default realm and a `jazn-data.xml` file.
- XML User Manager—This is not the most secure option. It requires a `principals.xml` file.
- Custom User Manager—This User Manager must be programmed; provide the class name in this field.

For more information on security and User Managers, see the *Oracle Application Server Containers for J2EE Security Guide*.

Publish OracleAS Web Services

Publish any OracleAS Web Services defined in your application. This feature requires the UDDI registry. OracleAS Web Services are not installed with a core install.

If you have defined any OracleAS Web Services, they are shown in the following screen:

Figure 2–12 Publish Web Services

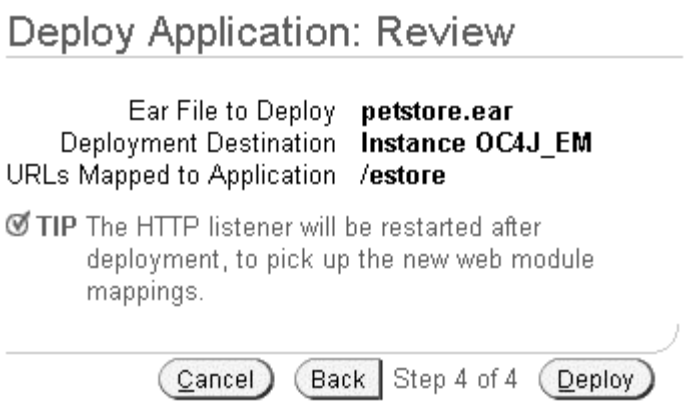
If you want to publish these OracleAS Web Services, then click on the Publish button. This leads you through the process of publishing your OracleAS Web Services. When finished, it brings you back to this screen.

Click the **Next** button to go to the next step in the wizard deployment process.

Deployment Review

At this point, you will receive a review of your application deployment modules and configuration, as follows:

Figure 2–13 Review Panel



In order to deploy this application, click on the **Deploy** button. A message will be displayed that tells you that your application deployed.

Post-Deployment Application Modifications

You can modify any fields and add additional configuration by returning to the OC4J Home page, select the application name in the Applications section. This brings you to a screen with the details of the deployed application.

From within this screen, you can view the Web and EJB modules. In addition, you can add and modify application-specific properties, resources, and security options in the Administration section. It is in this Administration section, that you can add application-specific data sources or security groups or users mentioned in the deployment wizard.

Recovering From Deployment Errors

If the deployment process is interrupted for any reason, you may need to clean up the temp directory, which by default is `/var/tmp`, on your system. The deployment wizard uses 20 MB in swap space of the temp directory for storing information during the deployment process. At completion, the deployment wizard cleans up the temp directory of its additional files. However, if the wizard is interrupted, it may not have the time or opportunity to clean up the temp directory. Thus, you must clean up any additional deployment files from this directory yourself. If you do not, this directory may fill up, which will disable any further deployment. If you receive an `Out of Memory` error, check for space available in the temp directory.

To change the temp directory, set the command-line option for the OC4J process to `java.io.tmpdir=<new_tmp_dir>`. You can set this command-line option in the Server Properties page. Drill down to the OC4J Home Page. Scroll down to the Administration Section. Select **Server Properties**. On this page, Scroll down to the Command Line Options section and add the `java.io.tmpdir` variable definition to the OC4J Options line. All new OC4J processes will start with this property.

Undeploying Web Applications

You can remove a J2EE Web application from the OC4J Web server by selecting the application in the Applications section of the OC4J Home Page (see Figure 2–2) and clicking the **Undeploy** button. This command removes the deployed J2EE application and results in the following:

- The application is removed from the OC4J runtime.
- All bindings for the Web modules are removed from all the Web sites to which the Web modules were bound.
- The application files are removed from both the `applications/` and `application-deployments/` directories.

Note: You can also undeploy applications with the DCM command. See the *Distributed Configuration Management Reference Guide* for directions.

3

Advanced Configuration, Development, and Deployment

Chapter 2, "Configuration and Deployment", discusses basic configuration, development, and deployment of a J2EE application. This chapter discusses both global J2EE service configuration and advanced J2EE application configuration.

This chapter discusses the following topics:

- Configuring OC4J Using Oracle Enterprise Manager
- Overview of OC4J and J2EE XML Files
- What Happens When You Deploy?
- Sharing Libraries
- Understanding and Configuring OC4J Listeners
- Configuring Oracle HTTP Server With Another Web Context
- Building and Deploying Within a Directory
- Developing Startup and Shutdown Classes
- Setting Performance Options
- Enabling OC4J Logging
- OC4J Debugging

Configuring OC4J Using Oracle Enterprise Manager

You can configure J2EE services, J2EE applications, and Oracle Application Server clusters with Oracle Enterprise Manager. Some aspects are configured at the OC4J instance level; thus, they provide a global configuration for all deployed applications in the instance. Other aspects are configured at the application level; thus, this type of configuration is local and applicable only to the application.

The following sections provide you with an overview of advanced configuration within Oracle Enterprise Manager for OC4J:

- OC4J Instance Level Configuration
- Application Level Configuration

OC4J Instance Level Configuration

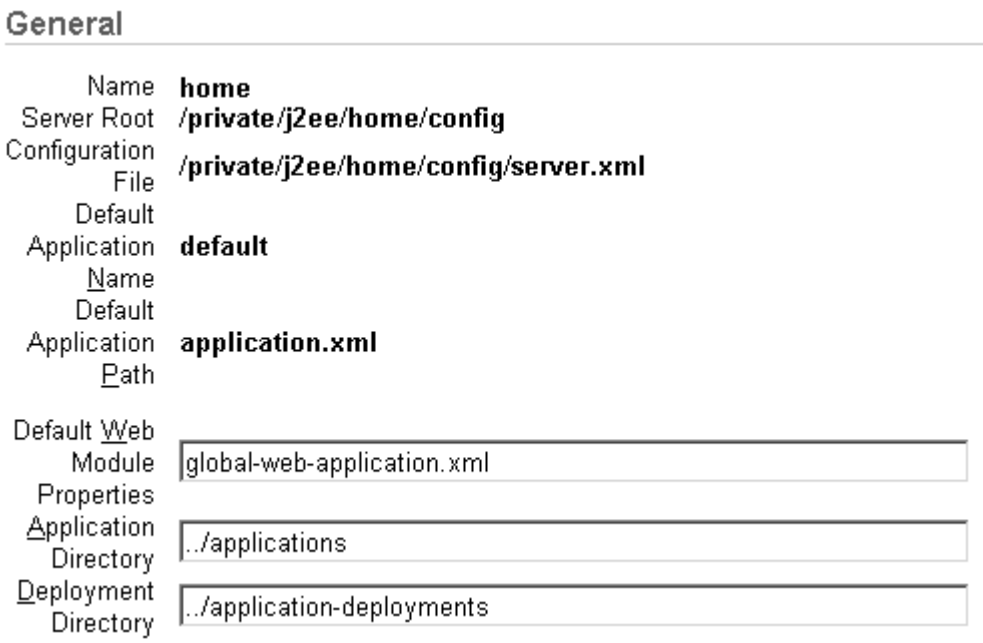
Off of the OC4J Home page, you can configure global services that apply to all applications by drilling down to the Administration page, which allows you to configure the following:

- Configuring Server Properties
- Configure a Web Site
- Configure JSP Container Parameters
- Configure Replication Parameters
- Advanced Configuration Through XML Files
- Configure Data Sources
- Configure Security
- Configure JMS
- Configure Global Web Application Parameters
- Configure RMI

Configuring Server Properties

To configure OC4J properties, scroll down to the Instance Properties section and select Server Properties. The General section of this page contains the following fields:

Figure 3–1 General Properties



The following information about the default server properties is displayed in the upper half of the page.

- Default application—The default application is what most deployed applications used as its parent. Thus, these deployed applications can see the classes within the default application. See "Select Application" on page 2-15 for more information on parent applications.
- Default application path—There exists a file named `application.xml`, which is separate from the `application.xml` included with each EAR file. This `application.xml` file is known as the global `application.xml` file. It defines properties that are used across all deployed applications within this OC4J instance.

In this section, you can modify OC4J server defaults. These are listed below:

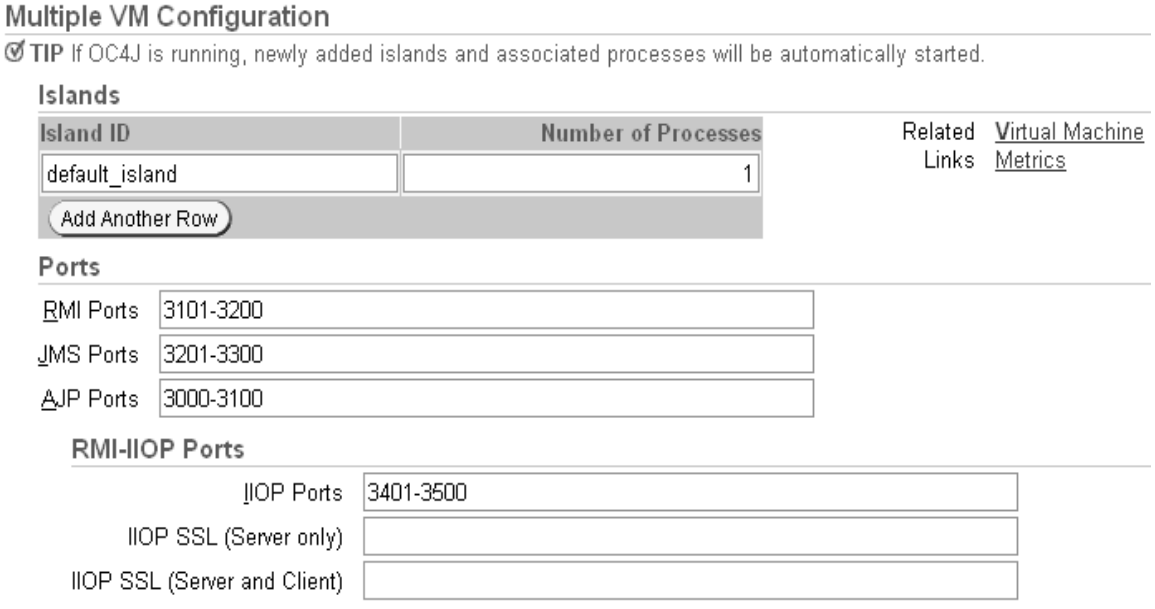
- Default Web module properties—These are specified in an XML file called `global-web-application.xml`. If you want it to refer to another XML file, you can change the pointer to this file here. However, this file must conform to the DTD that Oracle specifies. The directory is relative to `j2ee/home/config`.

If you want to actually modify elements contained within this file, update entries in either the Web Site Properties or Advanced Properties section. These are discussed more in "Configure a Web Site" on page 3-6 and "Modifying XML Files Included in the Deployed Application EAR File" on page 3-20.

- Application and deployment directories—The default directory to place the "master" EAR file of the deployed application is the `/applications` directory. The default directory is where OC4J places modified module deployment descriptors with added defaults. Currently, this location is in the `/application-deployments` directory. You can change the locations of the default directories in these fields. The directory is relative to `j2ee/home/config`. See "What Happens When You Deploy?" on page 3-28 for more information on the usage of these directories.

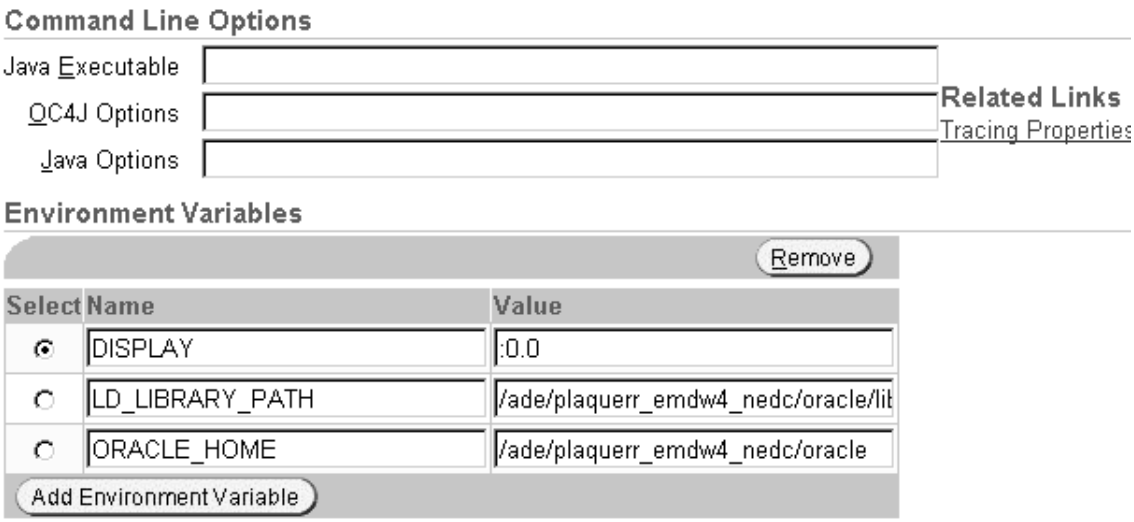
The next section, Multiple VM Configuration, is dedicated as part of the cluster configuration. The following details what each field means; however, the context of how to configure clusters using this section is discussed fully in Chapter 8, "OC4J Clustering".

Figure 3–2 Clustering and Ports



- Islands—Designate the number of islands within the cluster. Each island is created when you click on the Add Another Row button. You can supply a name for each island within the Island ID field. You can designate how many OC4J processes should be started within each island by the number configured in the Number of Processes field. For more information on configuring islands for clustering, see "OC4J Cluster Configuration" on page 8-11.
- Ports—This section enables you to configure what the port ranges should be for RMI, JMS, AJP, and IIOP. For more information on these services, see the *Oracle Application Server Containers for J2EE Services Guide*.

Figure 3–3 Command-Line Options and Environment Variables



- Command Line Options—This section enables you to configure the following:
  - the Java executable command that should be used, such as `javac`
  - any OC4J options to include when starting a new OC4J process
  - any Java options to include when executing '`java`'
  - any OC4J system properties

  See "OC4J Command-Line Options and System Properties" on page A-41 for a list of command-line options and system properties.

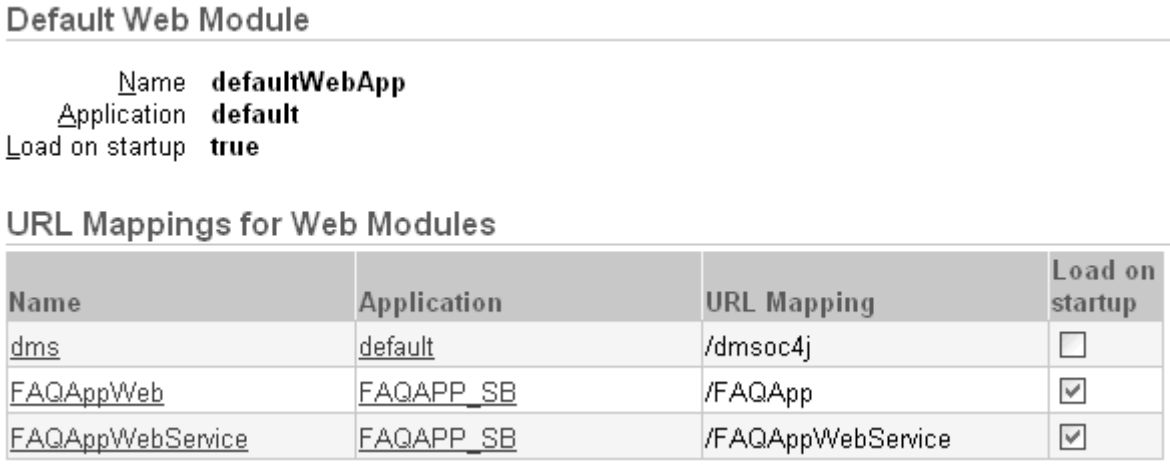
- Environment Variables—This section enables you to configure environment variables for OC4J. To add a new environment variable, click **Add Environment Variable**. A new row is added for you to define the variable name in the left column and the value in the right column.

Configure a Web Site

To configure your Web site, select Website Properties in the Instance Properties column on the Administration page.

The Web site page has two sections. In the first section, you can see what is the default Web application. In the second section—URL Mappings for Web Modules—you can specify whether each Web application is to be loaded upon startup. These parameters are discussed in more detail in the *Oracle Application Server Containers for J2EE Servlet Developer's Guide* and are stored in the `default-web-site.xml` file.

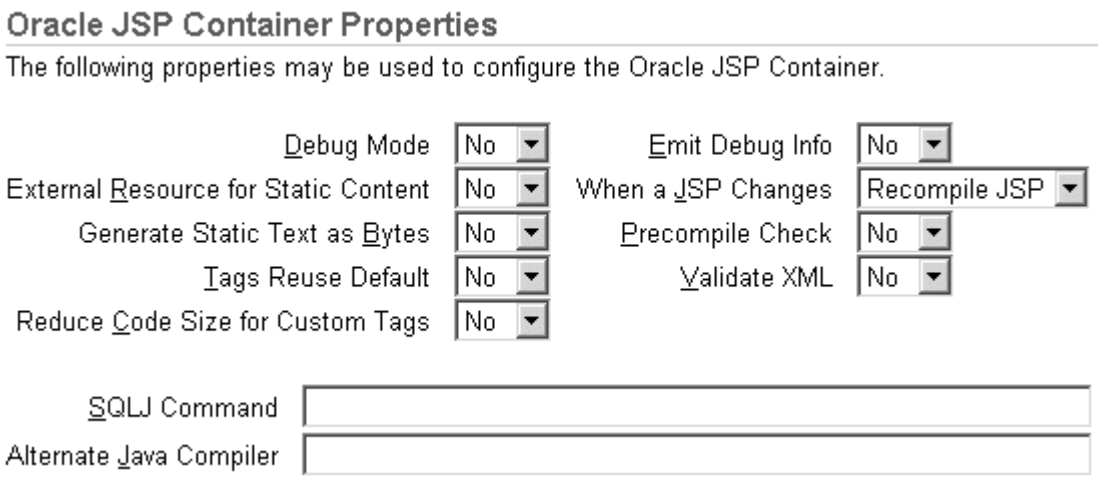
Figure 3–4 Default Web Module Properties



Configure JSP Container Parameters

You can configure global JSP Container parameters. These apply to all JSPs deployed in this OC4J instance. To configure JSP Container parameters, select JSP Container Properties in the Instance Properties column on the Administration page. This brings you to the following page:

Figure 3–5 Oracle JSP Container Properties



Most of the properties indicated here are described in Chapter 3 of the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide.* These properties can be included within the `global-web-application.xml` file within the `<servlet>` element.

Configure Replication Parameters

For clustering servlet or EJBs, you may need to configure replication parameters. See "OC4J Instance Configuration" on page 8-11 for details.

Advanced Configuration Through XML Files

In OC4J version 1.0.2.2, you configured the OC4J server and all deployed application parameters through XML files. Since the OC4J server existed on a single node and did not need high availability and clustering management, this worked well. However, with the integration of OC4J with Oracle Application Server, increased enterprise management abilities with clustering and high availability options, all configuration must be accomplished through Oracle Enterprise Manager.

When you select Advanced Properties off of the Administration page, you can modify the OC4J Server XML files. These include the XML files that configure the server and its services. The files that are in this group are `server.xml`, `global-web-application.xml`, `rmi.xml`, `jms.xml`, and `default-web-site.xml`.

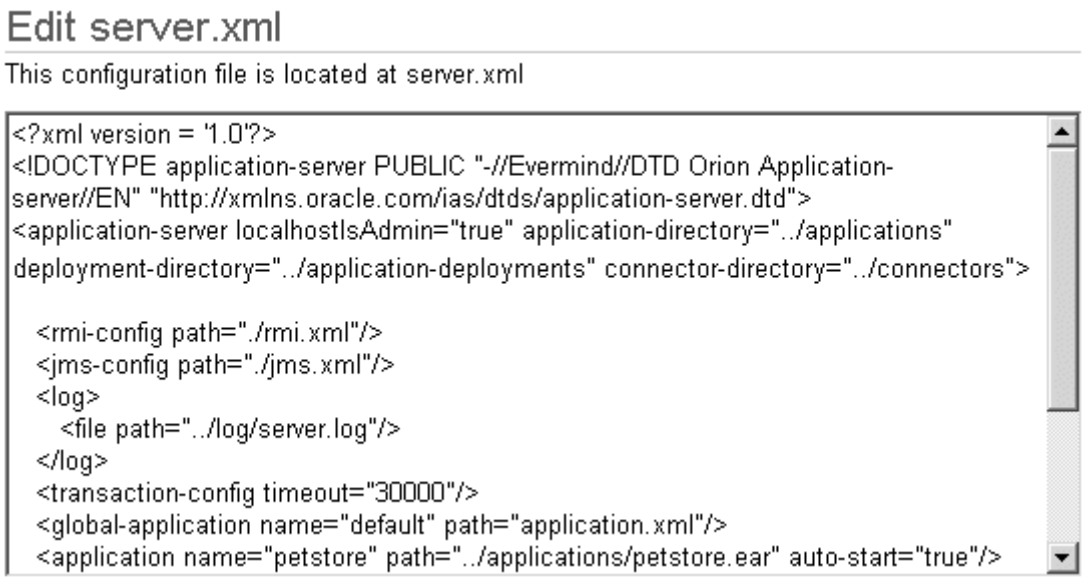
Modify any of these XML files in the Advanced Properties page off of the OC4J Home Page.

Other XML configuration files can be modified in other areas of the Application Server Control.

- Global application XML files: These include XML files that apply to all applications deployed in the OC4J instance. These include the global `application.xml`, `data-sources.xml`, the security XML file and `oc4j-connectors.xml`. To modify these XML files, select Applications off of the OC4J Home Page. On the Applications page, select default. On the default application page, scroll down to the Administration section and choose Advanced Properties.
- Local application XML files. You can modify XML files that configure the overall application. These include local data sources, local security, and OC4J-specific application configuration. These XML files include `data-sources.xml`, `orion-application.xml`, and security XML files. To modify these files, drill down to the specific application from the Deployed Applications section on the Applications page. On the specified application screen, scroll down to the Administration section and choose Advanced Properties.
- Application module XML files: When the EAR or WAR file is deployed, you provided module deployment descriptors, such as `web.xml`, `orion-web.xml`, `ejb-jar.xml`, and `orion-ejb-jar.xml`. You can modify parameters only in the OC4J-specific (`orion-xxx.xml`) XML files. You cannot modify the J2EE XML files, such as `web.xml` or `ejb-jar.xml`. For more information on modifying these XML files, see "Modifying XML Files Included in the Deployed Application EAR File" on page 3-20.

As an example, the `server.xml` page is shown. Notice that you can hand edit the XML elements.

Figure 3–6 Editing the SERVER.XML File



If you do not understand the OC4J XML files, see "Overview of OC4J and J2EE XML Files" on page 3-21 for a discussion of these files and their relation to each other. Other books in the OC4J documentation set describe the elements within each of these files.

Configure Data Sources

You can configure global or local data sources. A global data source is available to all deployed applications in this OC4J instance. A local data source is configured within the deployed application and can only be used by that application.

See *Oracle Application Server Containers for J2EE Services Guide* for a full explanation of how to configure a data source and the elements within the `data-sources.xml` file.

To configure global data sources, select one of the following off of the OC4J Home Page:

- Data Sources under the Application Defaults column on the Administration page—This page allows you to add data source definitions one field at a time. See "Data Source Field Page" on page 3-10 for a description of this page.

- Off the default application page—You can either modify or add a data source either through a GUI or by directly accessing the `data-sources.xml` file.

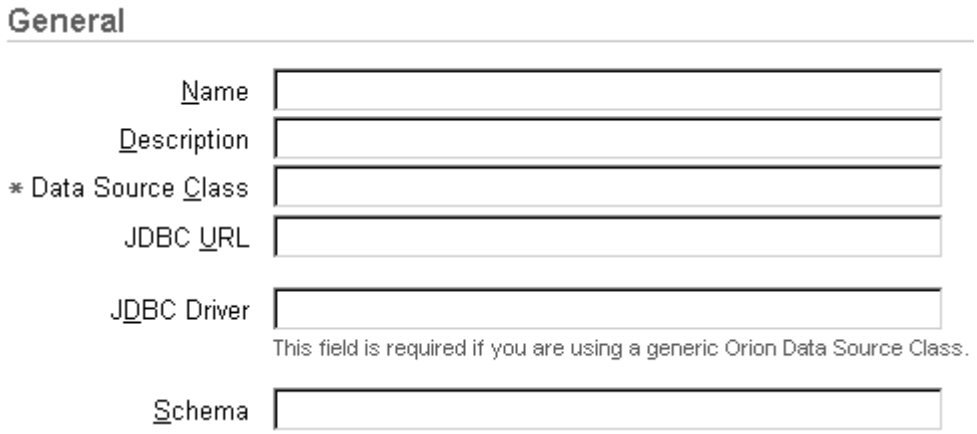
  1. Select Applications off of the OC4J Home Page.
  2. On the Applications page, select default next to the Default Application Name.
  3. On the default application page, scroll down to the Administration section and do one of the following:
     - Select Data Sources under the Resources column. This allows you to add or modify a data source using a GUI form.
     - Select Advanced Properties under the Properties column. Select `data-sources.xml` on this page. This allows you to add data sources using the XML definitions. This is useful if you have been provided the XML. You can just copy in the already configured data sources.

To configure local data sources, you perform the same selection off of the application page. You drill down to the particular application to which this data source is local. On the application page, choose Data Source under the Resources column. It displays the same data source field page that is discussed in "Data Source Field Page" on page 3-10.

Data Source Field Page To configure a new Data Source, click **Add Data Source**. This brings you to a page where you can enter all configuration details about the data source. This page is divided up into four sections.

Figure 3–7 shows the General section.

Figure 3–7 General Section of Data Source Definition

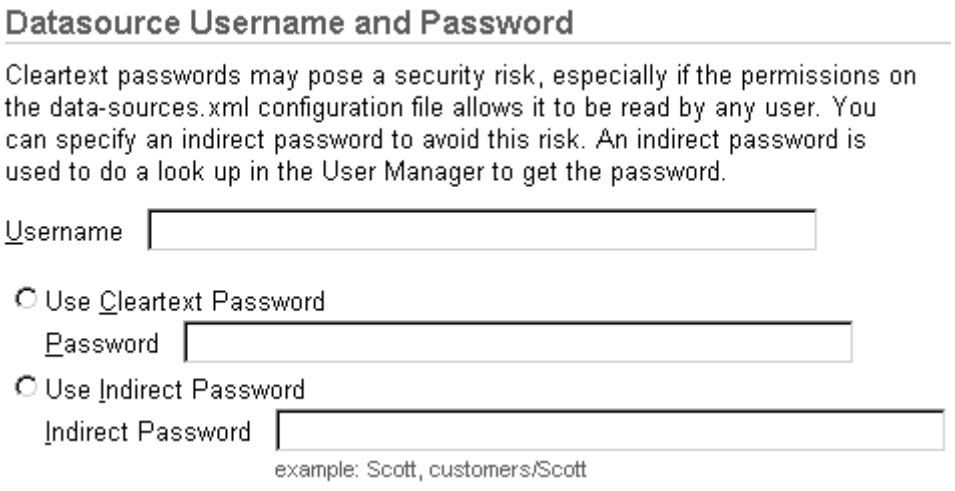


The General section enables you to define the following aspects about a data source:

- Name—A user-defined name to identify the data source.
- Description—A user-defined description of the data source.
- Data Source Class—This is the class, such as `com.evermind.sql.DriverManagerDataSource`, that the data source is instantiated as.
- JDBC URL—The URL to the database represented by this data source. For example, if using an Oracle Thin driver, the URL could be the following: `jdbc:oracle:thin:@my-lap:1521:SID`.
- JDBC Driver—The JDBC driver to use. One example of a JDBC driver is `oracle.jdbc.driver.OracleDriver`.
- Schema—This is an optional parameter. Input the file name that contains the Java to database mappings for a particular database.

Figure 3–8 shows the username and password.

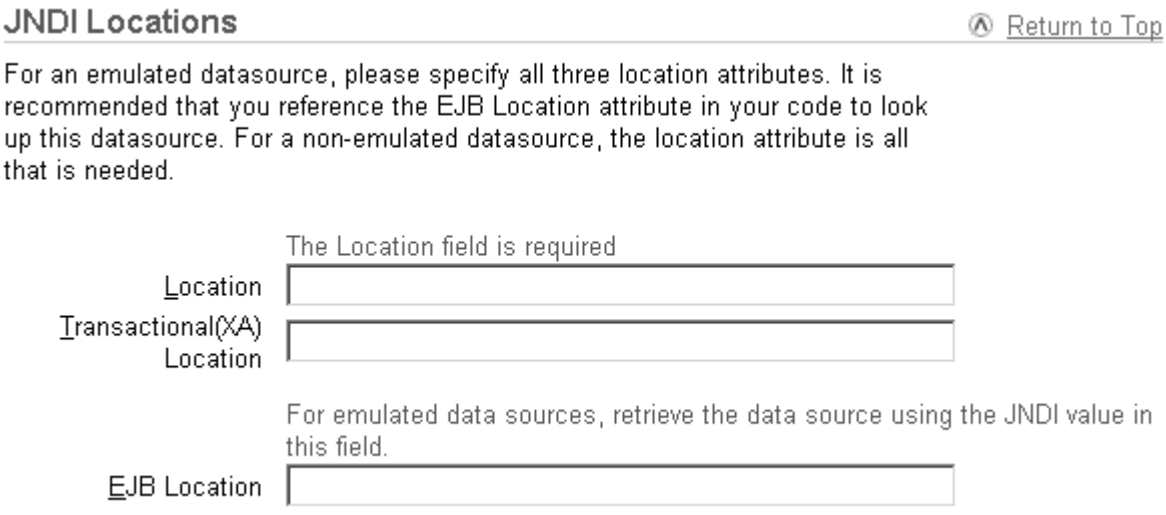
Figure 3–8 Username and Password



Username / Password—The username and password used to authenticate to the database that this data source represents. The password can either be entered as clear text, or you can provide a username for an indirect password. For details, see the *Oracle Application Server Containers for J2EE Services Guide.*

Figure 3–9 shows the JNDI Locations section.

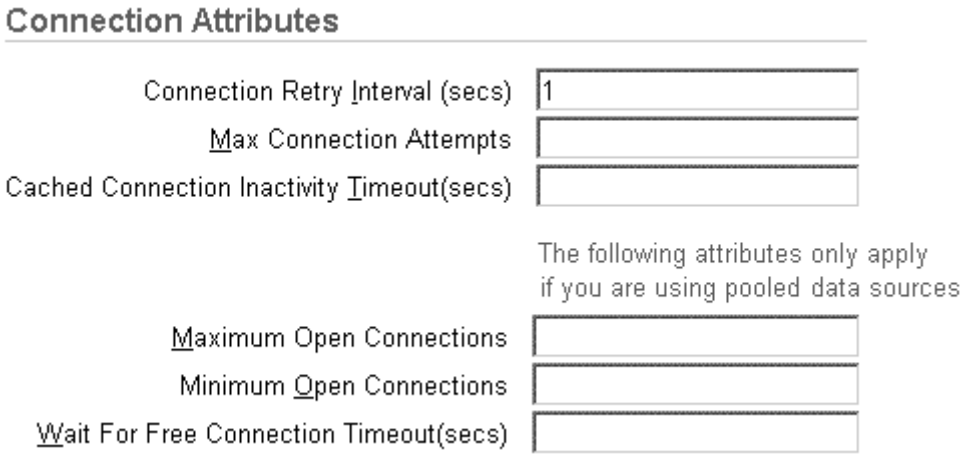
Figure 3–9 JNDI Locations



The JNDI Locations section enables you to define the JNDI location string that the data source is bound with. This JNDI location is used within the JNDI lookup for retrieving this data source.

Figure 3–10 shows the Connection Attributes section.

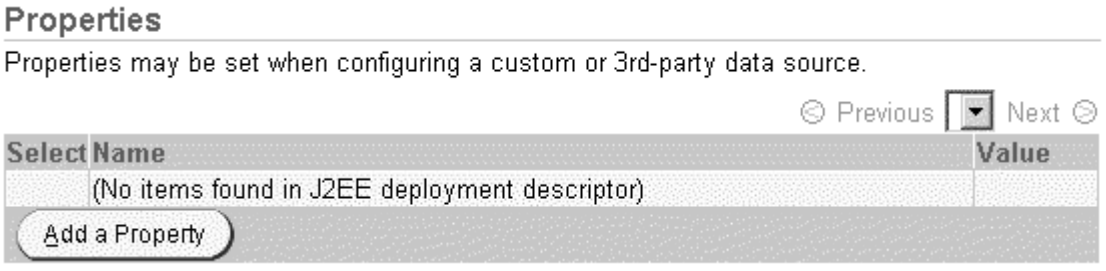
Figure 3–10 Connection Attributes



This section enables you to modify connection tuning parameters, including the retry interval, pooling parameters, timeout parameters, and maximum attempt parameter.

Figure 3–11 shows the Properties section for the data source.

Figure 3–11 Properties



If your data source is a third party data source, you may need to set certain properties. These properties would be defined in the third-party documentation. In addition, properties must be set for JTA transactions for the two-phase commit coordinator.

Configure Security

The user manager, employing the user name and password, verifies the user's identity based on information in the user repository. The user manager defines what type of authentication you will be using. It contains your definitions for users, groups, or roles. The default user manager is the `JAZNUserManager`. You can define a user manager for all applications or for specific applications.

See the *Oracle Application Server Containers for J2EE Security Guide* for a full description of OC4J security, including user managers.

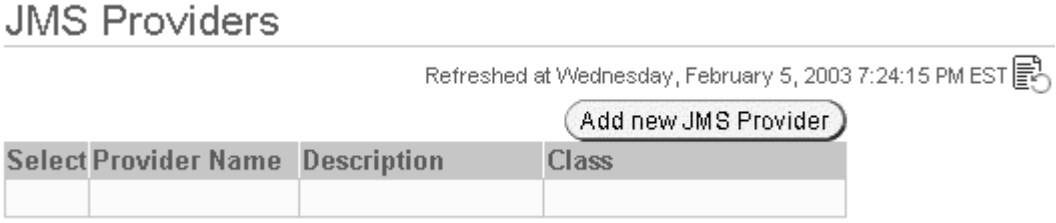
Configure JMS

JMS can be configured either within the JMS section or directly within the `jms.xml` file, as follows:

- Add Oracle JMS or Third-Party Providers by Editing the JMS Section
- Add Queues or Topics for OC4J JMS by Editing the JMS XML File

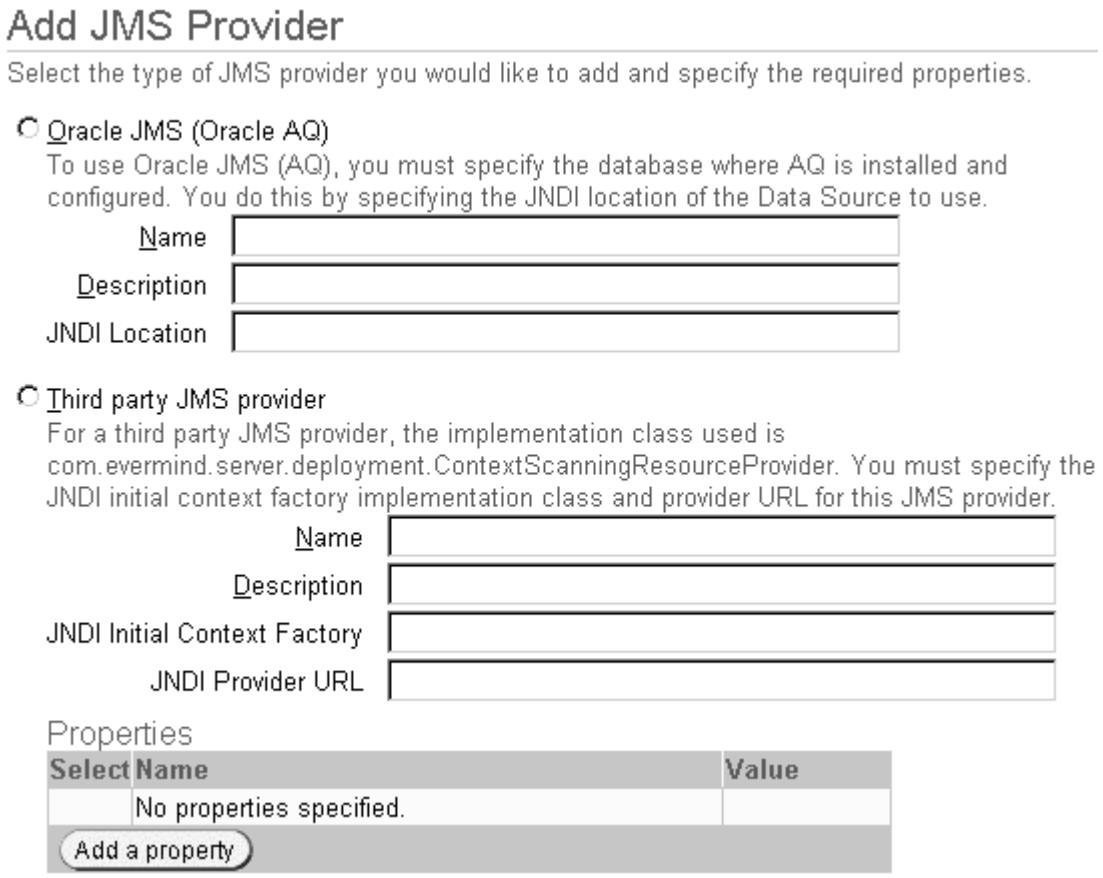
Add Oracle JMS or Third-Party Providers by Editing the JMS Section To add Oracle JMS or third-party JMS providers, select JMS Providers under the Application Defaults column on the Administration page. This brings you to the following page:

Figure 3–12 JMS Providers



Click the **Add new JMS Provider** button to configure each JMS provider, which brings up the following page:

Figure 3–13 Adding a JMS Provider



This page enables you to configure either Oracle JMS or a third-party JMS provider. OC4J JMS is always provided and preconfigured, except for the topics and queues, with the OC4J installation.

Once you choose the typs of JMS provider, you must provide the following:

- Oracle JMS: Provide the data source name and JNDI location for the database where Oracle JMS is installed and configured.
- Third-party JMS provider: Provide the name, JNDI initial context factory class, and JNDI URL for the third-party provider. To add JNDI properties for this JMS provider, such as `java.naming.factory.initial` and

`java.naming.provider.url`, click **Add a property**. A row is added where you can add the name for each JNDI property and its value.

This only configures the providers; it does not configure the `Destination` objects (topic, queue, and subscription). See the *Oracle Application Server Containers for J2EE Services Guide* or more information on JMS providers.

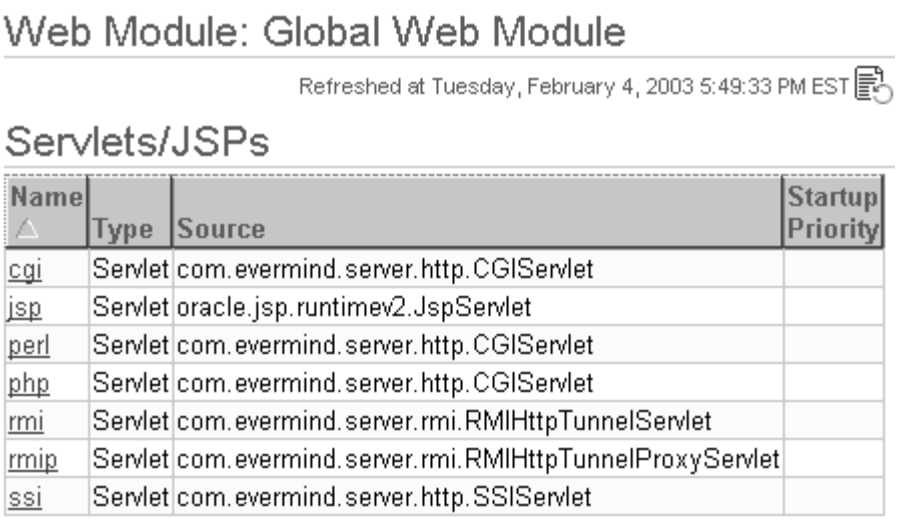
To configure a JMS provider that is only for a specific application, select the application from the Applications page, scroll down to the Resources column, and select JMS Providers. The screens that appear are the same as for the default JMS provider.

Add Queues or Topics for OC4J JMS by Editing the JMS XML File To add queues and topics for OC4J JMS, you can edit the `jms.xml` file directly as follows: select the Advanced Server Properties section under the Instance Properties column on the Administration page. In this section, choose `jms.xml` to modify the straight XML file. See the *Oracle Application Server Containers for J2EE Services Guide* or descriptions of the elements in the `jms.xml` file.

Configure Global Web Application Parameters

To configure Web parameters that apply to all deployed Web applications, select Global Web Module in the Application Defaults column on the Administration page. This brings you to the following page:

Figure 3–14 Global Web Module



Web Module: Global Web Module

Refreshed at Tuesday, February 4, 2003 5:49:33 PM EST

Servlets/JSPs

Name	Type	Source	Startup Priority
cgi	Servlet	com.evermind.server.http.CGIServlet	
jsp	Servlet	oracle.jsp.runtimev2.JspServlet	
perl	Servlet	com.evermind.server.http.CGIServlet	
php	Servlet	com.evermind.server.http.CGIServlet	
rmi	Servlet	com.evermind.server.rmi.RMIHttpTunnelServlet	
rmip	Servlet	com.evermind.server.rmi.RMIHttpTunnelProxyServlet	
ssi	Servlet	com.evermind.server.http.SSIServlet	

The type of parameters that you can define for Web modules concern mappings, filtering, chaining, environment, and security. Drill down into each of the provided links under the Properties and Security columns to modify these parameters. Each of these parameters are discussed in detail in the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*. These parameters are stored in the `global-web-application.xml` and `orion-web.xml` files. This guide discusses the elements in these files.

Configure RMI

RMI can only be defined within an XML definition. To edit the `rmi.xml` file, select Advanced Properties under the Instance Properties column on the Administration page. In this section, choose `rmi.xml` to modify the XML file. See the *Oracle Application Server Containers for J2EE Services Guide* for descriptions of the elements in the `rmi.xml` file.

Application Level Configuration

You can deploy, redeploy, or undeploy a J2EE application that exists in an EAR or WAR file archival format. To deploy an application, click the **Deploy EAR File** or **Deploy WAR File** buttons to deploy in the Deployed Applications section on the Applications page.

This starts the deployment wizard that is covered in "Deploying Applications" on page 2-14. If you deploy an EAR file, it must contain an `application.xml` that describes the application modules; if you deploy a WAR file, the `application.xml` file is created for you automatically.

To undeploy, click the **Select** radio button for the application and then click the **Undeploy** button.

To redeploy, click the **Select** radio button for the application and then click the **Redeploy** button.

> **Note:** You can also deploy, undeploy, or redeploy simple applications with the DCM command. See the *Distributed Configuration Management Reference Guide* for directions.

Once you have deployed your application, you can modify most of the parameters for this application. To configure application-specific parameters, do the following:

1. On the OC4J Home Page, select the Applications page.
2. Select the application where you want to change the configuration using one of the following methods:
   a. Click the **Select** radio button for the application and click the **Edit** button.
   b. Select the application name in the Name column in the applications box.

This page is the initiating point for changing general application configuration as well as configuration specific to a certain part of your deployed application, such as a WAR file.

The following sections provide a brief overview of these configuration options:

- Configuring Application General Parameters
- Configuring Local J2EE Services
- Modifying XML Files Included in the Deployed Application EAR File

Configuring Application General Parameters

If you drill down to the application, scroll down to the Properties column and select the General link, you can configure a multitude of application details, as follows:

- persistence path
- data sources path

- library paths
- EJB properties
  - automatically create database tables for CMP beans
  - automatically delete old database tables for CMP beans
- default data source (JNDI name)
- User Manager configuration

Configuring Local J2EE Services

As described in "Configure Data Sources" on page 3-9 and "Configure Security" on page 3-15, you can configure data sources and security either for all deployed applications (global) or only for a specific application (local). See these sections for directions on how to configure your J2EE services for your application.

Modifying XML Files Included in the Deployed Application EAR File

You can modify only the OC4J-specific XML files of your application after deployment. This includes `orion-ejb-jar.xml`, `orion-web.xml`, and `orion-application-client.xml`. You cannot modify the J2EE XML files, such as `web.xml`, `ejb-jar.xml`, and `application-client.xml`.

In order to modify the OC4J-specific XML files, do the following:

1. From the application screen, select the JAR or WAR file whose configuration you are interested in modifying. The application screen is shown.
2. You can modify parameters for the application in one of the following manners:
   - Follow links in the Administration section for modifying parameters.
   - Select the bean or servlet in the section that details the beans, servlets, or JSPs deployed. This drills down to another level of configuration.
   - The Administration section contains either a Properties or Advanced Properties section that allows you to modify XML directly for the OC4J-specific deployment descriptors—`orion-ejb-jar.xml`, `orion-web.xml`, and `orion-application-client.xml`.

Overview of OC4J and J2EE XML Files

This section contains the following topics:

- XML Configuration File Overview
- XML File Interrelationships

XML Configuration File Overview

Each XML file within OC4J exists to satisfy a certain role; thus, if you have need of that role, you will understand which XML file to modify and maintain.

Figure 3–15 illustrates all the OC4J XML files and their respective roles.

- OC4J server: All XML files within this box are used to set up this instance of the OC4J server. These files configure things such as listening ports, administration passwords, security, and other basic J2EE services.

  These files configure the OC4J server and point to other key configuration files. The settings in the OC4J configuration files are not related to the deployed J2EE applications directly, but to the server itself.

- Oracle HTTP Server: These files are configuration files within the Oracle HTTP Server. However, they are included in this diagram because you may need to modify these to change how requests are handed off to the OC4J server.

- Web site: These XML files configure listening ports, protocols, and Web contexts for the OC4J Web site.

- Application XML files: Each J2EE application type (EJB, servlet, JSP, connector) requires its own configuration (deployment) files. Each application type has one J2EE deployment descriptor and one OC4J-specific deployment descriptor, which is denoted with an "`orion-`" prefix. In addition, the following are global configuration files for all components in the application:

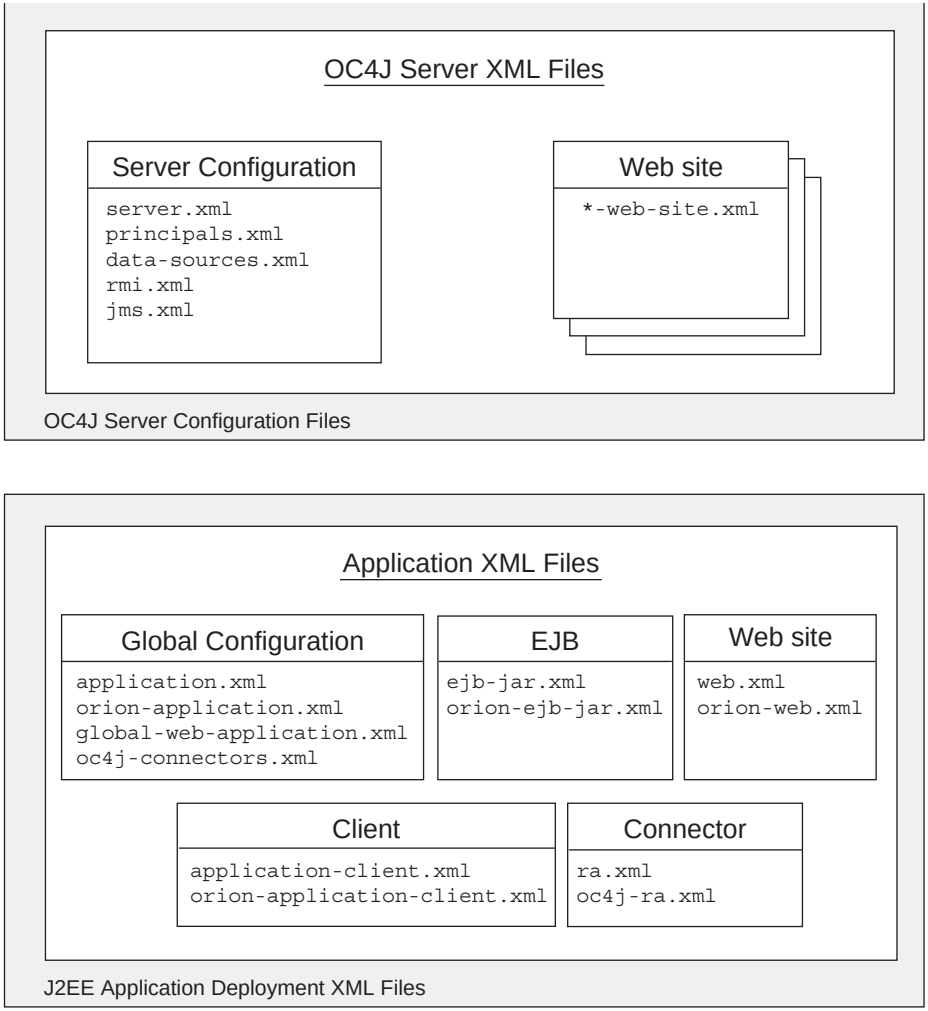
  – The `application.xml` as the global application configuration file that contains common settings for all applications in this OC4J instance.

  – The `orion-application.xml` file contains OC4J-specific global application information for all applications in this OC4J instance.

  – The `global-web-application.xml` file contains OC4J-specific global Web application configuration information that contains common settings for all Web modules in this OC4J instance.

– The `oc4j-connectors.xml` file contains global connector configuration information.

Figure 3–15 OC4J and J2EE Application Files



Note: Each deployed application uses an `application.xml` as its standard J2EE application descriptor file. That XML file is local to the application and separate from the global `application.xml`, which configures options that are applied to all applications deployed in this OC4J server instance.

Table 3–1 describes the role and function for each XML file that was displayed in the preceding figure.

Table 3–1 OC4J Features and Components

XML Configuration File	Features/Components
`server.xml`	OC4J overall server configuration. Configures the server and points to the XML files that add to this file, such as `jms.xml` for JMS support. The listing of other XML files enables the services to be configured in separate files, but the `server.xml` file denotes that they be used for the OC4J configuration.
`jazn.xml` `jazn-data.xml`	OC4J security configuration for OracleAS JAAS Provider security required for accessing the server.
`data-sources.xml`	OC4J data source configuration for all databases used by applications within OC4J.
`rmi.xml`	OC4J RMI port configuration and RMI tunneling over HTTP.
`jms.xml`	OC4J JMS configuration for `Destination` topics and queues that are used by JMS and MDBs in OC4J.
`default-web-site.xml`	OC4J Web site definition.
`mod_oc4j.conf`	The `mod_oc4j` module is an Oracle HTTP Server module that forwards OC4J requests. This file configures the mount point that denotes what contexts to be directed to OC4J.

Table 3–1 OC4J Features and Components (Cont.)

XML Configuration File	Features/Components
application.xml orion-application.xml	J2EE application standard J2EE application descriptor file and configuration files. ■ The global application.xml file exists in the j2ee/home/config directory and contains common settings for all applications in this OC4J instance. This file defines the location of the security XML definition file—jazn-data.xml and the datasource XML definition file—data-sources.xml. This is a different XML file than the local application.xml files. ■ The local application.xml file defines the J2EE EAR file, which contains the J2EE application modules. This file exists within the J2EE application EAR file. ■ The orion-application.xml file is the OC4J-specific definitions for all applications.
global-web-application.xml web.xml orion-web.xml	J2EE Web application configuration files. ■ global-web-application.xml is an OC4J-specific file for configuring servlets that are bound to all Web sites. ■ web.xml and orion-web.xml for each Web application. The web.xml files are used to define the Web application deployment parameters and are included in the WAR file. In addition, you can specify the URL pattern for servlets and JSPs in this file. For example, servlet is defined in the <servlet> element, and its URL pattern is defined in the <servlet-mapping> element.
ejb-jar.xml orion-ejb-jar.xml	J2EE EJB application configuration files. The ejb-jar.xml files are used to define the EJB deployment descriptors and are included in the EJB JAR file.
application-client.xml orion-application-client.xml	J2EE client application configuration files.

Table 3–1 OC4J Features and Components (Cont.)

XML Configuration File	Features/Components
oc4j-connectors.xml ra.xml oc4j-ra.xml	Connector configuration files. ■ The oc4j-connectors.xml file contains global OC4J-specific configuration for connectors. ■ The ra.xml file contains J2EE configuration. ■ The oc4j-ra.xml file contains OC4J-specific configuration.

XML File Interrelationships

Some of these XML files are interrelated. That is, some of these XML files reference other XML files—both OC4J configuration and J2EE application (see Figure 3–17).

Here are the interrelated files:

- server.xml—contains references to the following:
  - All *-web-site.xml files for each Web site for this OC4J server, including the default default-web-site.xml file.
  - The location of each of the other OC4J server configuration files, except jazn-data.xml and data-sources.xml which are defined in the global application.xml, shown in Figure 3–15
  - The location of each application.xml file for each J2EE application that has been deployed in OC4J
- default-web-site.xml—references applications by name, as defined in the server.xml file. And this file references an application-specific EAR file.
- application.xml—contains references to the jazn-data.xml and data-sources.xml files.

The server.xml file is the keystone that contains references to most of the files used within the OC4J server. Figure 3–16 shows the XML files that can be referenced in the server.xml file:

Figure 3–16 XML Files Referenced Within server.xml

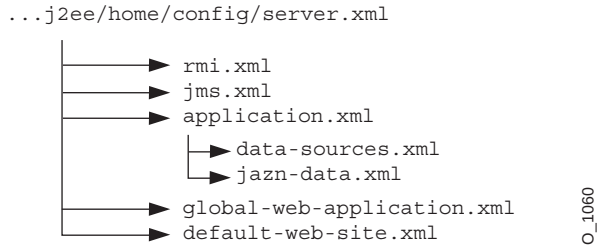
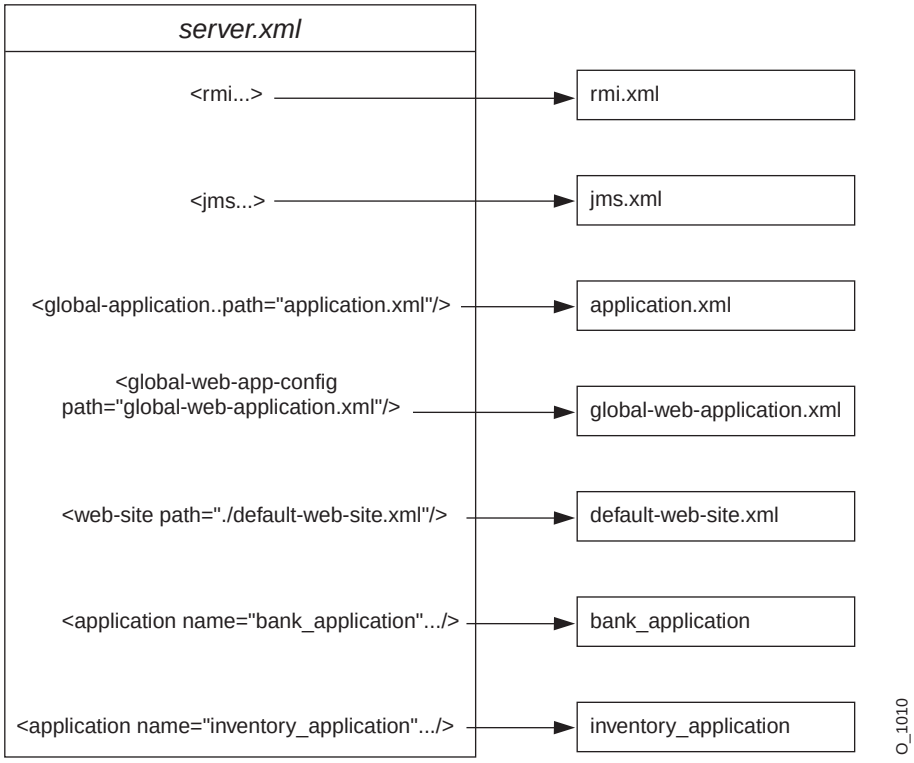


Figure 3–17 demonstrates how the `server.xml` points to other XML configuration files. For each XML file, the location can be the full path or a path relative to the location of where the `server.xml` file exists. In addition, the name of the XML file can be any name, as long as the contents of the file conform to the appropriate DTD.

- The `<rmi-config>` tag denotes the name and location of the `rmi.xml` file.
- The `<jms-config>` tag denotes the name and location of the `jms.xml` file.
- The `<global-application>` tag denotes the name and location of the global `application.xml` file.
- The `<global-web-app-config>` tag denotes the name and location of the `global-web-application.xml` file.
- The `<web-site>` tag denotes the name and location of one `*-web-site.xml` file. Since you can have multiple Web sites, you can have multiple `<web-site>` entries.

In addition to pointing to the OC4J server configuration files, the `server.xml` file describes the applications that have been deployed to this OC4J server. Each deployed application is denoted by the `<application>` tag.

Figure 3–17 Server.xml File and Related XML Files



Other tags for `server.xml` are described in "Elements in the server.xml File" on page 3-21.

Note: If you understand the OC4J XML files from previous releases of OC4J, you can simply change most of the OC4J server XML configuration files by drilling to the OC4J Home Page, scroll down to Administration, and click on Advanced Properties. From here, you can modify the XML files using an Oracle Enterprise Manager editor.

What Happens When You Deploy?

When you become more proficient with OC4J and deploying applications, you should acquaint yourself with what OC4J does for you. The following sections help you understand these tasks:

- OC4J Tasks During Deployment
- Configuration Verification of J2EE Applications

OC4J Tasks During Deployment

When you deploy your application, the following occurs:

OC4J opens the EAR file and reads the descriptors.

1. OC4J opens, parses the `application.xml` that exists in the EAR file. The `application.xml` file lists all of the modules contained within the EAR file. OC4J notes these modules and initializes the EAR environment.
2. OC4J reads the module deployment descriptors for each module type: Web module, EJB module, connector module, or client module. The J2EE descriptors are read into memory. If OC4J-specific descriptors are included, these are also read into memory. The JAR and WAR file environments are initialized.
3. OC4J notes any unconfigured items that have defaults and writes these defaults in the appropriate OC4J-specific deployment descriptor. Thus, if you did not provide an OC4J-specific deployment descriptor, you will notice that OC4J provides one written with certain defaults. If you did provide an OC4J-specific deployment descriptor, you may notice that OC4J added elements.
4. OC4J reacts to the configuration details contained in both the J2EE deployment descriptors and any OC4J-specific deployment descriptors. OC4J notes any J2EE component configurations that require action on OC4J's part, such as wrapping beans with their interfaces.
5. After defaults have been added and necessary actions have been taken, the new module deployment descriptors are written to the `application-deployments/` directory. These are the descriptors that OC4J uses when starting and restarting your application. But do not modify these descriptors. Always change your deployment descriptors in the "master" location.
6. OC4J copies the EAR file to the "master" directory. This defaults to the `applications/` directory. You can change the "master" directory in the Server Properties page off of the OC4J Home Page. In the General section, modify the

Application Directory field to the new location of the "master" directory. The location of the directory is relative to `/j2ee/home/config`.

> **Note:** Each time you deploy this EAR file without removing the EAR file from the `applications/` directory, the new deployment renames the EAR file prepended with an underscore. It does not copy over the EAR file. Instead, you can copy over the EAR file. OC4J notices the change in the timestamp and redeploys.

7. Finally, OC4J updates the `server.xml` file with the notation that this application has been deployed.

Configuration Verification of J2EE Applications

After deployment, you can see your application configuration in the `server.xml` and `default-web-site.xml` files, as follows:

- In `server.xml`, each existing application contains a line with an `<application name=... path=... auto-start="true" />` entry. The `auto-start` attribute designates that you want this application automatically started when OC4J starts. The path is either the location of the EAR file to be deployed or the exploded directory where the applicatoin has been built. See "Basic Deployment" on page 2-14 or "Building and Deploying Within a Directory" on page 3-33 for more information.
- In `default-web-site.xml`, a `<web-app...>` entry exists for each Web application that is bound to the Web site upon OC4J startup. Because the `name` attribute is the WAR filename (without the `.WAR` extension), there is one line for each WAR file included in your J2EE application.

  For each Web application binding included in a WAR file, the following line has been added:

  ```
  <web-app application="myapp" name="myapp-web" root="/myapp" />
  ```

  - The `application` attribute is the name provided in the `server.xml` as the application name.
  - The `name` attribute is the name of the WAR file, without the `.WAR` extension.
  - The `root` attribute defines the root context for the application off of the Web site. For example, if you defined your Web site as

"http://<ohs_host>:7777/j2ee", then to initiate the application, point your browser at "http://<ohs_host>:7777/j2ee/myapp".

Note: Wait for automatic startup to complete before trying to access the client. The client fails on lookup if it tries to access before the completion of these processes.

Sharing Libraries

If you have libraries that you want to share among applications, add a <library> element in the global application.xml file, indicating the directory where you are placing the libraries, as follows:

Windows:

```
<library path="d:\oc4j\j2ee\home\applib\"/>
```

UNIX:

```
<library path="/private/oc4j/j2ee/home/applib/"/>
```

For each directory to be included, use a separate <library> element on a separate line, as follows:

```
<library path="/private/oc4j/j2ee/home/applib/"/>
<library path="/private/oc4j/j2ee/home/mylibrary/"/>
```

As a default, a <library> element exists in the global application.xml file with the j2ee/home/applib directory. Instead of modifying the <library> element to contain other directories, you could move your libraries into the applib directory. However, note that adding libraries to this directory increases the size of OC4J and effects the performance as all libraries are searched for unknown classes. Use this with discretion.

Note: The default j2ee/home/applib directory is not created when OC4J is installed. If you want to add shared libraries to this directory, you must first create it before adding your libraries.

If you can, you should keep your shared libraries local to the application through the orion-application.xml file deployed with the application. You can add <library> elements in the orion-application.xml file for the application to indicate where the libraries are located, which are used only within the application.

Understanding and Configuring OC4J Listeners

Incoming client requests use one of three protocols: AJP, HTTP, or RMI. AJP and HTTP are used for HTTP requests. AJP is used between the OHS and OC4J components. HTTP is used for HTTP requests directed past OHS to OC4J. RMI is used for incoming EJB requests.

HTTP Requests

All HTTP requests, whether through OHS or directly to OC4J, must have a listener configured in an OC4J Web site. You can have two Web sites for each OC4J instance: one for each protocol type. That is, one Web site is only for AJP requests and the other is for HTTP requests. You cannot have one Web site listen for both types of protocols. Thus, OC4J provides two Web site configuration files:

- `default-web-site.xml`—This is the AJP protocol listener and the default for most HTTP requests that use Oracle Application Server. After installation, the Oracle HTTP Server front-end forwards incoming HTTP requests over the AJP port. The OC4J Web server configuration file (`default-web-site.xml`) indicates the AJP listener port. The `default-web-site.xml` file defines the default AJP port as zero. This enables OC4J and the Oracle HTTP Server to negotiate a port upon startup. The range of port values that the AJP port can be is configured in the OPMN configuration. See the High Availability chapter in the *Oracle Application Server 10g Administrator's Guide* for more information on OPMN.

  The following shows the entry in the `default-web-site.xml` for the default AJP listener:

  ```
  <web-site host="oc4j_host" port="0" protocol="ajp13"
      display-name="Default OC4J WebSite">
  ```

  You can configure the AJP default Web site protocol in two places: Website Properties or Advanced Properties off of the OC4J Home Page.

- `http-web-site.xml`—This is the HTTP protocol listener. If you want to bypass OHS and go directly to OC4J, you use the port number defined in this file. However, you must be careful. The AJP port is chosen at random every time OC4J is started. If it chooses the same port number that is hard-coded in this XML file, there will be a conflict. If you use this method for incoming requests, verify that the port number you choose is outside of the range for AJP port numbers, which is defined in the OPMN configuration.

  The default HTTP port is 7777. The following shows the entry in the `http-web-site.xml` for an HTTP listener with a port number of 7777:

```
<web-site host="oc4j_host" port="7777" protocol="http"
    display-name="HTTP OC4J WebSite">
```

Note: In a UNIX environment, the port number should be greater than 1024, unless the process has administrative privileges.

You access the `http-web-site.xml` file only in the Advanced Properties on the OC4J Home Page.

RMI Requests

The RMI protocol listener is set up by OPMN in the RMI configuration—`rmi.xml`. It is separate from the Web site configuration. EJB clients and the OC4J tools access the OC4J server through a configured RMI port. OPMN designates a range of ports that the RMI listener could be using. When you use the "`opmn:ormi://`" string in the lookup, the client retrieves automatically the assigned RMI port. See "Accessing the EJB" in the EJB Primer chapter in *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.*

Configuring Oracle HTTP Server With Another Web Context

The `mod_oc4j` module in the Oracle HTTP Server is configured at install time to direct all `j2ee/` context bound applications to the OC4J server. If you want to use a different context, such as `pubs/`, you can add another mount for this context in the `mod_oc4j.conf` configuration file.

To modify this file, drill down to the Oracle HTTP Server Page and select `mod_oc4j.conf`. The file is presented for edits in the right-hand frame.

1. Find the `Oc4jMount` directive for the `j2ee/` directory. Copy it to another line. The mount directive is as follows:

   ```
   Oc4jMount /j2ee/* OC4Jworker
   ```

   Note: The OC4Jworker is defined further down in the `mod_oc4j.conf` file to be the OC4J instance.

2. Modify the `j2ee/` context to your desired context. In our example, you would have another line with a `pubs/` mount configuration. Assuming that the OC4J worker name is `OC4Jworker`, then both lines would be as follows:

```
Oc4jMount /j2ee/* OC4Jworker
Oc4jMount /pubs/* OC4Jworker
```

3. Restart the Oracle HTTP Server to pick up the new mount point.

Then all incoming requests for the `pubs/` context will be directed to the OC4J server. Note that when you deploy an application using the deployment wizard, the wizard automatically adds a mount point as described here for your URL mapping.

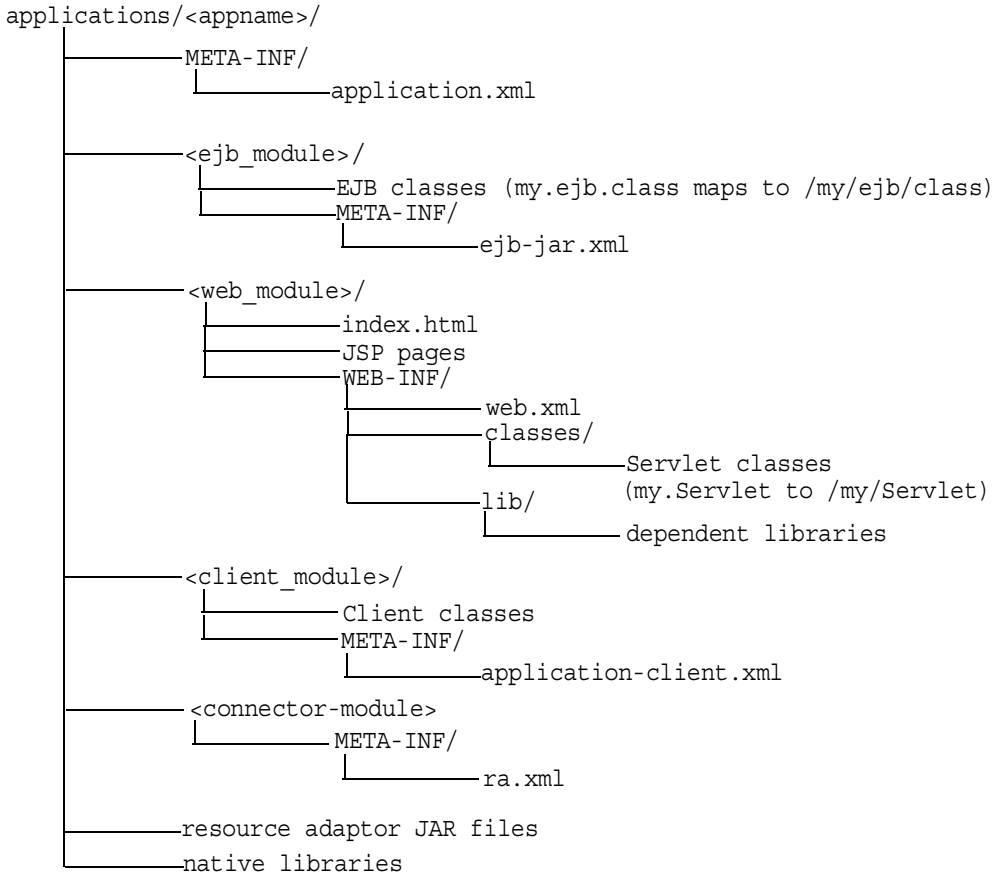
See the *Oracle HTTP Server Administrator's Guide* for complete details on the `mod_oc4j` module configuration.

Building and Deploying Within a Directory

When developing applications, you want to quickly modify, compile, and execute your classes. OC4J can automatically deploy your applications as you are developing them within an expanded directory format. OC4J automatically deploys applications if the timestamp of the top directory, noted by `<appname>` in Figure 3–18, changes. This is the directory that server.xml knows as the "master" location.

The application must be placed in the "master" directory in the same hierarchical format as necessary for JAR, WAR, and EAR files. For example, if `<appname>` is the directory where your J2EE application resides, Figure 3–18 displays the necessary directory structure.

Figure 3–18 Development Application Directory Structure



To deploy EJB or complex J2EE applications in an expanded directory format, complete the following steps:

1. Place the files in any directory. Figure 3–18 demonstrates an application placed into `j2ee/home/applications/<appname>/`. The directory structure below `<appname>` is similar to that used within an EAR file, as follows:

    a. Replace the EJB JAR file name, Web application WAR file name, client JAR file name, and Resource Adapter Archive (RAR) file name with a directory name of your choosing to represent the separate modules. Figure 3–18 demonstrates these directory names by `<ejb_module>/`, `<web_module>/`, `<client_module>/`, and `<connector_module>/`.

b. Place the classes for each module within the appropriate directory structure that maps to their package structure.

2. Modify the `server.xml`, `application.xml`, and `*-web-site.xml` files. The `server.xml` and `*-web-site.xml` files are located in `j2ee/home/config` directory, while the `application.xml` is under `j2ee/home/applications/<appname>/META-INF` directory. Modify these files as follows:

- In `server.xml`, add a new or modify the existing `<application name=... path=... auto-start="true" />` element for each J2EE application. The path points to the "master" application directory. In Figure 3–18, this is `j2ee/home/applications/<appname>/`.

  You can specify the path in one of two manners:

  * Specifying the full path from root to the parent directory.

    In the example in Figure 3–18, if `<appname>` is `"myapp"`, then the fully-qualified path is as follows:

    ```
    <application_name="myapp"
        path="/private/j2ee/home/applications/myapp"
        auto-start="true" />
    ```

  * Specifying the relative path. The path is relative to where the `server.xml` file exists to where the parent directory lives.

    In the example in Figure 3–18, if `<appname>` is `"myapp"`, then the relative path is as follows:

    ```
    <application_name="myapp" path="../applications/myapp"
    auto-start="true" />
    ```

- In `application.xml`, modify the `<module>` elements to contain the directory names for each module—not JAR or WAR files. You must modify the `<web-uri>`, the `<ejb>`, and the `<client>` elements in the `application.xml` file to designate the directories where these modules exist. The path included in these elements should be relative to the "master" directory and the parent of the `WEB-INF` or `META-INF` directories in each of these application types.

  For example, if the `<web_module>/` directory in Figure 3–18 was `"myapp-web/"`, then the following example designates this as the Web module directory within the `<web-uri>` element as follows:

```
<module>
  <web>
    <web-uri>myapp-web</web-uri>
  </web>
</module>
```

- In the *-web-site.xml file, add a <web-app...> element for each Web application. This is important, because it binds the Web application within the Web site. The application attribute value should be the same value as that provided in the server.xml file. The name attribute should be the directory for the Web application. Note that the directory path given in the name element follows the same rules as for the path in the <web-uri> element in the application.xml file.

  To bind the"myapp" Web application, add the following:

  ```
  <web-app application="myapp" name="myapp-web" root="/myapp" />
  ```

  Note: You achieve better performance if you are deploying with a JAR file. During execution, the entire JAR file is loaded into memory and indexed. This is faster than reading in each class from the development directory when necessary.

3. Add mount points for the Web application in the mod_oc4j.conf configuration file. Based on the root="/myapp" in this example, the mod_oc4j.conf should be updated to contain the following lines:

   ```
   Oc4jMount /myapp home
   Oc4jMount /myapp/* home
   ```

Developing Startup and Shutdown Classes

You can develop classes that are called after OC4J initializes or before OC4J terminates. Startup classes can start services and perform functions after OC4J initiates; shutdown classes can terminate these services and perform functions before OC4J terminates. The oc4j.jar must be in the Java CLASSPATH when you compile these classes.

OC4J deploys and executes the OC4J startup and shutdown classes based on configuration of these classes in the server.xml file.

- OC4J Startup Classes
- OC4J Shutdown Classes

OC4J Startup Classes

Startup classes are executed only once after OC4J initializes. They are not re-executed everytime the `server.xml` file is touched. Your startup class implements the `com.evermind.server.OC4JStartup` interface that contains two methods—`preDeploy` and `postDeploy`—in which you can implement code for starting services or performing other initialization routines.

- The `preDeploy` method executes before any OC4J application initialization.
- The `postDeploy` method executes after all OC4J applications initialize.

Each method requires two arguments—a `Hashtable` that is populated from the configuration and a JNDI `Context` to which you can bind to process values contained within the `Context`. Both methods return a `String`, which is currently ignored.

Once created, you must configure the startup class within the `<startup-classes>` element in the `server.xml` file. You access this file by selecting **Advanced Properties** on the OC4J home page. Each `OC4JStartup` class is defined in a single `<startup-class>` element within the `<startup-classes>` element. Each `<startup-class>` defines the following:

- The name of the class that implements the `com.evermind.server.OC4JStartup` interface.
- Whether a failure is fatal. If considered fatal, then when an exception is thrown, OC4J logs the exception and exits. If not considered fatal, then OC4J logs the exception and continues. Default is not fatal.
- The order of execution where each startup class receives an integer number that designates in what order the classes are executed.
- The initialization parameters that contain key-value pairs, of type `String`, which OC4J takes, which are provided within the input `Hashtable` argument. The names for the key-value pairs must be unique, as JNDI is used to bind each value to its name.

In the `<init-library path="../[xxx]" />` element in the `server.xml` file, configure the directory where the startup class resides, or the directory and JAR filename where the class is archived. The `path` attribute can be fully-qualified or relative to `j2ee/home/config`.

Example 3–1 Startup Class Example

The configuration for the TestStartup class is contained within a <startup-class> element in the server.xml file. The configuration defines the following:

- The failure-is-fatal attribute is true, so that an exception causes OC4J to exit.
- The execution-order is 0, so that this is the first startup class to execute.
- Two initialization key-value pairs defined, of type String, which will be populated in the Hashtable, of the following:

```
"oracle.test.startup" "true"
"startup.oracle.year" "2002"
```

> **Note:** The names of the key-value pairs must be unique in all startup and shutdown classes, as JNDI binds the name to its value.

Thus, configure the following in the server.xml file to define the TestStartup class:

```
<startup-classes>
        <startup-class classname="TestStartup" failure-is-fatal="true">
            <execution-order>0</execution-order>
                <init-param>
                    <param-name>oracle.test.startup</param-name>
                    <param-value>true</param-value>
                </init-param>
                <init-param>
                    <param-name>startup.oracle.year</param-name>
                    <param-value>2002</param-value>
                </init-param>
        </startup-class>
 </startup-classes>
```

The container provides the two initialization kay-value pairs within the input Hashtable parameter to the startup class.

The following example shows TestStartup, which implements the com.evermind.server.OC4JStartup interface. The preDeploy method retrieves the key-value pairs from the Hashtable and prints them out. The postDeploy method is a null method. The oc4j.jar must be in the Java CLASSPATH when you compile TestStartup.

```
import com.evermind.server.OC4JStartup;
```

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

public class TestStartup implements OC4JStartup {
    public String preDeploy(Hashtable args, Context context) throws Exception {
        // bind each argument using its name
        Enumeration keys = args.keys();
        while(keys.hasMoreElements()) {
            String key = (String)keys.nextElement();
            String value = (String)args.get(key);
            System.out.println("prop: " + key + " value: " + args.get(key));
            context.bind(key, value);
        }

        return "ok";
    }

    public String postDeploy(Hashtable args, Context context) throws Exception {
        return null;
    }
}
```

Assuming that the `TestStartup` class is archived in `"../app1/startup.jar"`, modify the `<init-library>` element in the `server.xml` file as follows:

```
<init-library path="../app1/startup.jar" />
```

When you start OC4J, the `preDeploy` method is executed before any application is initialized. OC4J populates the JNDI context with the values from the `Hashtable`. If `TestStartup` throws an exception, then OC4J exits since the `failure-is-fatal` attribute was set to `TRUE`.

OC4J Shutdown Classes

Shutdown classes are executed before OC4J terminates. Your shutdown class implements the `com.evermind.server.OC4JShutdown` interface that contains two methods—`preUndeploy` and `postUndeploy`—in which you can implement code for shutting down services or perform other termination routines.

- The `preUndeploy` method executes before any OC4J application terminates.
- The `postUndeploy` method executes after all OC4J applications terminates.

Each method requires two arguments—a `Hashtable` that is populated from the configuration and a JNDI `Context` to which you can bind to process values contained within the `Context`.

The implementation and configuration is identical to the shutdown classes as described in "OC4J Startup Classes" on page 3-37 with the exception that the configuration is defined within the `<shutdown-classes>` and `<shutdown-class>` elements and there is no `failure-is-fatal` attribute. Thus, the configuration for a `TestShutdown` class would be as follows:

```
<shutdown-classes>
        <shutdown-class classname="TestShutdown">
            <execution-order>0</execution-order>
                <init-param>
                    <param-name>oracle.test.shutdown</param-name>
                    <param-value>true</param-value>
                </init-param>
                <init-param>
                    <param-name>shutdown.oracle.year</param-name>
                    <param-value>2002</param-value>
                </init-param>
        </shutdown-class>
 </shutdown-classes>
```

Assuming that the `TestShutdown` class is archived in `"../app1/shutdown.jar"`, add another `<init-library>` element in the `server.xml` file as follows:

```
<init-library path="../app1/shutdown.jar" />
```

Setting Performance Options

Most performance settings are discussed in the *Oracle Application Server 10g Performance Guide.*

You can manage these performance settings yourself from either the OC4J command-line option or by editing the appropriate XML file element.

- Performance Command-Line Options
- Thread Pool Settings
- Statement Caching
- Task Manager Granularity

Performance Command-Line Options

Each -D command-line option, except for the dedicated.rmicontext option, defaults to the recommended setting. However, you can modify these options by providing each -D command-line option as an OC4J option. See the"OC4J Command-Line Options and System Properties" on page A-41 for an example.

- dedicated.rmicontext=true/false. The default value is false. This replaces the deprecated dedicated.connection setting. When two or more clients in the same process retrieve an InitialContext, OC4J returns a cached context. Thus, each client receives the same InitialContext, which is assigned to the process. Server lookup, which results in server load balancing, happens only if the client retrieves its own InitialContext. If you set dedicated.rmicontext=true, then each client receives its own InitialContext instead of a shared context. When each client has its own InitialContext, then the clients can be load balanced.

  This parameter is for the client. You can also set this in the JNDI properties.

- oracle.dms.sensors=[none, normal, heavy, all]. You can set the value for Oracle Application Server built-in performance metrics to the following: None (off), normal (medium amount of metrics), heavy (high number of metrics), or all (all possible metrics). The default is normal.This parameter should be set on the OC4J server. The previous method for turning on these performance metrics, oracle.dms.gate=true/false, is replaced by the oracle.dms.sensors variable. However, if you still use oracle.dms.gate, then setting this variable to false is equivalent to setting oracle.dms.sensors=none.

- DefineColumnType=true/false. The default is false. Set this to true if you are using an Oracle JDBC driver that is prior to 9.2. For these drivers, setting this variable to true avoids a round-trip when executing a select over the Oracle JDBC driver. This parameter should be set on the OC4J server.

  When you change the value of this option and restart OC4J, it is only valid for applications deployed after the change. Any applications deployed before the change are not affected.

  When true, the DefineColumnType extension saves a round trip to the database that would otherwise be necessary to describe the table. When the Oracle JDBC driver performs a query, it first uses a round trip to a database to determine the types that it should use for the columns of the result set. Then, when JDBC receives data from the query, it converts the data, as necessary, as it populates the result set. When you specify column types for a query with the DefineColumnType extension set to true, you avoid the first round trip to the

Oracle database. The server, which is optimized to do so, performs any necessary type conversions.

Thread Pool Settings

You can specify an unbounded, one, or two thread pools for an OC4J process through the `<global-thread-pool>` element in the `server.xml` file. If you do not specify this element, then an infinite number of threads can be created, which is the unbounded option.

There are two types of threads in OC4J:

- short lived threads: A worker thread that is process intensive and uses database resources. These threads are mapped `ApplicationServerThreadPool`.
- long lived threads: A connection thread that is not process intensive. It listens for events or processes socket IOs. These threads are mapped to `ConnectionThreadPool`.

OC4J always maintains a certain amount of worker threads, so that any client connection traffic bursts can be handled.

If you specify a single thread pool, then both short and long lived threads exist in this pool. The risk is that all the available threads in the pool are one type of thread. Then, performance can be poor because of a lack of resources for the other type of thread. However, OC4J always guarantees a certain amount of worker threads, which are normally mapped to short lived threads. If a need for a worker thread arises and no short lived thread is available, the work is handled by a long lived thread.

If you specify two thread pools, then each pool contains one type of thread.

To create a single pool, configure the `min`, `max`, `queue`, and `keepAlive` attributes. To create two pools, configure the `min`, `max`, `queue`, and `keepAlive` attributes for the first pool and the `cx-min`, `cx-max`, `cx-queue`, and `cx-keepAlive` attributes for the second pool. In order to activate two thread pools, you must configure all the attributes for the first thread pool, which includes min, max, queue, and keepAlive. If any of these attributes is not configured, you cannot configure the second pool. Instead, you will receive the following error message:

```
Error initializing server: Invalid Thread Pool parameter: null
```

The `global-thread-pool` element provides the following attributes:

Table 3–2 The Thread Pool Attributes

Thread Pool Attributes	Description
`min`	The minimum number of threads that OC4J can simultaneously execute. By default, a minimum number of threads are preallocated and placed in the thread pool when the container starts. Value is an integer. The default is 20. The minimum value you can set this to is 10.
`max`	The maximum number of threads that OC4J can simultaneously execute. New threads are spawned if the maximum size is not reached and if there are no idle threads. Idle threads are used first before a new thread is spawned. Value is an integer. The default is 40.
`queue`	The maximum number of requests that can be kept in the queue. Value is an integer. The default is 80.
`keepAlive`	The number of milliseconds to keep a thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains alive. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one. Value is a long. The default is 600000 milliseconds.
`cx-min`	The minimum number of connection threads that OC4J can simultaneously execute. Value is an integer. The default is 20. The minimum value you can set this to is 10.
`cx-max`	The maximum number of connection threads that OC4J can simultaneously execute. Value is an integer. The default is 40.
`cx-queue`	The maximum number of connection requests that can be kept in the queue. Value is an integer. The default is 80.
`cx-keepAlive`	The number of milliseconds to keep a connection thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains alive. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one. Value is a long. The default is 600000 milliseconds.
`debug`	If true, print the application server thread pool information at startup. The default is false.

Recommendations:

- The `queue` attributes should be at least twice the size of the maximum number of threads.
- The minimum and maximum number of worker threads should be a multiple of the number of CPUs installed on your machine and fairly small. The more threads you have, the more burden you put on the operating system and the garbage collector. The minimum that you should set it to is 10.
- The `cx-min` and `cx-max` sets the thread pool size for the connection threads; thus, they are relative to the number of the physical connections you have at any point in time. The `cx-queue` handles burst in connection traffic.
- When running benchmarks or in a production environment, once you figure out the right number of threads, set the minimum to the maximum number and the `keepAlive` attribute to negative one.

Example 3–2 Setting Thread Pool

The following example initializes two thread pools for the OC4J process. Each contains at minimum 10 threads and maximum of 100 threads. The number of requests outstanding in each queue can be 200 requests. Also, idle threads are kept alive for 700 seconds. The thread pool information is printed at startup.

```
<application-server ...>
...
  <global-thread-pool min="10" max="100" queue="200"
      keepAlive="700000" cx-min="10" cx-max="100" cx-queue="200"
      cx-keepAlive="700000" debug="true"/>
...
</application-server>
```

Statement Caching

You can cache database statements, which prevents the overhead of repeated cursor creation and repeated statement parsing and creation. In the `DataSource` configuration, you enable JDBC statement caching, which caches executable statements that are used repeatedly. A JDBC statement cache is associated with a particular physical connection. See *Oracle9i JDBC Developer's Guide and Reference* for more information on statement caching.

You can dynamically enable and disable statement caching programmatically through the `setStmtCacheSize()` method of your connection object or through the `stmt-cache-size` XML attribute in the `DataSource` configuration. An

integer value is expected with the size of the cache. The cache size you specify is the maximum number of statements in the cache. The user determines how many distinct statements the application issues to the database. Then, the user sets the size of the cache to this number.

If you do not specify this attribute or set it to zero, this cache is disabled.

Example 3–3 Statement Caching

The following XML sets the statement cache size to 200 statements.

```
<data-source>
 ...
 stmt-cache-size="200"
</data-source>
```

Task Manager Granularity

The task manager is a background process that performs cleanup. However, the task manager can be expensive. You can manage when the task manager performs its duties through the `taskmanager-granularity` attribute in `server.xml`. This attribute sets how often the task manager is kicked off for cleanup. Value is in milliseconds. Default is 1000 milliseconds.

```
<application-server ...  taskmanager-granularity="60000" ...>
```

Enabling OC4J Logging

OC4J logs messages both to standard error, standard out, and several log files for OC4J services and deployed applications.

- Viewing OC4J System and Application Log Messages: This section describes the separate log files for OC4J sub-systems and deployed applications. You can manage how large these files can be and where they are located.
- Redirecting Standard Out and Standard Error: This section describes how to forward standard out and standard error messages to a log file.

Note: Also, OC4J supports Jakarta `log4j`. See the "Open Source Frameworks and Utilities" appendix in the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*.

Viewing OC4J System and Application Log Messages

Each OC4J process included in the Oracle Application Server environment has a set of log files, as shown in Table 3–3. If there are multiple processes running for an OC4J instance, there is a multiple set of log files.

Table 3–3 List of Log Files Generated for OC4J

Default Log File Name	Description	Scope	Configuration File
`application.log`	All events, errors, and exceptions for a deployed application.	One log file for each application deployed.	`orion-application.xml`
`global-application.log`	All common events, errors, and exceptions related to applications.	All applications, including the default application.	`application.xml`
`jms.log`	All JMS events and errors.	JMS sub-system	`jms.xml`
`rmi.log`	All RMI events and errors.	RMI sub-system	`rmi.xml`
`server.log`	All events not associated with a particular sub-system or an application. This logs history of server startup, shutdown internal server errors.	server-wide	`server.xml`
`web-access.log`	Logs all accesses to the Web site.	Each Web site	`default-web-site.xml`

There are two types of log files:

- Oracle Diagnostic Logging (ODL) Log Files: The messages logged in these files use an XML format that is read by the Oracle Enterprise Manager GUI. We recommend that you use this format for your logging, even though it is not the default, when you are using OC4J within Oracle Application Server.
- Text Log Files: The messages logged in these files are not in XML and are simply for reading within any editor. This is the default. Normally, those who use OC4J standalone would benefit from viewing their log messages in a text format.

Oracle Diagnostic Logging (ODL) Log Files

The ODL log entries are each written out in XML format in its respective log file. Each XML message can be read through the Oracle Enterprise Manager GUI or through your own XML reader. The advantages for ODL logging is that the log files and the directory have a maximum limit. When the limit is reached, the log files are overwritten.

When you enable ODL logging, each new message goes into the current log file, named `log.xml`. When the log file is full—that is, the log file size maximum is reached—then it is copied to an archival log file, named `log`*N*`.xml`, where N is a number starting at one. When the last log file is full, the following occurs:

1. The least recent log file is erased to provide space in the directory.
2. The `log.xml` file is written to the latest `log`*N*`.xml` file, where N increments by one over the most recent log file.

Thus, your log files are constantly rolling over and do not encroach on your disk space.

Within each XML file listed in Table 3–3, you enable ODL logging by uncommenting the ODL configuration line, as follows:

- Uncommment the `<odl>` element within the `<log>` element in all XML files listed in Table 3–3, except for the `default-web-site.xml` file.
- Uncomment the `<odl-access-log>` element in the `default-web-site.xml` file.

The attributes that you can configure are:

- `path`: Path and folder name of the log folder for this area. You can use an absolute path or a path relative to where the configuration XML file exists, which is normally in the `j2ee/home/config` directory. This denotes where the log files will reside for the feature that the XML configuration file is concerned with. For example, modifying this element in the `server.xml` file denotes where the server log files are written.
- `max-file-size`: The maximum size in KB of each individual log file.
- `max-directory-size`: The maximum size of the directory in KB.

New files are created within the directory, until the maximum directory size is reached. Each log file is equal to or less than the maximum specified in the attributes.

Thus, to specify log files of 1000 KB and a maximum of 10,000 KB for the directory in the `ORACLE_HOME/j2ee/log/server` directory in the `server.xml` file, configure the following:

```
<log>
<odl path="../log/server/" max-file-size="1000" max-directory-size="10000" />
</log>
```

When OC4J is executing, all log messages that are server oriented are logged in the *`ORACLE_HOME`*`/j2ee/log/server` directory.

The XML message that is logged is of the following format:

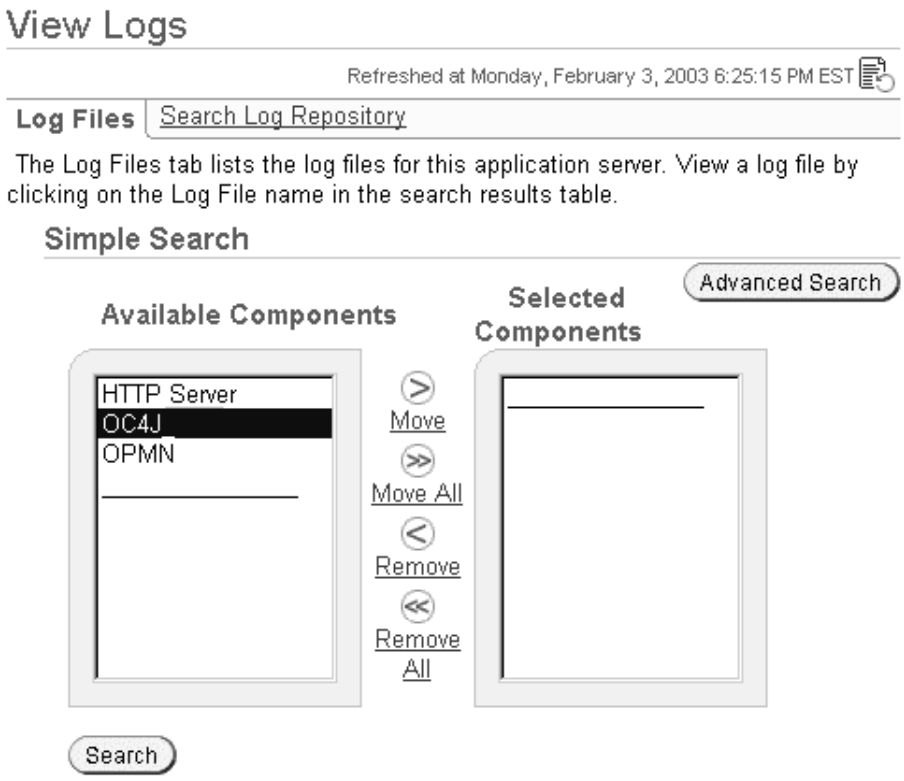
```
<MESSAGE>
<HEADER>
<TSTZ_ORIGINATING>2002-11-12T15:02:07.051-08:00</TSTZ_ORIGINATING>
<COMPONENT_ID>oc4j</COMPONENT_ID>
<MSG_TYPE TYPE="ERROR"></MSG_TYPE>
<MSG_LEVEL>1</MSG_LEVEL>
<HOST_ID>myhost</HOST_ID>
<HOST_NWADDR>001.11.22.33</HOST_NWADDR>
<PROCESS_ID>null-Thread[Orion Launcher,5,main]</PROCESS_ID>
<USER_ID>dpda</USER_ID>
</HEADER>
<PAYLOAD>
<MSG_TEXT>java.lang.NullPointerException at
com.evermind.server.ApplicationServer.setConfig(ApplicationServer.java:1070)
at com.evermind.server.ApplicationServerLauncher.run
(ApplicationServerLauncher.java:93) at java.lang.Thread.run(Unknown Source)
</MSG_TEXT>
</PAYLOAD>
</MESSAGE/>
```

You can have both the ODL and text logging turned on. To save on disk space, you should turn off one of these options. If you decide to enable ODL logging, turn off the text logging functionality by commenting out the `<file>` subelement of the `<log>` element for all XML files except the `default-web-site.xml` file. For the `default-web-site.xml` file, turn off the text logging by commenting out the `<access-log>` element.

You can view ODL log files by performing the following:

1. At the bottom of the Oracle Application Server Instance Home page, select **Logs**.
2. Select the OC4J Instances for which you want to view the log files.
   a. Select the OC4J Instances in the Available Components column.
   b. Select **Move** to transfer your select to the Selected Components column.
   c. Select Search to view the log files for these OC4J Instances.

Figure 3–19 Viewing Logs



See Also: "Managing Log Files" chapter in the *Oracle Application Server 10g Administrator's Guide*

Text Log Files

Full text logging is still available in OC4J. Primarily, you should use text logging within OC4J standalone. It is easier to read within any editor, as it is not in XML format.

The text logging facility separates messages out in alignment with the XML files. However, instead of writing to multiple log files of the same size, all messages for that component are written into a single file. The text logging does not have any imposed limits or log rollover. Instead, the log files will continue to grow, unless you stop OC4J, remove the file, and restart OC4J to start the log files over. You can

overrun your disk space if you do not monitor your log files. This is only feasible in a standalone, development environment.

Text messaging is the default and is configured in the same XML files as listed for ODL logging in Table 3–3. Text messaging is enabled in the `<file>` subelement the `<log>` element of the XML files, except the `default-web-site.xml` file. For the `default-web-site.xml` file, the text messaging is enabled with the `<access-log>` element. To turn off text messaging, eliminate or comment out the `<file>` or `<access-log>` element. If you do not remove this line and enable ODL logging, you will have both logging facilities turned on. The location and filename for text messaging does have defaults, as shown in Table 3–4, but you can specify the location and filename within the path attribute of the `<log>` or `<access-log>` elements.

The following table shows the default location directory for text messaging log files of an OC4J process.

Table 3–4 OC4J Log File Locations

Log File	Default Location
application.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/application-deployments/<application-name>/<OC4J IslandName>
global-application.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
jms.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
rmi.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
server.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
web-access.log	The location is configurable from *-web-site.xml with the <access-log> element, as follows: <access-log path="../log/http-web-access.log" />
OPMN log file, named <OC4J_Instance_Name>~<Island_Name>~<Process#>	*$ORACLE_HOME*/opmn/logs/

The location of all of the above log files can be specified, except the `web-access.log` file, using the `<log>` element in the respective configuration files. You can specify either absolute paths or paths relative to the `j2ee/home/config` directory. For example, specify the server log file in the `server.xml` configuration file, as follows:

```
<log>
<file path="../log/my-server.log" />
</log>
```

You can also specify an absolute path for the location of the log file, as follows:

```
<log>
<file path="d:\log-files\my-server.log" />
</log>
```

Redirecting Standard Out and Standard Error

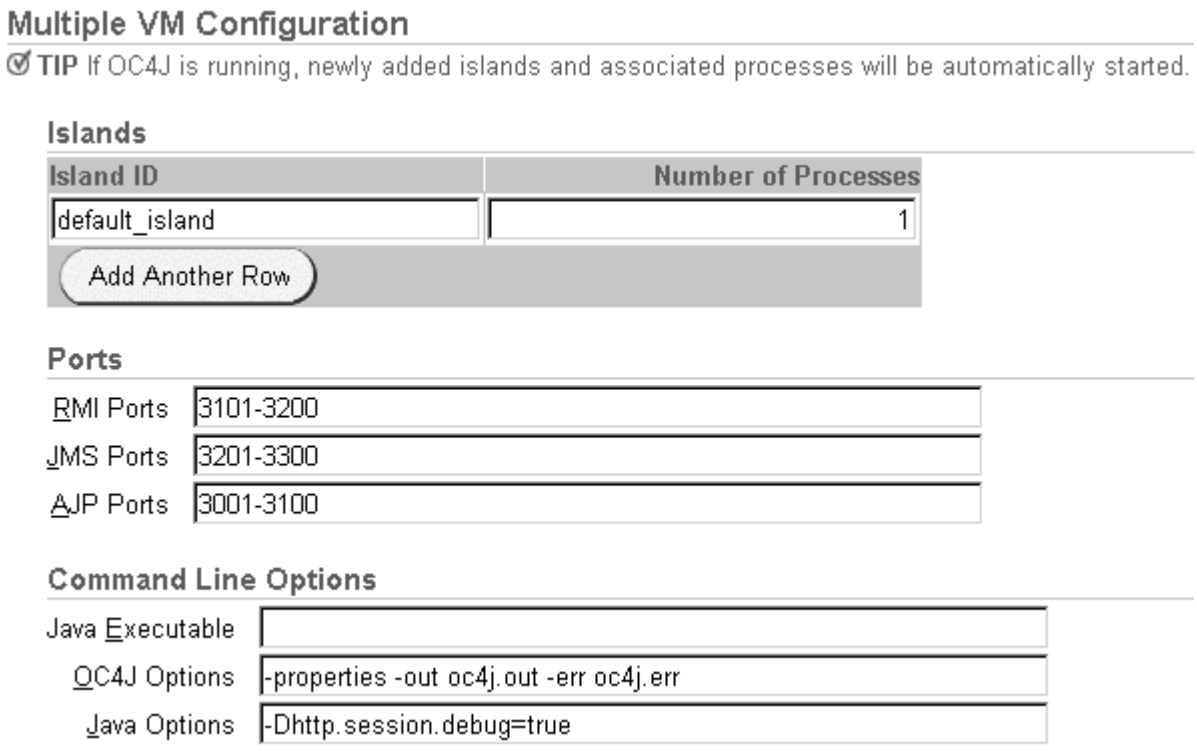
In an Oracle Application Server environment, the standard output and standard errors for OC4J are routed to the OPMN log for the OC4J instance. For example, if you have an OC4J instance called `OC4J_Demos`, and it has an island called `default_island`, and there is only one process in the island, the file that will contain the STDOUT and STDERR streams is *`$ORACLE_HOME`*`/opmn/logs/OC4J_Demos.default_island.1`

Figure 3–20 shows how you specify the `-out` and `-err` parameters in the OC4J command-line options for your OC4J Instance. When you specify these parameters without specifying a specific directory, the log files are created in the *`$J2EE_HOME`*`/<OC4JInstanceName>_<IslandName>_<Process#>` file. For example, if you have an OC4J instance called `OC4J_Demos` and it has an island called `default_island`, then the log file will be created in the *`$J2EE_HOME`*`/OC4J_Demos_default_island_1`.

> **Note:** In the preceding examples, it is assumed that there is only one OC4J process for the default island. If there are more processes for an island there are separate log files for each OC4J process.

In the Application Server Control, select Server Properties on the Administration page, which will bring you to the screen in Figure 3–20.

Figure 3–20 EM Console to Modify Server Properties for an OC4J Instance



OC4J Debugging

OC4J properties are configuration switches that can be set on the command-line. As shown in Figure 3–20, the properties are prefaced with a `-D` in the Java Options line. OC4J provides several debug properties for generating additional information on the operations performed by the various sub-systems of OC4J. These debug properties can be set for a particular sub-system while starting up OC4J.

Note: Turning on excessive debug options can slow down the execution of your applications and use large amounts of disk space with the contents of the log files.

OC4J is started and managed by OPMN. You have to specify the Java system properties for your OC4J instance using Oracle Enterprise Manager as shown in Figure 3–20. The supplied properties are saved in the OPMN configuration file. OPMN starts OC4J with these supplied properties when you shutdown and restart your OC4J Instance. See "OC4J Command-Line Options and System Properties" on page A-41 for OC4J general properties.

By default, the debug information is written to the OPMN log file for the OC4J Instance, which resides in the *$ORACLE_HOME*/opmn/logs/ directory. For example, you have a OC4J instance named OC4J_DEMOS and there is only one island and the island has only process for this OC4J instance, then the debug information is logged in *$ORACLE_HOME*/opmn/logs/OC4J_Demos.default_island.1. However, if there is -out and -err command-line options specified with OC4J, then the debug information is redirected to the appropriate files.

The following tables provide useful debug options that available with OC4J. These debug options have two states either true or false. By default these are set to false. For a complete list of debug properties, see "OC4J Command-Line Options and System Properties" on page A-41.

Table 3–5 HTTP Debugging Options

HTTP Debugging	Description of Option
http.session.debug	Provides information about HTTP session events
http.request.debug	Provides information about each HTTP request
http.cluster.debug	Provides information about HTTP clustering events
http.error.debug	Prints all HTTP errors
http.method.trace.allow	Default: false. If true, turns on the trace HTTP method.

Table 3–6 JDBC Debugging Options

JDBC Debugging	Description of Option
datasource.verbose	Provides verbose information on creation of data source and connections using Data Sources and connections released to the pool, and so on,
jdbc.debug	Provides very verbose information when JDBC calls are made

Table 3–7 EJB Debugging Options

EJB Debugging	Description of Options
`ejb.cluster.debug`	Turns on EJB clustering debug messages

Table 3–8 RMI Debugging Options

RMI Debugging	Description of Options
`rmi.debug`	Prints RMI debug information
`rmi.verbose`	Provides very verbose information on RMI calls

Table 3–9 OracleAS Web Services Debugging Options

OracleAS Web Services Debugging	Description of Options
`ws.debug`	Turns on Web Services debugging

In addition to the specific sub-system switches, you can also start OC4J with a supplied verbosity level. The verbosity level is an integer between 1 and 10. The higher the verbosity level, the more information that is printed in the console. You specify the verbosity level with the `-verbosity` OC4J option in the Oracle Enterprise Manager in the OC4J command-line options section. The following examples show the output with and without verbosity:

Example 3–4 Error Messages Displayed Without Verbosity

```
D:\oc4j903\j2ee\home>java -jar oc4j.jar
Oracle Application Server Containers for J2EE initialized
```

Example 3–5 Error Messages Displayed With Verbosity Level of 10

```
D:\oc4j903\j2ee\home>java -jar oc4j.jar -verbosity 10
Application default (default) initialized...
Binding EJB work.ejb.WorkHours to work.ejb.WorkHours...
Application work (work) initialized...
Application serv23 (Servlet 2.3 New Features Demo) initialized...
Web-App default:defaultWebApp (0.0.0.0/0.0.0.0:8888) started...
Oracle Application Server Containers for J2EE initialized
```

Servlet Debugging Example

You deployed a Web application to OC4J that is having some problems with servlets. You are losing the client session when you use a pre-configured data source to make database connection. You want to know what OC4J is doing when the servlet is accessing the data source. In order to generate the debug information on HTTP Session and data source usage, you must set two debug options - `http.session.debug` and `datasource.verbose` to true.

Perform the following tasks

1. Logon to Oracle Enterprise Manager console as administrator.
2. Drill down to the OC4J Instance.
3. Select **Server Properties** for the OC4J Instance.
4. Enter Java Options as follows: `-Dhttp.session.debug=true -Ddatasource.verbose=true`
5. Restart the OC4J instance.

After the OC4J Instance is restarted, you re-execute your servlet and see the following type of debug information in the standard output for the OC4J Instance:

```
DataSource logwriter activated... jdbc:oracle:thin:@localhost:1521:DEBU:
Started
jdbc:oracle:thin:@localhost:1521:DEBU: Started
Oracle Application Server Containers for J2EE initialized
Created session with id '4fa5eb1b9a564869a426e8544963754f' at Tue APR 23
16:22:56 PDT 2002, secure-only: false
Created new physical connection: XA XA Orion Pooled
jdbc:oracle:thin:@localhost:1521:DEBU
null: Connection XA XA Orion Pooled jdbc:oracle:thin:@localhost:1521:DEBU
allocated (Pool size: 0)
jdbc:oracle:thin:@localhost:1521:DEBU: Opened connection
Created new physical connection: Pooled
oracle.jdbc.driver.OracleConnection@5f18
Pooled jdbc:oracle:thin:@localhost:1521:DEBU: Connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 allocated (Pool size: 0)
Pooled jdbc:oracle:thin:@localhost:1521:DEBU: Releasing connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 to pool (Pool size: 1)
null: Releasing connection XA XA Orion Pooled
jdbc:oracle:thin:@localhost:1521:DEBU to pool (Pool size: 1)
Orion Pooled jdbc:oracle:thin:@localhost:1521:DEBU: Cache timeout, closing
connection (Pool size: 0)
com.evermind.sql.OrionCMTDataSource/default/jdbc/OracleDS: Cache timeout,
```

```
closing connection (Pool size: 0)
```

Remote Debugging Using Oracle JDeveloper

You can remotely debug your applications deployed in OC4J using any Java debugging facility that supports the JPDA (Java Platform Debugging Architecture). With OC4J embedded directly inside of the Oracle JDeveloper IDE, debugging both locally deployed and remote J2EE applications is simple. For more details, please see the Oracle JDeveloper documentation or read the "Remote Debugging How-To" document that is posted on OTN.

4

Data Sources Primer

This chapter describes how to use the pre-installed default data source in your OC4J application. A data source, which is the instantiation of an object that implements the `javax.sql.DataSource` interface, enables you to retrieve a connection to a database server.

This chapter covers the following topics:

- Introduction
- Definition of Data Sources
- Retrieving a Connection From a Data Source

For more information on data sources, see the Data Source chapter in the *Oracle Application Server Containers for J2EE Services Guide*.

Introduction

A *data source* is a Java object that has the properties and methods specified by the `javax.sql.DataSource` interface. Data sources offer a portable, vendor-independent method for creating JDBC connections. Data sources are factories that return JDBC connections to a database. J2EE applications use JNDI to look up `DataSource` objects. Each JDBC 2.0 driver provides its own implementation of a `DataSource` object, which can be bound into the JNDI namespace. Once bound, you can retrieve this data source object through a JNDI lookup.

Because they are vendor-independent, we recommend that J2EE applications retrieve connections to data servers using data sources.

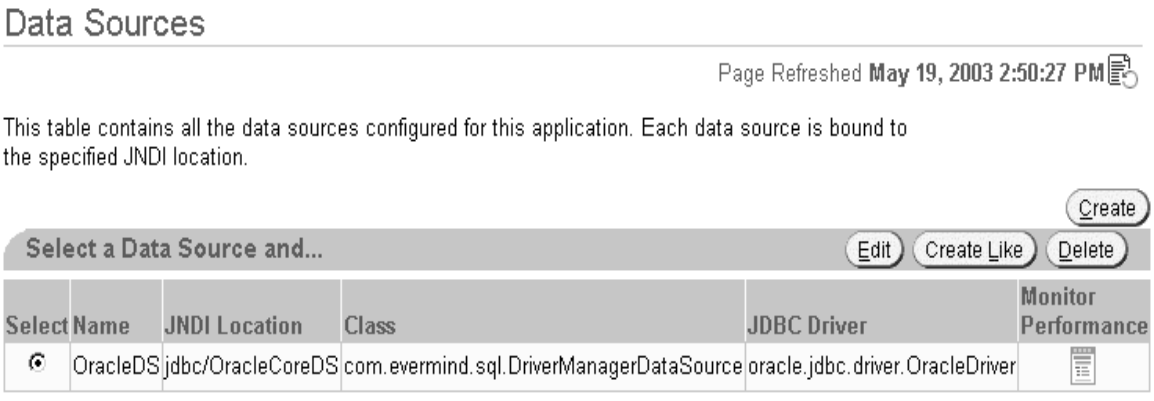
Definition of Data Sources

OC4J data sources are stored in an XML file known as `data-sources.xml`.

Defining Data Sources

The `data-sources.xml` file is pre-installed with a default data source named `OracleDS`. For most uses, this default is all you will need. However, you can also add your own customized data source definitions. Oracle Enterprise Manager displays all data sources in the global Data Sources page. From the OC4J Home Page, select the Administration page and choose Data Source from the Application Defaults column. The following graphic shows the Data Source page.

Figure 4–1 Data Source Page



These data sources are able to be used by all applications deployed in this OC4J instance. To create data sources that are local to a particular application, drill down to the application page and then choose Data Source in the Administration section.

The `OracleDS` default data source is an emulated data source. That is, it is a wrapper around Oracle data source objects. You can use this data source for applications that access and update only a single data server. If you need to update more than one database and want these updates to be included in a JTA transaction, you must use a non-emulated data source. See the Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide* for more information on non-emulated data sources.

The default emulated data source is extremely fast and efficient, because it does not enable two-phase commit operations. This would be necessary if you were to manage more than a single database.

The following shows the XML configuration for the default data source definition that you can use for most applications:

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="hr"
  password="hr"
  url="jdbc:oracle:thin:@myhost:1521:ORCL"
  inactivity-timeout="30"
/>
```

- The `class` attribute defines the type of data source you want to use.
- The `location`, `xa-location`, and `ejb-location` attributes are JNDI names that this data source is bound to within the JNDI namespace. While you must specify all three, we recommend that you use only the `ejb-location` JNDI name in the JNDI lookup for retrieving this data source.
- The `connection-driver` attribute defines the type of connection you expect to be returned to you from the data source.
- The URL, username, and password identify the database, its username, and password.

Note: Instead of providing the password in the clear, you can use password indirection. For details, see the *Oracle Application Server Containers for J2EE Services Guide*.

These fields can be modified in either the global Data Sources page or in the global `data-sources.xml` modification page. To navigate to the `data-sources.xml` modification page, select the default application from the OC4J Home page. Scroll down to the Administration section and choose Advanced Properties.

The Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide* fully describes all elements for configuring any type of data source.

Configuring A New Data Source

You can configure global or local data sources. A global data source is available to all deployed applications in this OC4J instance. A local data source is configured within the deployed application and can only be used by that application.

See the *Oracle Application Server Containers for J2EE Services Guide* for a full explanation of how to configure a data source and the elements within the `data-sources.xml` file.

To configure global data sources, select one of the following:

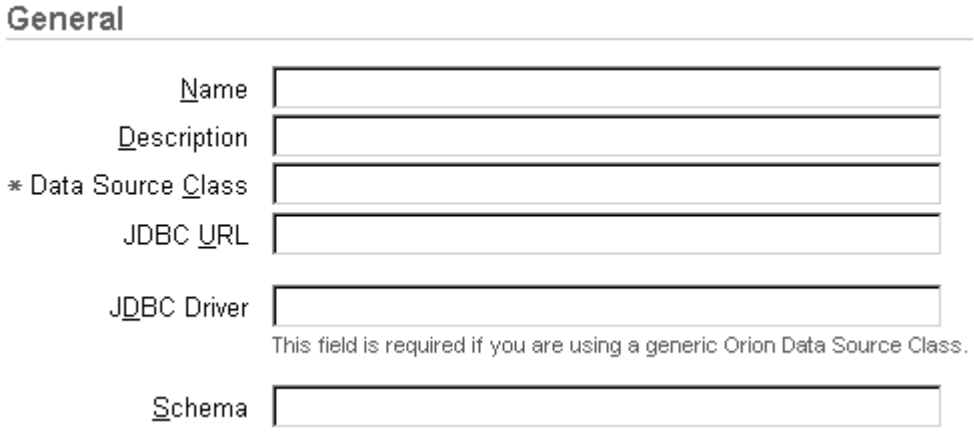
- Data Sources under the Application Defaults column on the Administration page—This page allows you to add data source definitions one field at a time. See "Data Source Field Page" on page 4-5 for a description of this page.
- Advanced Properties in the default application off the Applications page—On the Applications page, select the default application. Scroll down to the Administration section and select Advanced Properties. Select `data-sources.xml` on this page. This allows you to add data sources using the XML definitions. This is useful if you have been provided the XML. You can just copy in the data source XML.

To configure local data sources, you perform the same selection off of the application page. You must drill down to the particular application that this data source will be local to. On the application page, choose Data Source under the Resources column. It displays the same data source field page that is discussed in "Data Source Field Page" on page 4-5.

Data Source Field Page When you choose Data Sources under the Application Defaults column, you can enter all configuration details about the data source into fields provided. This page is divided up into five sections.

Figure 4–2 shows the General section.

Figure 4–2 General Section of Data Source Definition

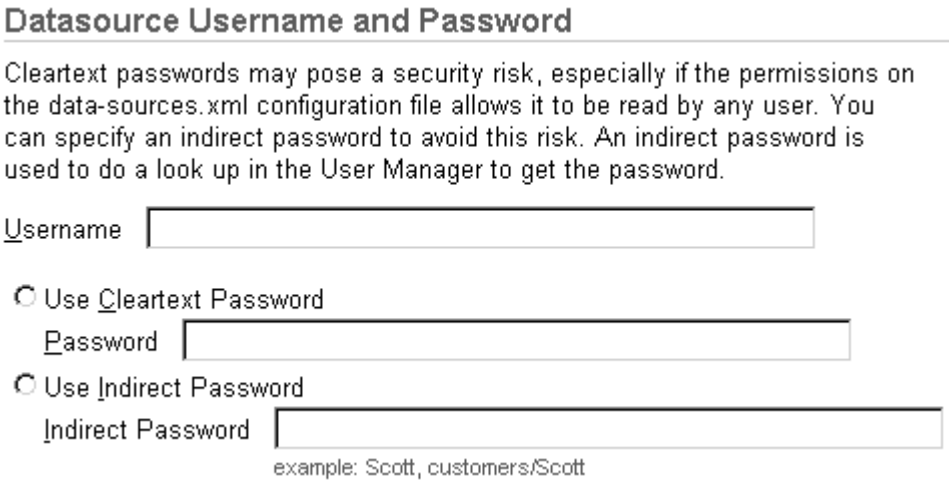


The General section enables you to define the following aspects about a data source:

- Name—A user-defined name to identify the data source.
- Description—A user-defined description of the data source.
- Data Source Class—This is the class, such as `com.evermind.sql.ConnectionDataSource`, that the data source is instantiated as.
- JDBC URL—The URL to the database represented by this data source. For example, if using an Oracle Thin driver, the URL could be the following: `jdbc:oracle:thin:@my-lap:1521:SID`.
- JDBC Driver—The JDBC driver to use. One example of a JDBC driver is `oracle.jdbc.driver.OracleDriver`.
- Schema—This is an optional parameter. Input the file name that contains the Java to database mappings for a particular database.

Figure 4–3 shows the username and password.

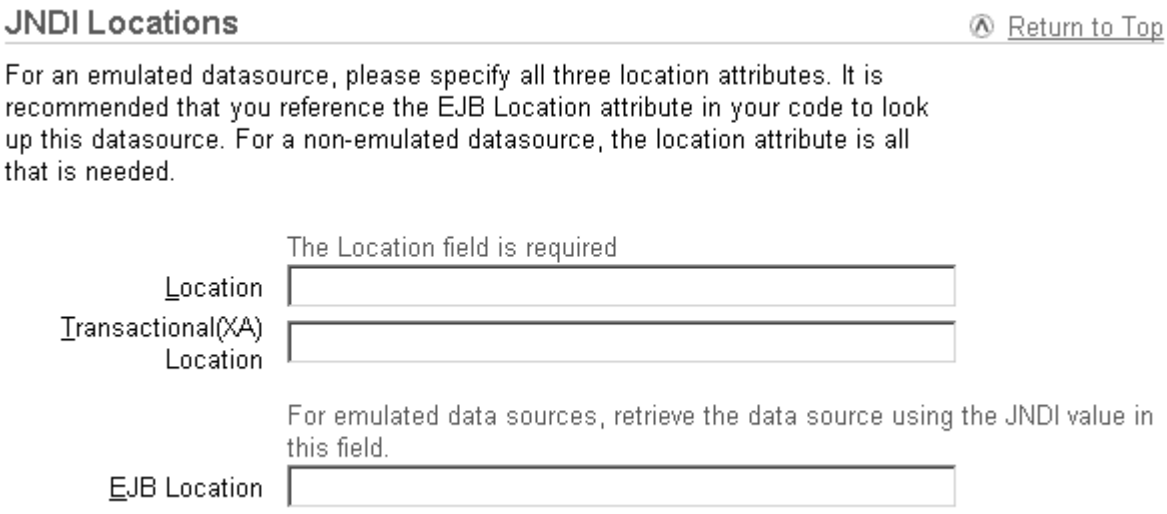
Figure 4–3 Username and Password



Username / Password—The username and password used to authenticate to the database that this data source represents. The password can either be entered as clear text, or you can provide a username for an indirect password. For details, see the *Oracle Application Server Containers for J2EE Services Guide.*

Figure 4–4 shows the JNDI Locations section.

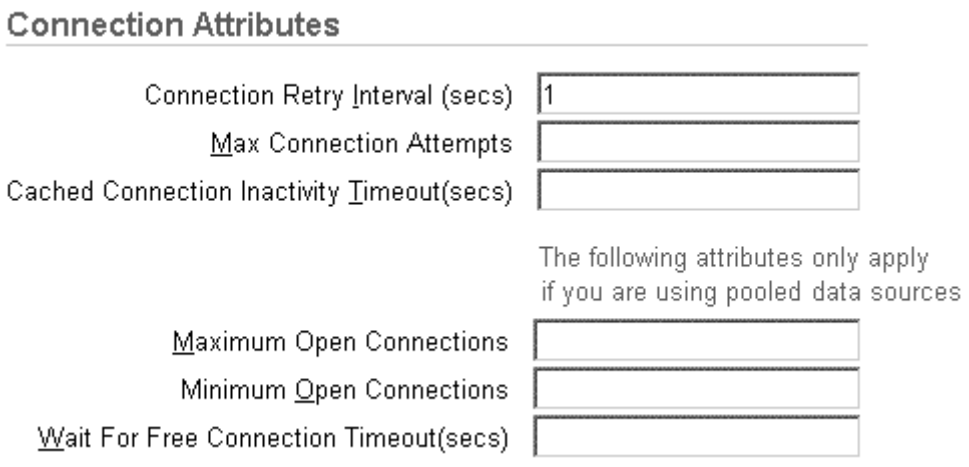
Figure 4–4 JNDI Locations



The JNDI Locations section enables you to define the JNDI location string that the data source is bound with. This JNDI location is used within JNDI lookup for retrieving this data source. For emulated, you must provide all locations, even though only the EJB Location is used. That is, you should only refer to the EJB Location in your application.

Figure 4–5 shows the Connection Attributes section.

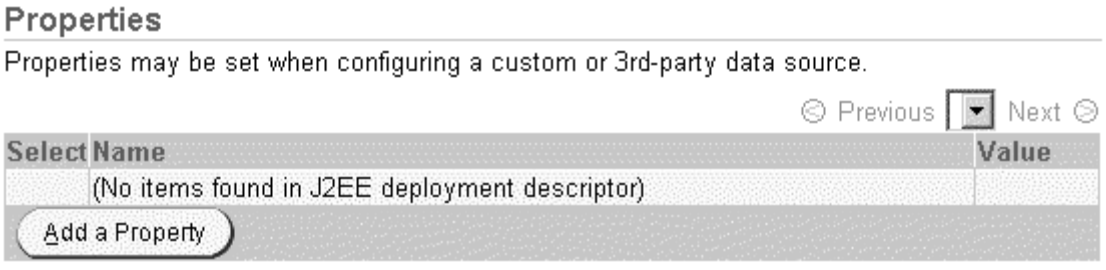
Figure 4–5 Connection Attributes



This section enables you to modify connection tuning parameters, including the retry interval, pooling parameters, timeout parameters, and maximum attempt parameter.

Figure 4–6 shows the Properties section for the data source.

Figure 4–6 Properties



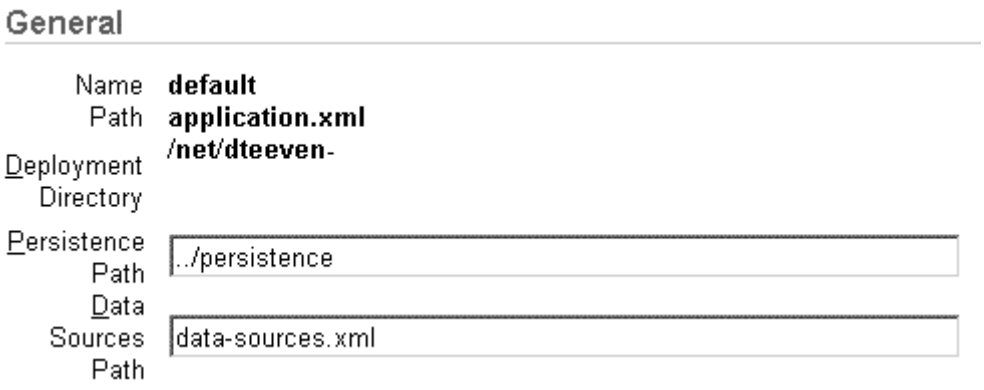
If your data source is a third party data source, you may need to set certain properties. These properties would be defined in the third-party documentation. In addition, properties must be set for JTA transactions for the two-phase commit coordinator.

Defining the Location of the DataSource XML Configuration File

The elements you add or modify are stored by Oracle Enterprise Manager in an XML file. This file defaults to the name of `data-sources.xml` and is located in `/j2ee/home/config`. If you want to change the name or the location of this file, you can do this in the General Properties page off of the default application screen.

On the Applications page, scroll down to Default Application. Choose default. This brings you to the default application screen. Scroll down to the Administration section and choose General from the Properties column. Within the General Properties screen, shown below, you can modify the name and location of the data sources XML configuration file. Any location that you configure in the data sources path field must be relative to the `/j2ee/home/config` directory.

Figure 4–7 Default Application Properties



When applied, the data sources XML filename and path are stored in the global `application.xml` file. In the application.xml file, the `<data-sources>` element contains both the name and path of the data sources XML file.

The following shows the default configuration:

```
<data-sources
  path = "data-sources.xml"
/>
```

The `path` attribute of the `<data-sources>` tag contains both path and name of the `data-sources.xml` file. The path can be fixed, or it can be relative to where the `application.xml` is located.

Retrieving a Connection From a Data Source

One way to modify data in your database is to retrieve a JDBC connection and use JDBC or SQLJ statements. We recommend that you use data source objects in your JDBC operations.

Do the following to modify data within your database:

1. Retrieve the `DataSource` object through a JNDI lookup on the data source definition.

   The lookup is performed on the logical name of the default data source, which is an emulated data source that is defined in the `ejb-location` element.

   You must always cast or narrow the object that JNDI returns to the `DataSource`, because the JNDI `lookup()` method returns a Java `object`.

2. Create a connection to the database represented by the `DataSource` object.

Once you have the connection, you can construct and execute JDBC statements against this database specified by the data source.

The following code represents the preceding steps:

```
Context ic = new InitialContext();
DataSource ds = (DataSource) ic.lookup("jdbc/OracleDS");
Connection conn = ds.getConnection();
```

Use the following methods of the `DataSource` object in your application code to retrieve the connection to your database:

- `getConnection();`

  The username and password are those defined in the data source definition.

- `getConnection(String username, String password);`

  This username and password overrides the username and password defined in the data source definition.

You can cast the connection object returned on the `getConnection` method to `oracle.jdbc.OracleConnection` and use all the Oracle extensions. This is shown below:

```
oracle.jdbc.OracleConnection conn =
                (oracle.jdbc.OracleConnection) ds.getConnection();
```

Once retrieved, you can execute SQL statements against the database either through SQLJ or JDBC.

For more information, see the Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide*.

5

Servlet Primer

In Oracle Application Server 10*g*, OC4J includes a servlet container that is fully compliant with the servlet 2.3 specification. This chapter covers the basics of running servlet applications in the OC4J environment. There is also a brief servlet review, although it is assumed that you are at least somewhat familiar with servlet technology.

There are a few assumptions before you try running the primers. See "Introduction to OC4J" on page 1-2.

This chapter includes the following sections:

- A Brief Overview of Servlet Technology
- Running a Simple Servlet
- Running a Data-Access Servlet
- Creating and Deploying the Servlet Primer Samples WAR File

For detailed information about the Oracle servlet implementation, see the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*.

A Brief Overview of Servlet Technology

The following sections provide a quick servlet overview:

- What Is a Servlet?
- Servlet Portability
- The Servlet Container
- Request and Response Objects
- Learning More About Servlets

What Is a Servlet?

In recent years, servlet technology has emerged as a powerful way to extend Web server functionality through dynamic Web pages. A servlet is a Java program that runs in a Web server (as opposed to an applet, which is a Java program that runs in a client browser). Typically, the servlet takes an HTTP request from a browser, generates dynamic content (such as by querying a database), and provides an HTTP response back to the browser. Alternatively, it can be accessed directly from another application component, or send its output to another component. Most servlets generate HTML text, but a servlet might instead generate XML to encapsulate data.

More specifically, a servlet runs in a J2EE application server, such as OC4J. Servlet is one of the main application component types of a J2EE application, along with JavaServer Pages (JSP) and Enterprise JavaBeans (EJB), which are also server-side J2EE component types. These are used in conjunction with client-side components such as applets (part of the Java 2 Standard Edition specification) and application client programs. An application might consist of any number of any of these components.

Using Java servlets allows you to use the standard servlet API for programming convenience, and enables you to employ any of the numerous standard Java and J2EE features and services, including JDBC to access a database, RMI to call remote objects, or JMS to perform asynchronous messaging.

Servlets outperform previous means of generating dynamic HTML. Once a servlet is loaded into memory, it is able to run as a single lightweight thread. The ability to run as a continuous process has led to servlets largely replacing older technologies such as server-side includes and CGI as a means of running code in the server.

Servlet Portability

Because servlets are written in the Java programming language, they are supported on any platform that has a Java virtual machine and has a Web server and J2EE containers. You can use servlets on different platforms without recompiling, and you can package servlets together with associated files such as graphics, sounds, and other data to make a complete Web application. This greatly simplifies application development.

A servlet-based application that was developed to run on any J2EE-compliant application server can be ported to OC4J with little effort.

The Servlet Container

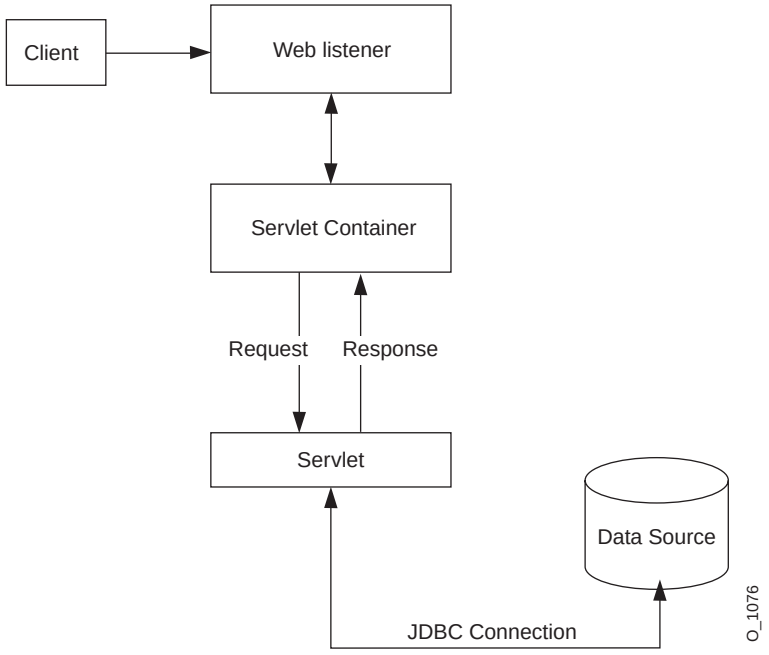
Unlike a Java client program, a servlet has no static `main()` method. Therefore, a servlet must execute under the control of an external container.

Servlet containers, sometimes referred to as *servlet engines*, execute and manage servlets. It is the servlet container that calls servlet methods and provides services that the servlet needs when executing. A servlet container is usually written in Java and is either part of a Web server (if the Web server is also written in Java) or otherwise associated with and used by a Web server. OC4J includes a fully standards-compliant servlet container.

The servlet container provides the servlet easy access to properties of the HTTP request, such as its headers and parameters. When a servlet is called or invoked, the Web server passes the HTTP request to the servlet container. The container, in turn, passes the request to the servlet.

Figure 5–1 illustrates the communication path between a client (such as a Web browser), the Web listener in the Web server, the servlet container, a servlet, and a back-end database.

Figure 5–1 Servlet and the Servlet Container



Request and Response Objects

In Java, an HTTP request is represented by an instance of a class that implements the standard `javax.servlet.http.HttpServletRequest` interface. Similarly, an instance of a class that implements the `javax.servlet.http.HttpServletResponse` interface is used for an HTTP response. These interfaces specify methods to be used in processing requests and responses.

A servlet extends one of two standard servlet base classes: `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`. Key `HttpServlet` methods such as `doGet()`, to process an HTTP `GET` request, and `doPost()`, to process an HTTP `POST` request, take an `HttpServletRequest` instance and an `HttpServletResponse` instance as input parameters. The servlet container passes these objects to the servlet and receives the response back from the servlet to pass on to the client or to another server object such as an EJB.

The servlet overrides the access methods implemented in `GenericServlet` and `HttpServlet` classes, as appropriate, in order to process the request and return the

response as desired. For example, most servlets override the `doGet()` and `doPost()` methods (or both) of `HttpServlet`.

Learning More About Servlets

For a first step in learning more about servlets, see the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*. This guide tells you what you need to know to develop servlets and Web applications in the OC4J environment.

For complete documentation of the J2EE APIs, including servlets, visit the Sun Microsystems Web site at:

`http://java.sun.com/j2ee/docs.html`

You can also find a great deal of tutorial information there about servlets as well as other aspects of J2EE application development.

Running a Simple Servlet

A good way to learn about servlets and how to code them is to view a basic servlet example. This section shows you how to create and run a simple "Hello World" servlet.

Create the Hello World Servlet

Here is the Hello World code, showing the basic servlet framework. This servlet just prints "`Hi There!`" back to the client browser. The numbered comments along the right side correspond to the code notes below.

Save this servlet in a file called `HelloWorldServlet.java` and compile it.

> **Note:** Before compiling the servlet, be sure that `servlet.jar`, supplied with OC4J, is in your classpath. This contains the Sun Microsystems `javax.servlet` and `javax.servlet.http` packages.

```
import java.io.*;                                              // 1
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {          // 2
```

```
  public void doGet (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {                          // 3
    resp.setContentType("text/html");                               // 4

    ServletOutputStream out = resp.getOutputStream();               // 5
    out.println("<html>");                                          // 6
    out.println("<head><title>Hello World</title></head>");
    out.println("<body>");
    out.println("<h1>Hi There!</h1>");
    out.println("</body></html>");
  }
}
```

Code Notes

1. You must import at least `java.io.*`, `javax.servlet.*`, and `javax.servlet.http.*` for any servlet you write. Additional packages are needed for SQL operations or to support Oracle JDBC drivers.
2. The servlet extends the `HttpServlet` class, which has base implementations of the methods that a servlet uses in processing HTTP requests and responses.
3. The `doGet()` method, which services HTTP `GET` requests, overrides the base implementation in `HttpServlet`. Like almost all `HttpServlet` methods, `doGet()` takes a request object and a response object as parameters. In this example, no methods are called on the request object (`req`), because this example requires no input data (that is, request data).
4. The servlet calls the `setContentType()` method of the response object to set the response content MIME type in the header. Here it is `text/html`.
5. The `getOutputStream()` method of the response object (`resp`) is called to get an output stream to use in sending the output from the server back to the client. Alternatively, you could call the `getWriter()` method to get a `java.io.PrintWriter` object.
6. The remainder of the servlet consists of output statements with HTML code to write a simple Web page to display "Hi There!" in a Heading 1 (`<h1>`) format. The Web browser will display this output when it receives the response object from the server.

Deploy the Hello World Servlet

Archive `HelloWorldServlet.class` into the WAR file for the servlet primer samples, and deploy the WAR file using the Oracle Enterprise Manager deployment

wizard. This is all described in "Creating and Deploying the Servlet Primer Samples WAR File" on page 5-14.

Run the Hello World Servlet

Assuming you specify a context path of `/hello`, as described in "Deploy the WAR File" on page 5-15, you can run the Hello World servlet with a URL such as the following:

```
http://host:port/hello/servlet/HelloWorldServlet
```

The `/servlet` part of the URL employs an OC4J feature that starts up a servlet, such as `HelloWorldServlet` in this case, according to its class name. The `servlet-webdir` attribute in the `<orion-web-app>` element of the `global-web-application.xml` file or `orion-web.xml` file defines this special URL component. Anything following it in the URL is assumed to be a servlet class name, including applicable package information, within the appropriate servlet context. By default in OC4J, the setting for this URL component is `"/servlet"`.

Important: Invoking a servlet in this way is recommended only for development and testing scenarios. Allowing the invocation of servlets by class name presents a significant security risk; OC4J should not be configured to operate in this mode in a production environment. See the *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for information.

Automatic Compilation

For easier test development, use the OC4J auto-compile feature. Set `development="true"` in the `<orion-web-app>` element of the `global-web-application.xml` configuration file, as follows:

```
<orion-web-app ... development="true" ... >
   ...
</orion-web-app>
```

If `development` is set to `"true"`, then each time you change the servlet (the `.java` or `.class` file) and save it in a particular directory, or change the `web.xml` file, the OC4J server automatically redeploys (essentially, restarts) the servlet or Web application. A modified `.java` file is also automatically recompiled upon first access.

The directory is determined by the setting of the `source-directory` attribute of `<orion-web-app>`. The default is `"WEB-INF/src"` if it exists, otherwise `"WEB-INF/classes"`.

Running a Data-Access Servlet

The `HelloWorldServlet` example shows a minimal servlet with only static output. The power of servlets, however, comes from the ability to retrieve data from a database, generate dynamic content based on the data, and send that content to the client. (Of course, a servlet can also update a database, based upon information passed to it in the HTTP request.)

In this next example, a servlet gets some information from the client (the Web browser), uses this information in constructing a database query, and reports the query results back to the client.

Although there are many ways that a servlet can get information from its client, this example uses a very common method: reading a query string from the HTTP request.

Note: This example works only if the HR schema has been installed in the Oracle database. This schema is part of the sample Common Schemas set available with the Oracle Database.

Create the HTML Form

First, create an HTML page that acts as the front end for the servlet. This page includes an HTML form through which the end user specifies the query parameters.

Enter or copy the following text into a file and name the file `EmpInfo.html`.

```
<html>

<head>
<title>Query the Employees Table</title>
</head>

<body>
<form method=GET ACTION="/hello/servlet/GetEmpInfo">
The query is<br>
SELECT LAST_NAME, EMPLOYEE_ID FROM EMPLOYEES WHERE LAST NAME LIKE ?.<p>

Enter the WHERE clause ? parameter (use % for wildcards).<br>
```

```
Example: 'S%':<br>
<input type=text name="queryVal">
<p>
<input type=submit>
</form>

</body>
</html>
```

Create the GetEmpInfo Servlet

The servlet called by the preceding HTML page constructs a `SELECT` statement (query), with the end user being prompted for the `WHERE` clause to complete the `SELECT` statement. For database access, this example uses JDBC connection, result set, and statement objects. If you are not familiar with JDBC, see the *Oracle9i JDBC Developer's Guide and Reference*.

This code also assumes default OC4J data source configuration in the `data-sources.xml` file, as in the following example:

```
<data-source
        class="com.evermind.sql.DriverManagerDataSource"
        name="OracleDS"
        location="jdbc/OracleCoreDS"
        xa-location="jdbc/xa/OracleXADS"
        ejb-location="jdbc/OracleDS"
        connection-driver="oracle.jdbc.driver.OracleDriver"
        username="hr"
        password="hr"
        url="jdbc:oracle:thin:@localhost:1521:orcl"
        inactivity-timeout="30"
/>
```

Note: For the URL, change `localhost` to an appropriate host name (such as according to the hosts file on UNIX), as applicable. Change `orcl` to the name of the Oracle database instance, if different.

For introductory information about data sources, see Chapter 4, "Data Sources Primer". For further information, see the *Oracle Application Server Containers for J2EE Services Guide*.

Here is the code for the servlet. Numbered comments along the right side correspond to the code notes below.

Enter or copy the code into a file called `GetEmpInfo.java` and compile it.

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.naming.*;                                          // 1
import javax.sql.*;                                             // 2
import java.sql.*;
import java.io.*;

public class GetEmpInfo extends HttpServlet {

  DataSource ds = null;
  Connection conn = null;

  public void init() throws ServletException {                  // 3
    try {
      InitialContext ic = new InitialContext();                 // 4
      ds = (DataSource) ic.lookup("jdbc/OracleDS");             // 5
      conn = ds.getConnection();                                // 6
    }
    catch (SQLException se) {                                   // 7
      throw new ServletException(se);
    }
    catch (NamingException ne) {                                // 8
      throw new ServletException(ne);
    }
  }

  public void doGet (HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {

    String queryVal = req.getParameter("queryVal");             // 9
    String query =                                              //10
      "select last_name, employee_id from employees " +
      "where last_name like " + queryVal;

    resp.setContentType("text/html");

    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("<head><title>GetEmpInfo</title></head>");
    out.println("<body>");
```

```
    try {
      Statement stmt = conn.createStatement();                    //11
      ResultSet rs = stmt.executeQuery(query);                    //12

        out.println("<table border=1 width=50%>");
        out.println("<tr><th width=75%>Last Name</th><th width=25%>Employee " +
                    "ID</th></tr>");

        int count=0;
        while (rs.next()) {                                       //13
         count++;
         out.println("<tr><td>" + rs.getString(1) + "</td><td>" +rs.getInt(2) +
                     "</td></tr>");

        }
         out.println("</table>");
         out.println("<h3>" + count + " rows retrieved</h3>");

      rs.close();                                                 //14
      stmt.close();
}
    catch (SQLException se) {                                     //15
      se.printStackTrace(out);
    }

    out.println("</body></html>");
  }

  public void destroy() {                                         //16
    try {
      conn.close();
    }
    catch (SQLException se) {                                     //15
      se.printStackTrace();
    }
  }
}
```

Code Notes

1. Import `javax.naming.*` to support the JNDI API.
2. Import JDBC standard interfaces in `java.sql` and extended interfaces in `javax.sql` (for support of data sources and connection pooling).
3. Override the `HttpServlet` `init()` method.

4. Get a JNDI initial context. For more information about using JNDI with OC4J, see the *Oracle Application Server Containers for J2EE Services Guide*.
5. Look up the data source with the JNDI name `jdbc/OracleDS`, which is configured by default in the `data-sources.xml` file.
6. Use the data source to get a connection to the database.
7. Catch any SQL exception from the connection attempt, and throw it as a `ServletException` instance.
8. Catch any JNDI naming exception and throw it as a `ServletException` instance.
9. Get the parameter that was passed in the request from the HTML form. This is the `WHERE` clause for the query.
10. Construct a SQL query using the `WHERE` clause specified by the user.
11. Create a JDBC statement object.
12. Execute the query, with the results going into a JDBC result set object.
13. Loop through the rows of the result set. Use the result set `getString()` and `getInt()` methods to get the particular data values and then output the values to the browser.
14. Close the result set and statement.
15. Catch any SQL exceptions from the query, processing of the result set, or closing of the statement object or connection object (two locations). Print the stack trace.
16. The `destroy()` method closes the database connection.

Deploy GetEmpInfo and the HTML Page

Archive `EmpInfo.html` and `GetEmpInfo.class` into the WAR file for the servlet primer samples, and deploy the WAR file using the Oracle Enterprise Manager deployment wizard. This is all described in "Creating and Deploying the Servlet Primer Samples WAR File" on page 5-14.

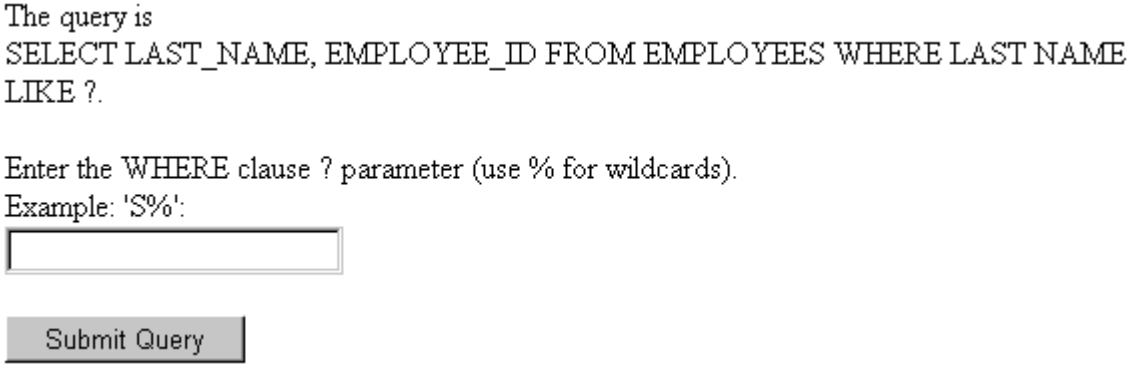
Run GetEmpInfo

Assuming you specify a context path of `/hello`, as described in "Deploy the WAR File" on page 5-15, you can access the front-end HTML page for the `GetEmpInfo` servlet with a URL such as the following:

`http://`*`host`*`:`*`port`*`/hello/EmpInfo.html`

When your browser invokes this page, you should see output like the following:

Figure 5–2 Query Output



Pressing **Submit Query** calls the `GetEmpInfo` servlet. If you first enter `'S%'` (for example) in the form for the `WHERE` clause, you will get the following results:

Figure 5–3 Submit Query Results

Last Name	Employee ID
Sciarra	111
Stiles	138
Seo	139
Sully	157
Smith	159
Sewall	161
Smith	171
Sullivan	182
Sarchand	184

9 rows retrieved.

Creating and Deploying the Servlet Primer Samples WAR File

Two examples have been covered in this chapter: a simple Hello World servlet and a servlet that accesses a database. For simplicity, we suggest that you package and deploy them in a single WAR file. This section shows the required files for the examples and how you can package and deploy them.

Note: Web components, such as JSP pages and servlets, are archived in WAR files. An application consisting of only Web components can be deployed as a WAR file directly, as shown here for simplicity; however, the primary J2EE deployment vehicle is an EAR file. An EAR file can include a WAR file as well as other types of archive files for other types of components, such as EJBs. (For the sample applications here, you also have the option of placing the WAR file within an EAR file, then deploying the EAR file.) See "Creating and Deploying the JSP Primer Samples EAR File" on page 6-14 for an example of an EAR file.

For additional information about building and deploying servlet applications, refer to the *Oracle Application Server Containers for J2EE Servlet Developer's Guide.*

WAR File Structure

Here is the structure of the WAR file for the examples in this chapter:

```
EmpInfo.html
WEB-INF/
   classes/
      HelloWorldServlet.class
      GetEmpInfo.class
```

If you use an IDE for your development, the IDE can usually create the WAR file for you. Otherwise, you can create it manually using the Java JAR utility.

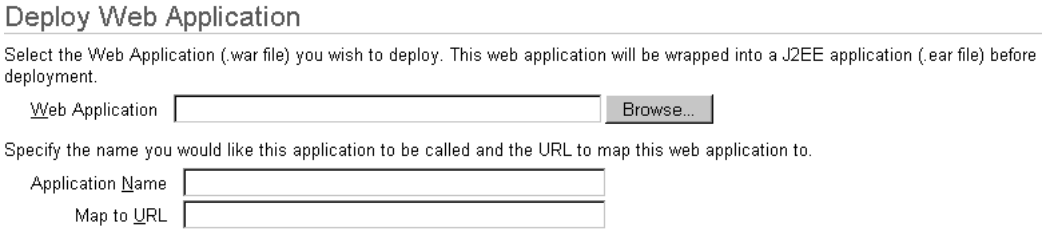
No special `web.xml` entries are required for the examples in this chapter.

Deploy the WAR File

Use Oracle Enterprise Manager, the Application Server Control, to deploy the examples. Select the Applications page. Click the **Deploy WAR file** button and follow the directions for the deployment wizard.

The following figure shows the key portion of the Oracle Enterprise Manager Deploy Web Application Page, which is the page for deploying a WAR file.

Figure 5–4 Deploy the Web Application



Click the **Browse** button to select the WAR file to deploy. Then specify the application name along with a URL mapping for the application. This specified mapping will become the context path portion of the URLs to run the examples in this chapter. In preceding sections showing how to run the examples, we assume that `/hello` is the URL mapping.

6

JSP Primer

In Oracle Application Server 10*g*, OC4J includes a JavaServer Pages (JSP) container that is fully compliant with the JSP 1.2 specification. This chapter covers the basics of running JSP applications in the OC4J environment. There is also a brief JSP review, although it is assumed that you are at least somewhat familiar with JSP technology.

There are a few assumptions before you try running the primers. See "Introduction to OC4J" on page 1-2.

This chapter includes the following sections:

- A Brief Overview of JavaServer Pages Technology
- Running a Simple JSP Page
- Running a JSP Page That Invokes a JavaBean
- Running a JSP Page That Uses Custom Tags
- Creating and Deploying the JSP Primer Samples EAR File

For a complete description of Web application deployment, see "Deploying Applications" on page 2-14.

For detailed information about the Oracle JSP implementation, refer to the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* .

A Brief Overview of JavaServer Pages Technology

Here is a quick JSP overview in the following sections:

- What Is JavaServer Pages Technology?
- JSP Translation and Runtime Flow
- Key JSP Advantages
- Overview of Oracle Value-Added Features for JSP Pages

What Is JavaServer Pages Technology?

JavaServer Pages, a part of the J2EE platform, is a technology that provides a convenient way to generate dynamic content in pages that are output by a Web application. This technology, which is closely coupled with Java servlet technology, allows you to include Java code snippets and calls to external Java components within the HTML code, or other markup code such as XML, of your Web pages. JSP technology works nicely as a front-end for business logic and dynamic functionality encapsulated in JavaBeans and Enterprise JavaBeans (EJB).

Traditional JSP syntax within HTML or other code is designated by being enclosed within `<%...%>` syntax. There are variations on this: `<%=...%>` to designate expressions or `<%!...%>` to designate declarations, for example.

Note: The JSP 1.2 specification introduces an XML-compatible JSP syntax as an alternative to the traditional syntax. This allows you to produce JSP pages that are syntactically valid XML documents. The XML-compatible syntax is described in the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* .

A JSP page is translated into a Java servlet, typically at the time that it is requested from a client. The JSP translator is triggered by the `.jsp` file name extension in a URL. The translated page is then executed, processing HTTP requests and generating responses similarly to any other servlet. Coding a JSP page is more convenient than coding the equivalent servlet.

JSP pages are fully interoperable with servlets. A JSP page can include output from a servlet or forward to a servlet, and a servlet can include output from a JSP page or forward to a JSP page.

Here is the code for a simple JSP page, `welcomeuser.jsp`:

```
<HTML>
<HEAD><TITLE>The Welcome User JSP</TITLE></HEAD>
<BODY>
<% String user=request.getParameter("user"); %>
<H3>Welcome <%= (user==null) ? "" : user %>!</H3>
<P><B> Today is <%= new java.util.Date() %>. Have a fabulous day! :-)</B></P>
<B>Enter name:</B>
<FORM METHOD=get>
<INPUT TYPE="text" NAME="user" SIZE=15>
<INPUT TYPE="submit" VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

This JSP page will produce something like the following output if the user inputs the name "Amy":

```
Welcome Amy!

Today is Wed Jun 21 13:42:23 PDT 2000. Have a fabulous day! :-)
```

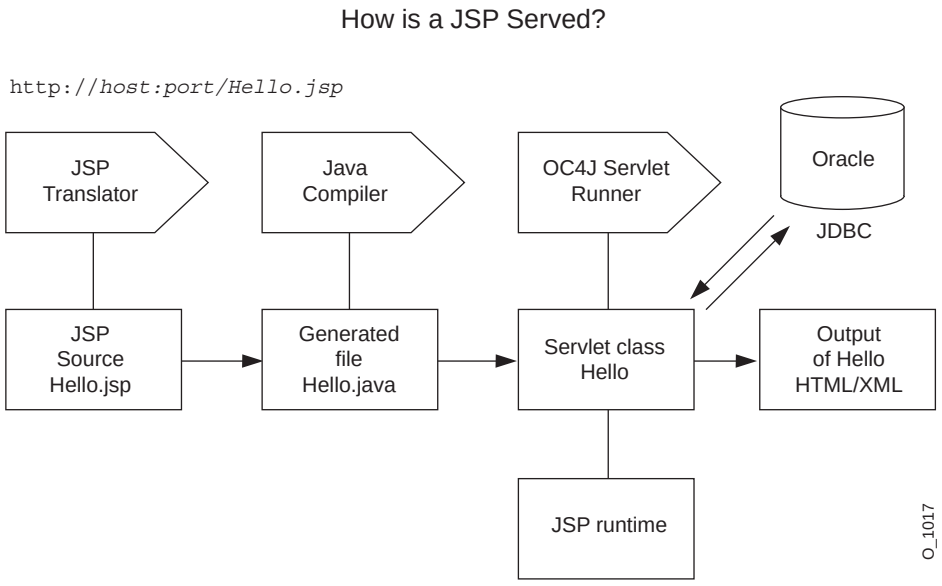
JSP Translation and Runtime Flow

Figure 6–1 shows the flow of execution when a user runs a JSP page, specifying its URL in the browser. Assume that `Hello.jsp` accesses a database.

Because of the `.jsp` file name extension, the following steps occur automatically:

1. The JSP translator is invoked, translating `Hello.jsp` and producing the file `Hello.java`.
2. The Java compiler is invoked, creating `Hello.class`.
3. `Hello.class` is executed as a servlet, using the JSP runtime library.
4. The `Hello` class accesses the database through JDBC (or perhaps SQLJ), as appropriate, and sends its output to the browser.

Figure 6–1 JSP Translation and Runtime Flow



Key JSP Advantages

For most situations, there are at least two general advantages to using JSP pages instead of servlets:

- **Coding convenience**: JSP syntax provides a shortcut for coding dynamic Web pages, typically requiring much less code than equivalent servlet code. The JSP translator also automatically handles some servlet coding overhead for you, such as implementing standard JSP or servlet interfaces and creating HTTP sessions. JSP tag libraries, typically supplied with J2EE products such as OC4J, provide even further programming convenience.
- **Separation of static content and dynamic content**: Realistically, a JSP developer will need some knowledge of Java. However, JSP technology attempts to allow some separation between the HTML code development for static content, and the Java code development for business logic and dynamic content. This makes JSP programming accessible and attractive to Web designers, as it simplifies the division of maintenance responsibilities between presentation and layout specialists who might be more proficient in HTML than in Java, and code specialists who might be more proficient in Java than in HTML. In a typical JSP page, most Java code and business logic will not be within snippets embedded

in the JSP page. Instead, business logic will be in JavaBeans or Enterprise JavaBeans that are invoked from the JSP page.

Overview of Oracle Value-Added Features for JSP Pages

The OC4J JSP implementation provides the following extended functionality through custom tag libraries and custom JavaBeans and classes that are generally portable to other JSP environments. These features are documented in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

- Support for the JavaServer Pages Standard Tag Library (JSTL)
- Extended types implemented as JavaBeans that can have a specified scope
- `JspScopeListener` for event-handling
- Integration with XML and XSL
- Data-access tag library (sometimes referred to as "SQL tags") and JavaBeans
- The JSP Markup Language (JML) custom tag library, which reduces the level of Java proficiency required for JSP development
- Oracle Personalization tag library
- OracleAS Web Services tag library
- Tag libraries and JavaBeans for uploading files, downloading files, and sending e-mail from within an application
- EJB tag library
- Additional utility tags (such as for displaying dates and currency amounts appropriately for a specified locale)

In addition, the OC4J JSP container offers integration with caching technologies, documented in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*:

- JESI tags for Edge Side Includes
- Web Object Cache tags and API

The OC4J JSP container also supports Oracle-specific programming extensions, such as for globalization support. These are documented in the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*:

Running a Simple JSP Page

This section shows you how to run the elementary JSP example from earlier in this chapter.

Create and Deploy welcomeuser.jsp

Copy or type the sample code from "What Is JavaServer Pages Technology?" on page 6-2 into a file, and save it as `welcomeuser.jsp`.

Archive `welcomeuser.jsp` into the WAR and EAR files for the JSP primer samples, and deploy the EAR file using the Oracle Enterprise Manager deployment wizard. This is all described in "Creating and Deploying the JSP Primer Samples EAR File" on page 6-14.

Run welcomeuser.jsp

Assuming you specify a URL mapping of `/jspprimer`, as described in "Deploy the EAR File" on page 6-16, you can run `welcomeuser.jsp` with a URL such as the following:

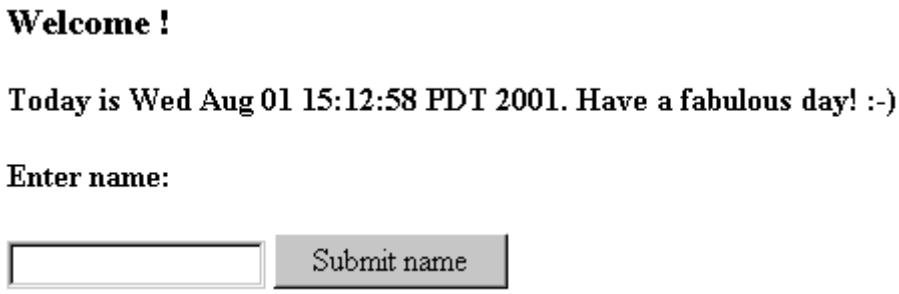
```
http://host:port/jspprimer/welcomeuser.jsp
```

This uses *host* to represent the name of the system where Oracle Application Server and the application are installed.

If the JSP were within a subdirectory below the top level of the WAR file, then this directory must be included in the URL. For example, if `welcomeuser.jsp` is located in the `mydir` directory in the WAR file, you would invoke it as follows:

```
http://host:port/jspprimer/mydir/welcomeuser.jsp
```

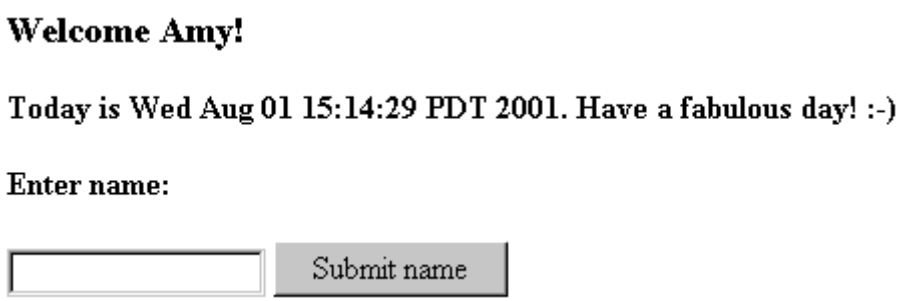
When you first run the page, you will see something like the following output:



Submitting a name, such as "Amy", rewrites the URL to indicate the input:

`http://host:port/jspprimer/welcomeuser.jsp?user=Amy`

and updates the page:



Running a JSP Page That Invokes a JavaBean

As mentioned earlier, JSP technology works nicely as a front-end for business logic and dynamic functionality encapsulated in JavaBeans. This section contains the code for a JavaBean and a JSP page that calls it.

The steps involved are covered in the following sections:

- Create the JSP: usebean.jsp
- Create the JavaBean: NameBean.java
- Deploy usebean.jsp and Namebean.java
- Run usebean.jsp

Create the JSP: usebean.jsp

This section lists the source for a JSP page that uses the standard JSP `jsp:useBean` tag to invoke a JavaBean. To run the code, you can copy or type it into a file called `usebean.jsp`. The numbers in JSP comment statements along the right edge correspond to the notes following the code.

```
<%@ page import="beans.NameBean"  %>                              <%--1--%>

<jsp:useBean id="pageBean" class="beans.NameBean" scope="page" /><%--2--%>
<jsp:setProperty name="pageBean" property="*" />                  <%--3--%>

<HTML>
<HEAD> <TITLE> The Use Bean JSP </TITLE> </HEAD>
<BODY BGCOLOR=white>

<H3> Welcome to the Use Bean JSP </H3>

<% if (pageBean.getNewName().equals("")) { %>
  I don't know you.
<% } else { %>
  Hello <%= pageBean.getNewName() %> !
<%  } %>

<P>May we have your name?
<FORM METHOD=get>
<INPUT TYPE=TEXT name=newName size = 20>
<INPUT TYPE=SUBMIT VALUE="Submit name">
</FORM>
</BODY>
</HTML>
```

Code Notes

1. This is a JSP construct called a `page` directive. In this case, it imports the JavaBean class. (There are other uses for `page` directives, and other kinds of directives.)
2. The standard `jsp:useBean` tag instantiates the JavaBean, specifying the package name, class name, and instance name. A scope setting of `page` specifies that the JavaBean instance is accessible only from the JSP page where it was created. Other possible scopes are `request`, `session`, and `application`.
3. The standard `jsp:setProperty` tag sets the values of one or more properties for the specified bean instance. A property setting of `*` results in iteration over the HTTP request parameters, matching bean property names with request

parameter names, and setting bean property values according to the corresponding request parameter values. In this case, the only bean property is `newName`. This corresponds to the `newName` HTTP request parameter specified in the HTML forms code in the page. (For use with the `jsp:useBean` tag, in addition to the `jsp:setProperty` tag, there is a `jsp:getProperty` tag to retrieve parameter values.)

For general information about any of these topics, see the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide.*

Create the JavaBean: NameBean.java

Here is the code for the JavaBean class, `NameBean`. The package name specified here must be consistent with the `page` directive and the `jsp:useBean` tag of the JSP page. To run the JSP page, you can copy or type this code into a file called `NameBean.java` and then compile it. The resulting `NameBean.class` file must be placed in a `beans` subdirectory under the `WEB-INF/classes` directory for your application, according to the `beans` package name.

```
package beans;

public class NameBean {

  String newName="";

  public void NameBean() {  }

  public String getNewName() {
    return newName;
  }
  public void setNewName(String newName) {
    this.newName = newName;
  }
}
```

Deploy usebean.jsp and Namebean.java

Archive `usebean.jsp` and `Namebean.class` into the WAR and EAR files for the JSP primer samples, and deploy the EAR file using the Oracle Enterprise Manager deployment wizard. This is all described in "Creating and Deploying the JSP Primer Samples EAR File" on page 6-14.

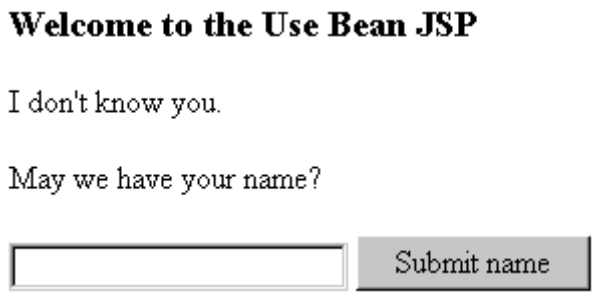
Run usebean.jsp

Assuming you specify a URL mapping of `/jspprimer`, as described in "Deploy the EAR File" on page 6-16, you can run `usebean.jsp` with a URL such as the following:

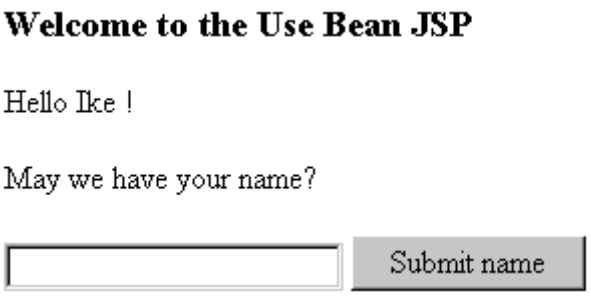
`http://host:port/jspprimer/usebean.jsp`

This example, as before, uses *host* as the name of the system where Oracle Application Server and the application are installed.

When you run this page, you will initially see the following output:



Once you submit a name, such as "Ike", the page is updated (but still prompts you in case you want to enter another name):



Running a JSP Page That Uses Custom Tags

The Sun Microsystems JavaServer Pages specification includes standard tags to use in JSP pages to perform various tasks. An example is the `jsp:useBean` tag

employed in "Running a JSP Page That Invokes a JavaBean" on page 6-7. The JSP specification also outlines a standard framework that allows vendors to offer their own custom tag libraries in a portable way.

Each custom tag library has a *tag library descriptor* (TLD) file that specifies the tags, tag attributes, and other properties of the library. Each tag requires support classes, at least a *tag handler* class with the code to execute tag semantics. (Tag handler classes implement standard tag interfaces, according to the JSP specification.) These classes must be available to your Web application.

OC4J supplies portable tag libraries with functionality in several areas. This section shows an example that uses tags from the Oracle data-access (SQL) tag library to access and query a database and output the results to the browser. A standard JSP `taglib` directive is used to access the TLD file and to specify a tag prefix to use for the tags of this library.

Here are the steps in using a JSP tag library:

The steps involved are covered in the following sections:

- Create the JSP Page: sqltagquery.jsp
- Files for Tag Library Support
- Deploy sqltagquery.jsp
- Run sqltagquery.jsp

For information about the standard tag library framework, including TLD files, tag handler classes, and the `taglib` directive, refer to the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*.

Create the JSP Page: sqltagquery.jsp

This section provides the source for a JSP page that uses data-access tags, supplied with OC4J, to open a database connection, run a simple query, and output the results as an HTML table. To run the page, you can copy or type the code into a file called `sqltagquery.jsp`. The numbers in JSP comment statements along the right edge correspond to the notes following the code.

```
<%@ taglib uri="http://xmlns.oracle.com/j2ee/jsp/tld/ojsp/sqltaglib.tld"
           prefix="sql" %>                                    <%--1--%>
<HTML>
  <HEAD>
    <TITLE>The SQL Tag Query JSP</TITLE>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF">
```

```
    <HR>
    <sql:dbOpen dataSource= "jdbc/OracleDS" >                <%--2--%>
       <sql:dbQuery>                                         <%--3--%>
             select * from EMP
       </sql:dbQuery>
    </sql:dbOpen>                                            <%--4--%>
    <HR>
  </BODY>
</HTML>
```

Code Notes

1. The `taglib` directive uses a JSP 1.2-style `uri` value, which is used as a keyword instead of actually indicating a physical location. (See the *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* for information about such URI functionality.) The `taglib` directive also specifies the tag prefix, "`sql`", to use in any tag statements.
2. `OracleDS` is the JNDI name of a data source that is available by default through the OC4J `data-sources.xml` file. Verify that the data source points to an appropriate database. For introductory information about data sources, see Chapter 4, "Data Sources Primer".
3. By default, the `dbQuery` tag uses the database connection established by the surrounding `dbOpen` tag. Also by default, the `dbQuery` tag outputs its results as an HTML table. Other choices are JDBC result set, XML string, or XML DOM object.
4. Because no explicit connection ID is specified in the `dbOpen` start-tag, the database connection is automatically closed when the `dbOpen` end-tag is reached.

For more information about the data-access tag library that is supplied with OC4J, refer to the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Files for Tag Library Support

JSP applications require some JAR files that are installed with OC4J:

- `ojsp.jar` (JSP container)
- `ojsputil.jar` (OC4J tag libraries and utilities)
- `xmlparserv2.jar` (XML parser)

- `xsu12.jar` (XML-SQL utility)

The tag handler class files and TLD files, including the tag handlers and TLD file (`sqltaglib.tld`) for this example, are in `ojsputil.jar`.

> **Note:** The library `ojsputil.jar` also gives you access to data-access JavaBeans and other Java utility classes that come with OC4J. These classes are described in the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Deploy sqltagquery.jsp

Archive `sqltagquery.jsp` into the WAR and EAR files for the JSP primer samples, and deploy the EAR file using the Oracle Enterprise Manager deployment wizard. This is all described in "Creating and Deploying the JSP Primer Samples EAR File" on page 6-14.

Run sqltagquery.jsp

Assuming you specify a URL mapping of `/jspprimer`, as described in "Deploy the EAR File" on page 6-16, you can run `sqltagquery.jsp` with a URL such as the following:

```
http://host:port/jspprimer/sqltagquery.jsp
```

This page produces output such as the following.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	1980-12-17 00:00:00.0	800		20
7499	ALLEN	SALESMAN	7698	1981-02-20 00:00:00.0	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22 00:00:00.0	1250	500	30
7566	JONES	MANAGER	7839	1981-04-02 00:00:00.0	2975		20
7654	MARTIN	SALESMAN	7698	1981-09-28 00:00:00.0	1250	1400	30
7698	BLAKE	MANAGER	7839	1981-05-01 00:00:00.0	2850		30
7782	CLARK	MANAGER	7839	1981-06-09 00:00:00.0	2450		10
7788	SCOTT	ANALYST	7566	1987-04-19 00:00:00.0	3000		20
7839	KING	PRESIDENT		1981-11-17 00:00:00.0	5000		10
7844	TURNER	SALESMAN	7698	1981-09-08 00:00:00.0	1500	0	30
7876	ADAMS	CLERK	7788	1987-05-23 00:00:00.0	1100		20
7900	JAMES	CLERK	7698	1981-12-03 00:00:00.0	950		30
7902	FORD	ANALYST	7566	1981-12-03 00:00:00.0	3000		20
7934	MILLER	CLERK	7782	1982-01-23 00:00:00.0	1300		10

Important: The Oracle JDBC driver classes are supplied with Oracle Application Server, but you must ensure that they are compatible with your JDK and your database version. The `classes12.zip` or `classes12.jar` library is for JDK 1.2.x or higher. Also, the driver release number must be compatible with your database release number.

Creating and Deploying the JSP Primer Samples EAR File

Three examples have been covered in this chapter: a simple JSP page, a page that calls a JavaBean, and a page that uses a tag library. For simplicity, we suggest that you package and deploy them in a single WAR file within an EAR file. This section

shows all the required files for the examples and how you can package and deploy them.

Note: Web components, such as JSP pages and servlets, are archived in WAR files. An application consisting of only Web components can be deployed as a WAR file directly; however, the primary J2EE deployment vehicle is an EAR file. An EAR file can include a WAR file as well as other types of archive files for other types of components, such as EJBs. This chapter uses an EAR file. See "Creating and Deploying the Servlet Primer Samples WAR File" on page 5-14 for an example using only a WAR file.

EAR and WAR File Structure

Here is the structure of the EAR file for the examples in this chapter:

```
jsp-primer-samples.war
META-INF/
   application.xml
   manifest.mf
```

And here is the structure of the nested WAR file:

```
sqltagquery.jsp
usebean.jsp
welcomeuser.jsp
WEB-INF/
   web.xml
   classes/
      beans/
         NameBean.class
```

If you use an IDE for your development, the IDE can usually create the WAR and EAR files for you. Otherwise, you can create them manually using the Java JAR utility or the `ant` utility.

However the standard `application.xml` file is created (either manually or through an IDE), it must have a `<web-uri>` entry that indicates the WAR file name, and a `<context-root>` entry that indicates the context path portion of the URL. The DTD for the standard `application.xml` file is according to the J2EE specification. Here is a sample file:

```
<?xml version="1.0" ?>
<!DOCTYPE application (View Source for full doctype...)>
```

```
<application>
  <display-name>OracleAS JAAS Provider Application: SimpleServlet</display-name>
  <module>
    <web>
      <web-uri>callerInfo-web.war</web-uri>
      <context-root>/callerInfo</context-root>
    </web>
  </module>
</application>
```

Note: There are no relevant or required `web.xml` entries for the examples in this chapter.

Deploy the EAR File

Use Oracle Enterprise Manager, the Application Server Control, to deploy the examples. Select the Applications page. Click the **Deploy EAR file** button to start the deployment wizard.

"Deploying Applications" on page 2-14 discusses the steps involved in deploying an application to OC4J. Only three of these steps are required to deploy the examples in this chapter:

1. Specify the application name.

   In the Deploy Application: Select Application Page, specify the J2EE application and the application name. You can use the same name in both places. In this example, we use `jsp-primer-samples` (here and in the `application.xml` file).

   See "Select Application" on page 2-15 for information.

2. Specify the URL mapping for the application.

   In the Deploy Application: URL Mapping for Web Modules Page, the servlet context path for your application must be specified. Typically, Oracle Enterprise Manager can pick this up from the `<context-root>` element of the `application.xml` file. For this discussion, assume `/jspprimer` is the context path. This will be part of the URL you would use to invoke the application.

   See "Provide The URL Mappings For All Web Modules" on page 2-17 for information.

3. Deploy the application.

In the Deploy Application: Summary Page, you can look over the application summary and then click the **Deploy** button.

See "Deployment Review" on page 2-22 for information.

7

EJB Primer

After you have installed Oracle Application Server Containers for J2EE (OC4J) and configured the base server and default Web site, you can start developing J2EE applications. This chapter assumes that you have a working familiarity with simple J2EE concepts and a basic understanding for EJB development.

The following sections describe how to develop and deploy EJB applications with OC4J:

- Develop EJBs—Developing and testing an EJB module within the standard J2EE specification.
- Prepare the EJB Application for Assembly—Before deploying, you must modify an XML file that acts as a standard J2EE application descriptor file for the enterprise application.
- Deploy the Enterprise Application to OC4J—Archive the enterprise Java application into an Enterprise ARchive (EAR) file and deploy it to OC4J.

This chapter uses a stateless session bean example to show you each development phase and deployment steps for an EJB. As an introduction to EJBs, a simple EJB with a basic OC4J-specific configuration is used. You can download the stateless session bean example from the OC4J sample code page at `http://otn.oracle.com/tech/java/oc4j/demos` on the OTN Web site.

For more information on EJBs in OC4J, see the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.*

Develop EJBs

You develop EJB components for the OC4J environment in the same way as in any other standard J2EE environment. Here are the steps to develop EJBs:

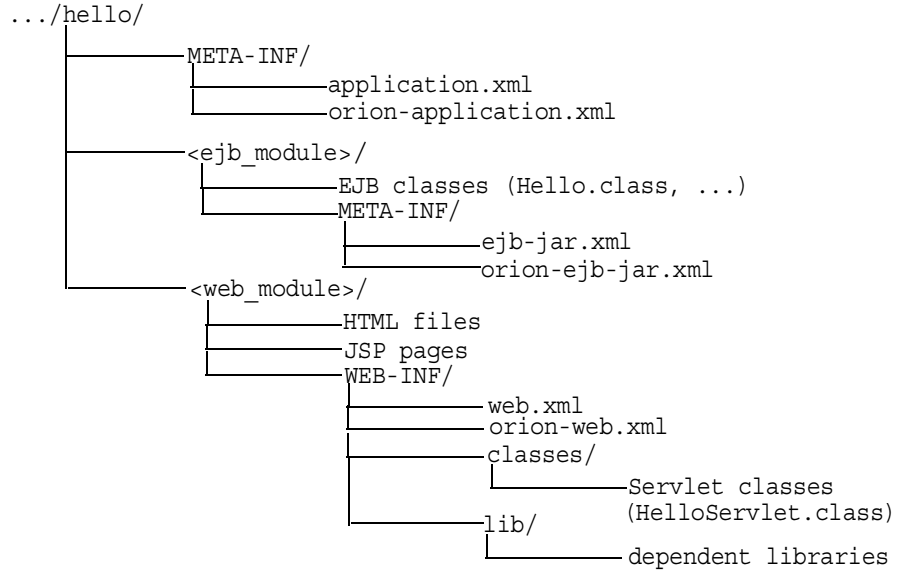
1. Create the Development Directory—Create a development directory for the enterprise application (as Figure 7–1 shows).
2. Implement the EJB—Develop your EJB with its home interfaces, component interfaces, and bean implementation.
3. Access the EJB—Develop the client to access the bean through the remote or local interface.
4. Create the Deployment Descriptor—Create the standard J2EE EJB deployment descriptor for all beans in your EJB application.
5. Archive the EJB Application—Archive your EJB files into a JAR file.

Create the Development Directory

Although you can develop your application in any manner, we encourage you to use consistent naming for locating your application easily. One method would be to implement your enterprise Java application under a single parent directory structure, separating each module of the application into its own subdirectory.

Our hello example was developed using the directory structure mentioned in "Creating the Development Directory" on page 2-6. Notice in Figure 7–1 that the EJB and Web modules exist under the `hello` application parent directory and are developed separately in their own directory.

Figure 7–1 Hello Directory Structure



Note: For EJB modules, the top of the module (`ejb_module`) represents the start of a search path for classes. As a result, classes belonging to packages are expected to be located in a nested directory structure beneath this point. For example, a reference to a package class '`myapp.Hello.class`' is expected to be located in "`...hello/ejb_module/myapp/Hello.class`".

Implement the EJB

When you implement a session or entity EJB, create the following:

> **Note:** Message-driven beans have similar, but not the same, requirements as listed below. See the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for more information.

1. The home interfaces for the bean. The home interface defines the `create` method for your bean. If the bean is an entity bean, it also defines the finder method(s) for that bean.
   a. The remote home interface extends `javax.ejb.EJBHome`.
   b. The local home interface extends `javax.ejb.EJBLocalHome`.
2. The component interfaces for the bean.
   a. The remote interface declares the methods that a client can invoke remotely. It extends `javax.ejb.EJBObject`.
   b. The local interface declares the methods that a collocated bean can invoke locally. It extends `javax.ejb.EJBLocalObject`.
3. The bean implementation includes the following:
   a. The implementation of the business methods that are declared in the component interfaces.
   b. The container callback methods that are inherited from either the `javax.ejb.SessionBean` or `javax.ejb.EntityBean` interfaces.
   c. The `ejb*` methods that match the home interface `create` methods:
      * For stateless session beans, provide an `ejbCreate` method with no parameters.
      * For stateful session beans, provide an `ejbCreate` method with parameters matching those of the create method as defined in the home interfaces.
      * For entity beans, provide `ejbCreate` and `ejbPostCreate` methods with parameters matching those of the `create` method as defined in the home interfaces.

Creating the Home Interfaces

The home interfaces (remote and local) are used to create the bean instance; thus, they define the `create` method for your bean. Each type of EJB can define the `create` method in the following ways:

EJB Type	Create Parameters
Stateless Session Bean	Can have only a single `create` method, with no parameters.
Stateful Session Bean	Can have one or more `create` methods, each with its own defined parameters.
Entity Bean	Can have zero or more `create` methods, each with its own defined parameters. All entity beans must define one or more finder methods, where at least one is a `findByPrimaryKey` method.

For each `create` method, a corresponding `ejbCreate` method is defined in the bean implementation.

Remote Invocation Any remote client invokes the EJB through its remote interface. The client invokes the `create` method that is declared within the remote home interface. The container passes the client call to the `ejbCreate` method—with the appropriate parameter signature—within the bean implementation. You can use the parameter arguments to initialize the state of the new EJB object.

1. The remote home interface must extend the `javax.ejb.EJBHome` interface.
2. All `create` methods must throw the following exceptions:
   - `javax.ejb.CreateException`
   - `javax.ejb.EJBException` or another `RuntimeException`

Example 7–1 Remote Home Interface for Session Bean

The following code sample illustrates a remote home interface for a session bean called `HelloHome`.

```
package hello;

import javax.ejb.*;
import java.rmi.*;

public interface HelloHome extends EJBHome
```

```
{
  public Hello create() throws CreateException, RemoteException;
}
```

Local Invocation An EJB can be called locally from a client that exists in the same container. Thus, a collocated bean, JSP, or servlet invokes the `create` method that is declared within the local home interface. The container passes the client call to the `ejbCreate` method—with the appropriate parameter signature—within the bean implementation. You can use the parameter arguments to initialize the state of the new EJB object.

1. The local home interface must extend the `javax.ejb.EJBLocalHome` interface.
2. All `create` methods may throw the following exceptions:
   - `javax.ejb.CreateException`
   - `javax.ejb.EJBException` or another `RuntimeException`

Example 7–2 Local Home Interface for Session Bean

The following code sample shows a local home interface for a stateless session bean called `HelloLocalHome`.

```
package hello;

import javax.ejb.*;

public interface HelloLocalHome extends EJBLocalHome
{
 public HelloLocal create() throws CreateException, EJBException;
}
```

Creating the Component Interfaces

The component interfaces define the business methods of the bean that a client can invoke.

Creating the Remote Interface The remote interface defines the business methods that a remote client can invoke. Here are the requirements for developing the remote interface:

1. The remote interface of the bean must extend the `javax.ejb.EJBObject` interface, and its methods must throw the `java.rmi.RemoteException` exception.
2. You must declare the remote interface and its methods as `public` for remote clients.
3. The remote interface, all its method parameters, and return types must be serializable. In general, any object that is passed between the client and the EJB must be serializable, because RMI marshals and unmarshals the object on both ends.
4. Any exception can be thrown to the client, as long as it is serializable. Runtime exceptions, including `EJBException` and `RemoteException`, are transferred back to the client as remote runtime exceptions.

Example 7–3 Remote Interface Example for Hello Session Bean

The following code sample shows a remote interface called `Hello` with its defined methods, each of which will be implemented in the stateless session bean.

```
package hello;

import javax.ejb.*;
import java.rmi.*;

public interface Hello extends EJBObject
{
  public String sayHello(String myName) throws RemoteException;
}
```

Creating the Local Interface The local interface defines the business methods of the bean that a local (collocated) client can invoke.

1. The local interface of the bean must extend the `javax.ejb.EJBLocalObject` interface.
2. You declare the local interface and its methods as `public`.

Example 7–4 Local Interface for Hello Session Bean

The following code sample contains a local interface called `HelloLocal` with its defined methods, each of which will be implemented in the stateless session bean.

```
package hello;
```

```
import javax.ejb.*;

public interface HelloLocal extends EJBLocalObject
{
  public String sayHello(String myName) throws EJBException;
}
```

Implementing the Bean

The bean contains the business logic for your application. It implements the following methods:

1. The signature for each of these methods must match the signature in the remote or local interface, except that the bean does not throw the `RemoteException`. Since both the local and the remote interfaces use the bean implementation, the bean implementation cannot throw the `RemoteException`.
2. The lifecycle methods are inherited from the `SessionBean` or `EntityBean` interface. These include the `ejb<Action>` methods, such as `ejbActivate`, `ejbPassivate`, and so on.
3. The `ejbCreate` methods that correspond to the `create` method(s) that are declared in the home interfaces. The container invokes the appropriate `ejbCreate` method when the client invokes the corresponding `create` method.
4. Any methods that are private to the bean or package used for facilitating the business logic. This includes private methods that your public methods use for completing the tasks requested of them.

Example 7–5 Hello Session Bean Implementation

The following code shows the bean implementation for the hello example.

> **Note:** You can download this example on OTN from the OC4J sample code page at `http://otn.oracle.com/tech/java/oc4j/demos`.

```
package hello;

import javax.ejb.*;
```

```
public class HelloBean implements SessionBean
{
  public SessionContext ctx;

  public HelloBean()
  {    // constructor
  }

  public void ejbCreate() throws CreateException
  {   // when bean is created
  }

  public void ejbActivate()
  {    // when bean is activated
  }

  public void ejbPassivate()
  {    // when bean is deactivated
  }

  public void ejbRemove()
  {   // when bean is removed
  }

  public void setSessionContext(SessionContext ctx)
  {    this.ctx = ctx;
  }

  public void unsetSessionContext()
  {    this.ctx = null;
  }

  public String sayHello(String myName) throws EJBException
  {
    return ("Hello " + myName);
  }
}
```

Note: You can download this example on OTN from the OC4J sample code page at `http://otn.oracle.com/tech/java/oc4j/demos`.

Access the EJB

All EJB clients perform the following steps to instantiate a bean, invoke its methods, and destroy the bean:

1. Look up the home interface through a JNDI lookup. Follow JNDI conventions for retrieving the bean reference, including setting up JNDI properties if the bean is remote to the client.
2. Narrow the returned object from the JNDI lookup to the home interface, as follows:
   a. When accessing the remote interface, use the `PortableRemoteObject.narrow` method to narrow the returned object.
   b. When accessing the local interface, cast the returned object with the local home interface type.
3. Create instances of the bean in the server through the returned object. Invoking the `create` method on the home interface causes a new bean to be instantiated and returns a bean reference.

   Note: For entity beans that are already instantiated, you can retrieve the bean reference through one of its finder methods.

4. Invoke business methods, which are defined in the component (remote or local) interface.
5. After you are finished, invoke the `remove` method. This will either remove the bean instance or return it to a pool. The container controls how to act on the `remove` method.

Important: In order to access EJBs, the client-side must download `oc4j_client.zip` file from `http://otn.oracle.com/software/products/ias/devuse.html`. Unzip the JAR into a directory that is in your CLASSPATH. This JAR contains the classes necessary for client interaction. If you download this JAR into a browser, you must grant certain permissions. See the "Granting Permissions" section of the Security chapter in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for a list of these permissions.

If the client is not on an Oracle Application Server installation, you must also download the `optic.jar` file, which is located in `ORACLE_HOME/opmn/lib` on your Oracle Application Server installation. This JAR file must be in the CLASSPATH.

Example 7–6 A Servlet Acting as a Local Client

The following example is executed from a servlet that is collocated with the Hello bean. Thus, the session bean uses the local interface, and the JNDI lookup does not require JNDI properties.

Note: The JNDI name is specified in the `<ejb-local-ref>` element in the session bean EJB deployment descriptor as follows:

```
<ejb-local-ref>
  <ejb-ref-name>ejb/HelloBean</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <local-home>hello.HelloLocalHome</local-home>
  <local>hello.HelloLocal</local>
</ejb-local-ref>
```

```
package hello;

import javax.servlet.http.*;
import javax.servlet.*;
import javax.ejb.*;
import javax.naming.*;
import java.io.IOException;

public class HelloServlet extends HttpServlet
```

```
{
  HelloLocalHome helloHome;
  HelloLocal hello;

  public void init() throws ServletException
  {
    try {
     // 1. Retreive the Home Interface using a JNDI Lookup
     // Retrieve the initial context for JNDI.
     // No properties needed when local
      Context context = new InitialContext();

      // Retrieve the home interface using a JNDI lookup using
      // the java:comp/env bean environment variable
      // specified in web.xml
      helloHome = (HelloLocalHome)
             context.lookup("java:comp/env/ejb/HelloBean");

     //2. Narrow the returned object to be an HelloHome object.
     // Since the client is local, cast it to the correct object type.
     //3. Create the local Hello bean instance, return the reference
      hello = (HelloLocal)helloHome.create();

    } catch(NamingException e) {
      throw new ServletException("Error looking up home", e);
    } catch(CreateException e) {
      throw new ServletException("Error creating local hello bean", e);
    }
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException
  {
    response.setContentType("text/html");
    ServletOutputStream out = response.getOutputStream();
    try
    {
      out.println("<html>");
      out.println("<body>");
     //4. Invoke a business method on the local interface reference.
```

```
      out.println(hello.sayHello("James Earl"));
      out.println("</body>");
      out.println("</html>");
    } catch(EJBException e) {
      out.println("EJBException error: " + e.getMessage());
    } catch(IOException e) {
      out.println("IOException error: " + e.getMessage());
    } finally {
      out.close();
    }
  }
}
```

Note: You can download this example on OTN from the OC4J sample code page at `http://otn.oracle.com/tech/java/oc4j/demos`.

Example 7–7 A Java Client as a Remote Client

The following example is executed from a pure Java client that is a remote client. Any remote client must set up JNDI properties before retrieving the object, using a JNDI lookup.

Note: The JNDI name is specified in the `<ejb-ref>` element in the client's `application-client.xml` file—as follows:

```
<ejb-ref>
    <ejb-ref-name>ejb/HelloBean</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>hello.HelloHome</home>
    <remote>hello.Hello</remote>
 </ejb-ref>
```

The `jndi.properties` file for this client is as follows:

```
java.naming.factory.initial=
       com.evermind.server.ApplicationClientInitialContextFactory
java.naming.provider.url=opmn:ormi://opmnhost:oc4j_inst1/helloworld
java.naming.security.principal=admin
java.naming.security.credentials=welcome
```

The pure Java client that invokes `Hello` remotely is as follows:

```
package hello;

import javax.ejb.*;
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.io.*;
import java.util.*;
import java.rmi.RemoteException;

/*
 * A simple client for accessing an EJB.
 */

public class HelloClient
{
  public static void main(String[] args)
  {
    System.out.println("client started...");
    try {
      // Initial context properties are set in the jndi.properties file
      //1. Retrieve remote interface using a JNDI lookup*/
      Context context = new InitialContext();

     // Lookup the HelloHome object. The reference is retrieved from the
     // application-local context (java:comp/env). The variable is
     // specified in the application-client.xml).
      Object homeObject = context.lookup("java:comp/env/Helloworld");

     //2. Narrow the reference to HelloHome. Since this is a remote
     //   object, use the PortableRemoteObject.narrow method.
      HelloHome home = (HelloHome) PortableRemoteObject.narrow
                                        (homeObject, HelloHome.class);

     //3. Create the remote object and narrow the reference to Hello.
      Hello remote =
        (Hello) PortableRemoteObject.narrow(home.create(), Hello.class);

     //4. Invoke a business method on the remote interface reference.
      System.out.println(remote.sayHello("James Earl"));
```

```
    } catch(NamingException e) {
      System.err.println("NamingException: " + e.getMessage());
    } catch(RemoteException e) {
      System.err.println("RemoteException: " + e.getMessage());
    } catch(CreateException e) {
      System.err.println("FinderException: " + e.getMessage());
    }
  }
}
```

Note: You can download this example on OTN from the OC4J sample code page at `http://otn.oracle.com/tech/java/oc4j/demos`.

Create the Deployment Descriptor

After implementing and compiling your classes, you must create the standard J2EE EJB deployment descriptor for all beans in the module. The XML deployment descriptor (defined in the `ejb-jar.xml` file) describes the EJB module of the application. It describes the types of beans, their names, and attributes. The structure for this file is mandated in the DTD file, which is provided at "`http://java.sun.com/dtd/ejb-jar_2_0.dtd`".

Any EJB container services that you want to configure is also designated in the deployment descriptor. For information about data sources and JTA, see the *Oracle Application Server Containers for J2EE Services Guide*. For information about security, see the *Oracle Application Server Containers for J2EE Security Guide*.

After creation, place the deployment descriptors for the EJB application in the `META-INF` directory that is located in the same directory as the EJB classes. See Figure 7–1 for more information.

The following example shows the sections that are necessary for the `Hello` example, which implements both a remote and a local interface.

Example 7–8 XML Deployment Descriptor for Hello Bean

The following is the deployment descriptor for a version of the hello example that uses a stateless session bean. This example defines both the local and remote interfaces. You do not have to define both interface types; you may define only one of them.

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
   <display-name>hello</display-name>
   <description>
      An EJB app containing only one Stateless Session Bean
   </description>
   <enterprise-beans>
      <session>
         <description>no description</description>
         <display-name>HelloBean</display-name>
         <ejb-name>HelloBean</ejb-name>
         <home>hello.HelloHome</home>
         <remote>hello.Hello</remote>
         <local-home>hello.HelloLocalHome</local-home>
         <local>hello.HelloLocal</local>
         <ejb-class>hello.HelloBean</ejb-class>
         <session-type>Stateless</session-type>
         <transaction-type>Container</transaction-type>
      </session>
   </enterprise-beans>

   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>HelloBean</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Supports</trans-attribute>
      </container-transaction>
      <security-role>
         <role-name>users</role-name>
      </security-role>
   </assembly-descriptor>
</ejb-jar>
```

> **Note:** You can download this example on OTN from the OC4J sample code page at `http://otn.oracle.com/tech/java/oc4j/demos`.

Archive the EJB Application

After you have finalized your implementation and created the deployment descriptors, archive your EJB application into a JAR file. The JAR file should include all EJB application files and the deployment descriptor.

> **Note:** If you have included a Web application as part of this enterprise Java application, follow the instructions for building the Web application in the *Oracle Application Server Containers for J2EE User's Guide.*

For example, to archive your compiled EJB class files and XML files for the `Hello` example into a JAR file, perform the following in the `../hello/ejb_module` directory:

```
% jar cvf helloworld-ejb.jar .
```

This archives all files contained within the `ejb_module` subdirectory within the JAR file.

Prepare the EJB Application for Assembly

To prepare the application for deployment, you do the following:

1. Modify the `application.xml` file with the modules of the enterprise Java application.
2. Archive all elements of the application into an EAR file.

These steps are described in the following sections:

- Modify the Application.xml File
- Create the EAR File

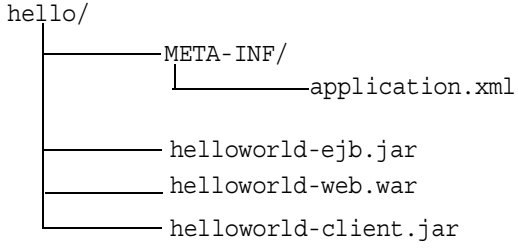
Modify the Application.xml File

The `application.xml` file acts as the standard J2EE application descriptor file for the application and contains a list of the modules that are included within your enterprise application. You use each `<module>` element defined in the `application.xml` file to designate what comprises your enterprise application. Each module describes one of three things: EJB JAR, Web WAR, or any client files. Respectively, designate the `<ejb>`, `<web>`, and `<java>` elements in separate `<module>` elements.

- The `<ejb>` element specifies the EJB JAR filename.
- The `<web>` element specifies the Web WAR filename in the `<web-uri>` element, and its context in the `<context>` element.
- The `<java>` element specifies the client JAR filename, if any.

As Figure 7–2 shows, the `application.xml` file is located under a `META-INF` directory under the parent directory for the application. The JAR, WAR, and client JAR files should be contained within this directory. Because of this proximity, the `application.xml` file refers to the JAR and WAR files only by name and relative path—not by full directory path. If these files were located in subdirectories under the parent directory, then these subdirectories must be specified in addition to the filename.

Figure 7–2 Archive Directory Format



For example, the following example modifies the `<ejb>`, `<web>`, and `<java>` module elements within `application.xml` for the `Hello` EJB application that also contains a servlet that interacts with the EJB.

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.2//EN" "http://java.sun.com/j2ee/dtds/application_1_
2.dtd">
```

```
<application>
  <display-name>helloworld j2ee application</display-name>
  <description>
     A sample J2EE application that uses a Helloworld Session Bean
     on the server and calls from java/servlet/JSP clients.
  </description>
  <module>
    <ejb>helloworld-ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>helloworld-web.war</web-uri>
      <context-root>/helloworld</context-root>
    </web>
  </module>
  <module>
    <java>helloworld-client.jar</java>
  </module>
</application>
```

Create the EAR File

Create the EAR file that contains the JAR, WAR, and XML files for the application. Note that the `application.xml` file serves as the EAR application descriptor file.

To create the `helloworld.ear` file, execute the following in the `hello` directory contained in Figure 7–2:

```
% jar cvf helloworld.ear .
```

This step archives the `application.xml`, the `helloworld-ejb.jar`, the `helloworld-web.war`, and the `helloworld-client.jar` files into the `helloworld.ear` file.

Deploy the Enterprise Application to OC4J

After archiving your application into an EAR file, deploy the application to OC4J. See "Deploying Applications" on page 2-14 for information on how to deploy your application.

8

OC4J Clustering

This chapter discusses OC4J clustering, and provides instructions on how to manage OC4J processes and applications within a cluster.

It contains the following topics:

- The OC4J Instance in a Cluster
- Instance-Specific Parameters
- OC4J Clustering Examples
- OC4J Cluster Configuration

The full picture of clustering within Oracle Application Server is discussed in the *Oracle Application Server 10g High Availability Guide*.

The OC4J Instance in a Cluster

The OC4J instance is the entity to which J2EE applications are deployed and configured. It defines how many OC4J processes exist within the application server and the configuration for these OC4J processes. The OC4J process is what executes the J2EE applications for the OC4J instance.

The OC4J instance has the following features:

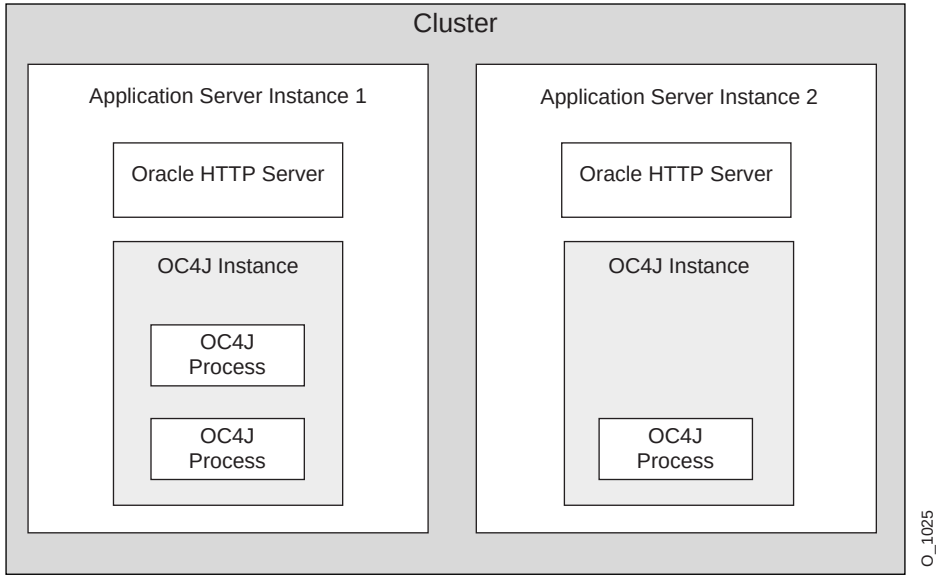
- The configuration of the OC4J instance is valid for one or more OC4J executable processes. This way, you can duplicate the configuration for multiple OC4J processes by managing these processes in the OC4J instance construct. When you modify the cluster-wide configuration within the OC4J instance, the modifications are valid for all OC4J processes.
- Each OC4J instance can be configured with one or more OC4J processes.
- When you deploy an application to an OC4J instance, the OC4J instance deploys the application to all OC4J processes defined in the OC4J instance. The OC4J instance is also responsible for replicating the state of its applications.
- The number of OC4J processes is specific to each OC4J instance. This must be manually configured for each application server instance in the cluster. The OC4J process configuration provides flexibility to tune according to the specific hardware capabilities of the host. By default, each OC4J instance is instantiated with a single OC4J process.

Within the application sever instance, you can configure multiple OC4J instances, each with its own number of OC4J processes. The advantage for this is for configuration management and application deployment for separate OC4J processes in your cluster.

Figure 8–1 demonstrates the `home` default OC4J instance. In the context of a cluster, the OC4J instance configuration is part of the cluster-wide configuration. Thus, the `home` instance, configured on the first application instance, is replicated on all other application server instances.

The number of processes in each `home` instance is an instance-specific parameter, so you must configure the `home` instance separately on each application server instance for the number of OC4J processes that exist on each application server instance. Figure 8–1 shows that the `home` instance on application server instance 1 contains two OC4J processes; the `home` instance on application server instance 2 contains only one OC4J process. Each OC4J instance defaults to having one OC4J process.

Figure 8–1 OC4J Processes in a Cluster



The OC4J Process in a Cluster

The OC4J process is the JVM process that executes J2EE applications. Each OC4J process is contained in an OC4J instance and inherits its configuration from the OC4J instance. All applications deployed to an OC4J instance are deployed to all OC4J processes in the OC4J instance.

You can define one or more OC4J processes within an OC4J instance, so that J2EE requests can be load balanced and have failover capabilities.

The configuration for the number of OC4J processes is instance-specific. Thus, you must configure each OC4J instance in each application server instance with the number of OC4J processes you want to start up for that OC4J instance. The default is one OC4J process.

Each host that you install the application server instances on has different capabilities. To maximize the hardware capabilities, configure the number of OC4J processes in each OC4J instance that will use these capabilities properly. For example, you can configure a single OC4J process on host A and five OC4J processes on host B.

When you define multiple OC4J processes, you enable the following:

- You can serve multiple users with multiple OC4J processes.
- You can provide failover if the state of the application is replicated across multiple OC4J processes.
- OHS provides load balancing for all OC4J processes in the OC4J instance. The OPMN component notifies each OHS when a new OC4J process is initiated. Thus, each OHS in the cluster knows of each OC4J process in the cluster.

Replicating Application State

The OC4J processes involved in the cluster can replicate application state to all OC4J processes. Once you configure replication, OC4J handles the propagation of the application state for you.

If one OC4J process fails, then another OC4J process—which has had the application state replicated to it—takes over the application request. When an OC4J process fails during a stateful request, the OHS forwards the request in the following order:

1. If another OC4J process is active within the same application server instance, OHS forwards the request to this process.
2. Otherwise, OHS forwards the state request to an OC4J process in another application server instance in the cluster.

There are two types of failure that you want to protect against: software failure and hardware failure.

Failure Type	Avoidance Technique
Software failure occurs when the OC4J process fails.	Multiple OC4J processes in the same OC4J instance. When one OC4J process fails, the OHS forwards the request to another OC4J process in the same OC4J instance.
Hardware failure occurs when the host goes down.	OC4J processes in the cluster configured on separate hosts. When the first host dies, the OC4J process on another host can take over the request. This requires that you have installed an application server instance on another host, which is a part of the cluster, and the OC4J instance has at least one OC4J process.

Islands

An island is a logical grouping of OC4J processes that allows you to determine which OC4J processes will replicate state.

In each OC4J instance, you can have more than one OC4J process. If we consider state replication in a situation where all OC4J processes tried to replicate state, then the CPU load can significantly increase. To avoid a performance degradation, the OC4J instance enables you to subgroup your OC4J processes. The subgroup is called an island.

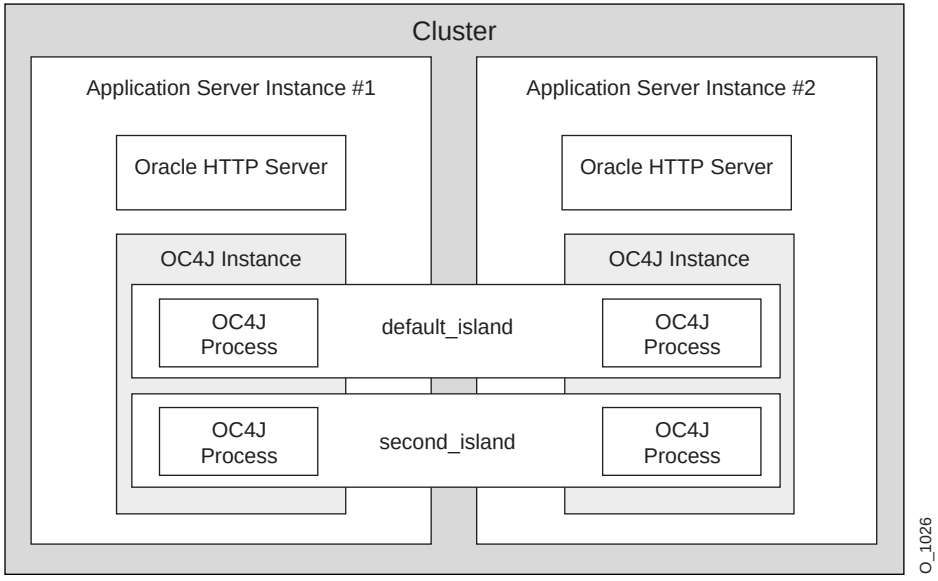
To ensure that the CPU load is partitioned among the processes, the OC4J processes of an OC4J instance can be partitioned into islands. The state for application requests is replicated only to OC4J processes that are grouped within the same island. All applications are still deployed to all OC4J processes in the OC4J instance. The only difference is that the state for these applications is confined to only a subset of these OC4J processes.

The island configuration is instance-specific. The name of the island must be identical in each OC4J instance, where you want the island to exist. When you configure the number of OC4J processes on each application server instance, you can also subgroup them into separate islands. The OC4J processes are grouped across application server instances by the name of the island. Thus, the application state is replicated to all OC4J processes within the island of the same name spanning application server instances.

The grouping of OC4J processes for the state replication is different for EJB applications than for Web applications. Web applications replicate state within the island sub-grouping. EJB applications replicate state between all OC4J processes in the OC4J instance and do not use the island sub-grouping.

Figure 8–2 demonstrates OC4J processes in islands within the cluster. Two islands are configured in the `home` instance: `default_island` and `second_island`. One OC4J process is configured in each island on each application server instance. The OC4J islands, designated within the shaded area, span application server instances.

Figure 8–2 Island Description



J2EE Applications Involved in a Cluster

J2EE applications are deployed in all cases to the OC4J instance—whether the application server instance is included in a cluster or not. However, when the application is deployed to an OC4J instance that is in a cluster, certain configuration details must be accomplished:

- Multicast host and port—The state of the applications is replicated from one OC4J process to another over a multicast address. In the case of an EJB application, you must also specify a username and password. You can either accept the defaults for the multicast address or configure it through the Oracle Enterprise Manager.
- State replication request—You request state replication for all applications through the Oracle Enterprise Manager.
- XML deployment descriptor elements—Both Web and EJB applications require an additional configuration in their respective XML deployment descriptors.
- Island definition—Web applications use the island subgrouping for its state replication. EJB applications ignore the island subgrouping and use all OC4J processes for its state replication.

Instance-Specific Parameters

The following parameters are not replicated across the cluster; thus, each must be set at the OC4J instance level on each application server.

- Islands and number of OC4J processes—While you want to keep the names of the islands consistent across the application server instances, the definition of the islands and the number of OC4J processes is configured independently. The host on which you install each application server instance has different capabilities. On each host, you can tune the number of OC4J processes to match the host capabilities. Remember that the state is replicated in islands across application boundaries. So the island names must be the same in each OC4J instance.
- Port numbers—The RMI, JMS, and AJP port numbers can be different for each host.

OC4J Clustering Examples

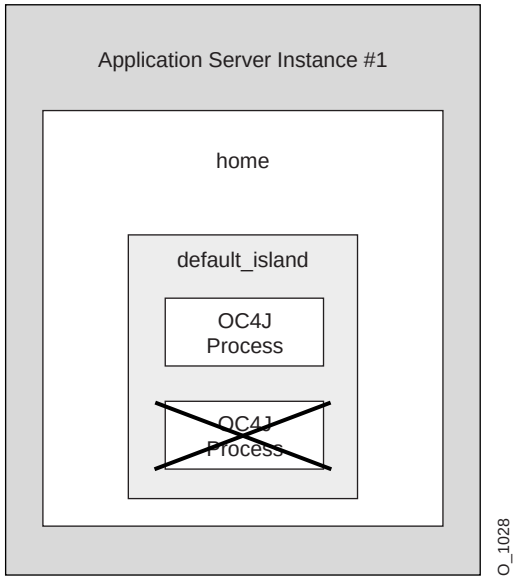
No matter how many application server instances you add within the cluster, the cluster-wide configuration is replicated within the cluster. You control protecting against software and hardware failure with how you configure island and OC4J processes, which are instance-specific parameters.

Software Failure

Suppose you configure more than one OC4J process within your OC4J instance, then if one of these processes fails, another process can take over the work load of the failed process. Figure 8–3 shows application server instance 1, which is involved in the cluster. Within this application server instance, there are two OC4J processes defined in the `default_island` in the `home` instance. If the first OC4J process fails, the other can pick up the work load.

Both of these OC4J processes are on the same host; so, if the host goes down, both OC4J processes fail and the client cannot continue processing.

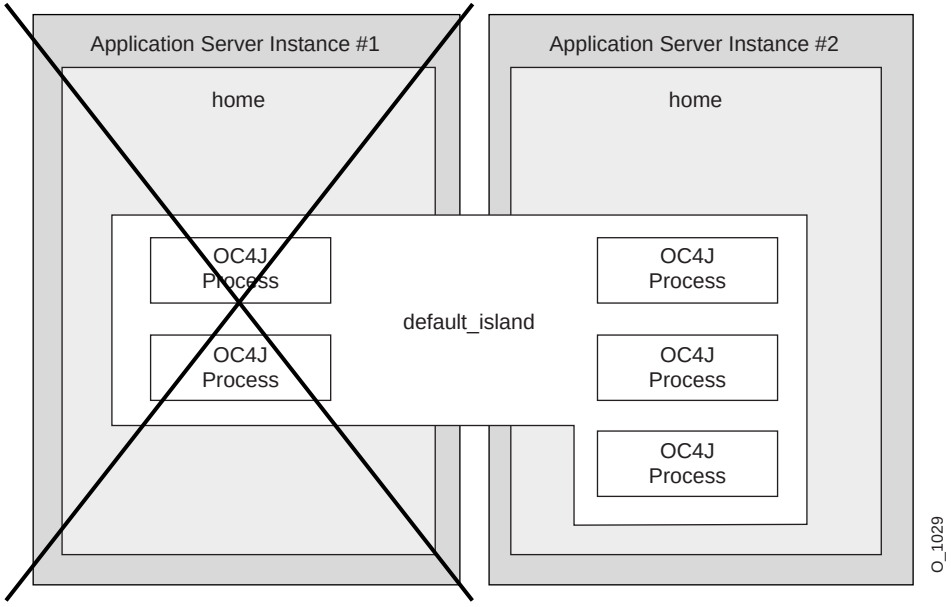
Figure 8–3 Software Failure Demonstration



Hardware Failure

To protect against hardware failure, you must configure OC4J processes in the same OC4J instance across hosts. Figure 8–4 shows `home` instance in application server instance 1 and 2. Within the `default_island`, two OC4J processes are configured on application server instance 1 and three are configured in application server instance 2. If a client is interacting with one of the OC4J processes in application server 1, which terminates abnormally, the client is redirected automatically to one of the OC4J processes in the `default_island` in application server 2. Thus, your client is protected against hardware failure.

Figure 8–4 Hardware Failure Demonstration



State Replication

If the client is a stateful application, then the state is replicated only within the same island. In the previous example, there is only a single island, so the state of the application would be preserved.

To enhance your performance, you want to divide up state replication among islands. However, you must also protect for hardware and software failure within these islands.

The optimal method of protecting against software and hardware failure, while maintaining state with the least number of OC4J processes, is to configure at least one OC4J process on more than one host in the same island. For example, if you have application server instance 1 and 2, within the `home` instance, you configure one OC4J process in the `default_island` on each application server instance. Thus, you are protected against hardware and software failure and your client maintains state if either failure occurs.

- If one of the OC4J processes fails, then the client request is redirected to the other OC4J process in the island. The state is preserved and the client does not notice any irregularity.
- If application server 1 terminates abnormally, then the client is redirected to the OC4J process in the `default_island` on application server 2. The state is preserved and the client does not notice any irregularity.

As demand increases, you will configure more OC4J processes. To guard against a performance slowdown, separate your OC4J processes into separate islands. For example, if fifteen OC4J processes utilize the hardware efficiently on the two hosts and serve the client demand appropriately, then you could divide these processes into at least two islands. The following shows the fifteen OC4J processes grouped into three islands:

Island Names	Application Server 1	Application Server 2
`default_island`	two	three
`second_island`	two	three
`third_island`	three	two

- The host where application server 1 is installed can handle seven OC4J processes; the host where application server 2 is installed can handle eight OC4J processes.
- Each island contains at least one OC4J process in each island across hosts to protect against software and hardware failure.
- The performance is maximized by dividing up the state replication across three islands.

OC4J Cluster Configuration

The following sections describe how to manage OC4J elements in the cluster: EJBs and Servlets. For directions on how to create and modify Oracle Application Server clusters, see the *Oracle Application Server 10g High Availability Guide.*

- OC4J Instance Configuration
- Configuring OC4J Instance-Specific Parameters

> **Note:** As an alternative to using Oracle Enterprise Manager, you can create a cluster, add application server instances to the cluster, and manage the cluster using the DCM command-line tool. See the *Distributed Configuration Management Reference Guide* for information on the DCM command-line tool.

OC4J Instance Configuration

The following sections describe how to configure your OC4J Instance for clustering:

- Configuring Islands and Processes
- Configuring Web Application State Replication
- Configuring EJB Application State Replication
- EJB Clustering Includes JNDI Namespace Replication

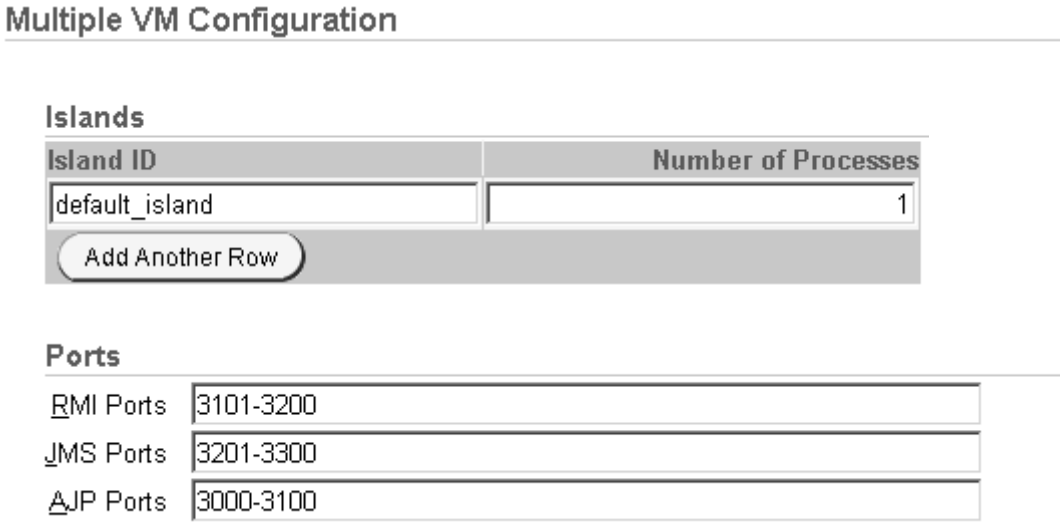
Configuring Islands and Processes

To modify the islands and the number of processes each island contains, do the following:

1. Select the Administration page off the OC4J Home Page.
2. Select Server Properties in the Instance Properties column.
3. Scroll down to the Multiple VM Configuration section. This section defines the islands and the number of OC4J processes that should be started on this application server instance in each island.
4. Create any islands for this OC4J instance within the cluster by clicking **Add Another Row**. You can supply a name for each island within the Island ID field. You can designate how many OC4J processes should be started within each island by the number configured in the Number of Processes field.

Figure 8–5 displays the Multiple VM Configuration section.

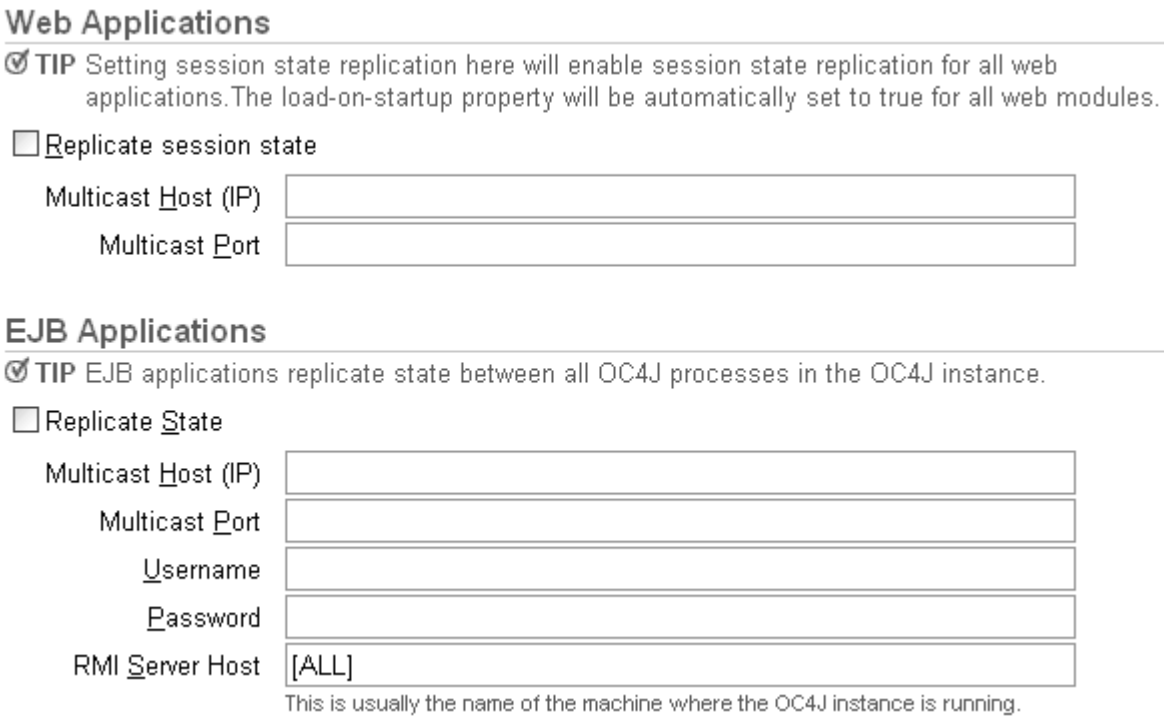
Figure 8–5 Island and Process Configuration



Configuring Web Application State Replication

Configuring state replication for stateful applications is different for Web applications than for EJB applications. To configure state replication for Web applications, do the following as shown in Figure 8–6:

Figure 8–6 Web State Replication Configuration



1. Select the Administration page off of the OC4J Home Page.
2. Select Replication Properties in the Instance Properties column.
3. Scroll down to the Web Applications section. Figure 8–6 shows this section.
4. Select the Replicate session state checkbox.
5. Optionally, you can provide the multicast host IP address and port number. If you do not provide the host and port for the multicast address, it defaults to host IP address 230.230.0.1 and port number 9127. The host IP address must be between 224.0.0.2 through 239.255.255.255. Do not use the same multicast address for both HTTP and EJB multicast addresses.

Note: When choosing a multicast address, ensure that it does not collide with the addresses listed in `http://www.iana.org/assignments/multicast-addresses`. Also, if the low order 23 bits of an address is the same as the local network control block (`224.0.0.0 - 224.0.0.255`), then a collision may occur. To avoid this, provide an address that does not have the same bits in the lower 23 bits of the address as the addresses in this range.

6. Add the `<distributable/>` tag to all `web.xml` files in all Web applications. If the Web application is serializable, you must add this tag to the `web.xml` file.

   The following shows an example of this tag added to `web.xml`:

```
<web-app>
  <distributable/>
  <servlet>
  ...
  </servlet>
</web-app>
```

Configuring EJB Application State Replication

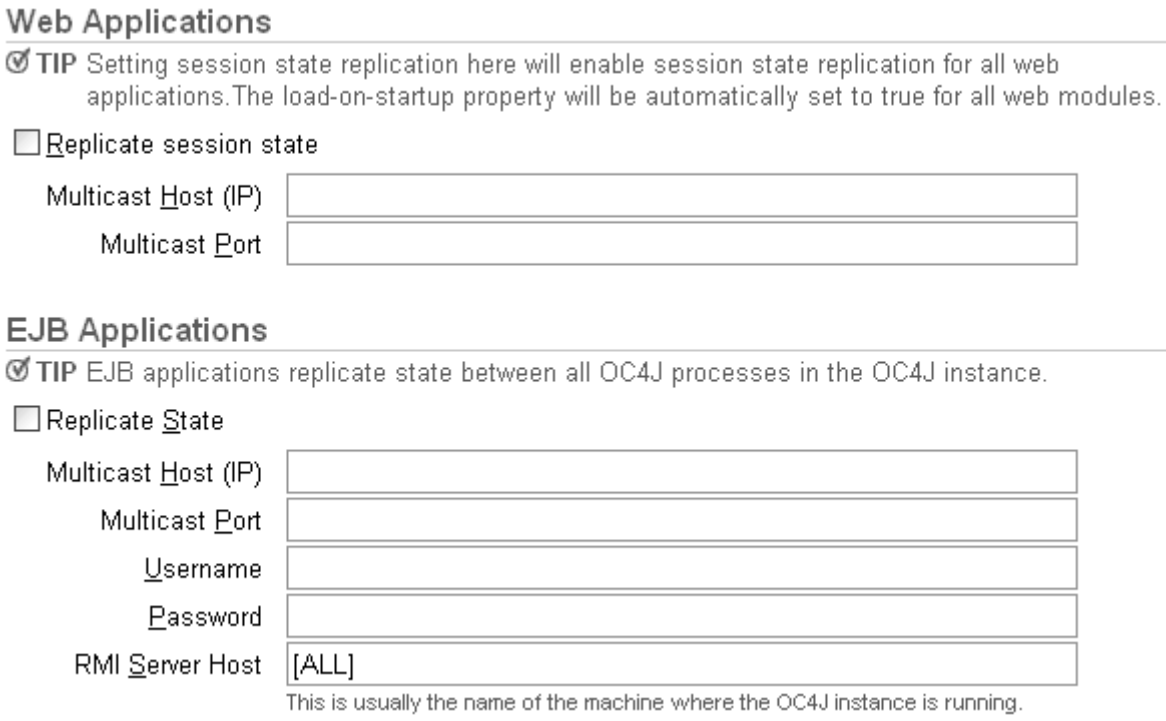
To create an EJB cluster, you specify the OC4J instances that are to be involved in the cluster, configure each of them with the same multicast address, username, and password, and deploy the EJB, which is to be clustered, to each of the nodes in the cluster.

Unlike HTTP clustering, EJBs involved in a cluster cannot be sub-grouped in an island. Instead, all EJBs within the cluster are in one group. Also, only session beans are clustered.

The state of all beans is replicated at the end of every method call to all nodes in the cluster using a multicast topic. Each node included in the EJB cluster is configured to use the same multicast address.

The concepts for understanding how EJB object state is replicated within a cluster are described in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.* To configure EJB replication, you configure the following as shows in Figure 8–7:

Figure 8–7 EJB State Replication Configuration



1. Select the Administration page off of the OC4J Home Page.
2. Select Replication Properties in the Instance Properties column.
3. Scroll down to the EJB Applications section. Figure 8–7 shows this section.
4. Select the Replicate session state checkbox.
5. Provide the username and password, which is used to authenticate itself to other hosts in the cluster. If the username and password are different for other hosts in the cluster, they will fail to communicate. You can have multiple username and password combinations within a multicast address. Those with the same username / password combinations will be considered a unique cluster.

6. Provide the host name where the OC4J Instance resides in the RMI Server Host field.
7. Optionally, you can provide the multicast host IP address and port number. If you do not provide the host and port for the multicast address, it defaults to host IP address 230.230.0.1 and port number 23791. The host IP address must be between 224.0.0.2 through 239.255.255.255. Do not use the same multicast address for both HTTP and EJB multicast addresses.

> **Note:** When choosing a multicast address, ensure that it does not collide with the addresses listed in `http://www.iana.org/assignments/multicast-addresses`. Also, if the low order 23 bits of an address is the same as the local network control block (`224.0.0.0 - 224.0.0.255`), then a collision may occur. To avoid this, provide an address that does not have the same bits in the lower 23 bits of the address as the addresses in this range.

8. Configure the type of EJB replication within the `orion-ejb-jar.xml` file within the JAR file. See "Stateful Session Bean Replication Configuration in the Application JAR" on page 8-16 for full details. You can configure these within the `orion-ejb-jar.xml` file before deployment or add this through the Oracle Enterprise Manager screens after deployment. If you add this after deployment, drill down to the JAR file from the application page.

Stateful Session Bean Replication Configuration in the Application JAR For stateful session beans, you may have you modify the **`orion-ejb-jar.xml`** file to add the state replication configuration. Since you configure the replication type for the stateful session bean within the bean deployment descriptor, each bean can use a different type of replication.

Stateful session beans require state to be replicated among nodes. In fact, stateful session beans must send all their state between the nodes, which can have a noticeable effect on performance. Thus, the following replication modes are available to you to decide on how to manage the performance cost of replication:

JVM Termination Replication The state of the stateful session bean is replicated to only one other node in the cluster (with the same multicast address) when the JVM is terminating. Since this uses JDK 1.3 shutdown hooks, you must use JVM version 1.3 or later. This is the most performant option, because the state is replicated only once. However, it is not very reliable for the following reasons:

- Your state is not replicated if the power is shut off unexpectedly.
- The state of the bean exists only on a single node at any time; the depth of failure is equal to one node.

Set the `replication` attribute of the `<session-deployment>` tag in the `orion-ejb-jar.xml` file to "`VMTermination`". This is shown below:

```
<session-deployment replication="VMTermination" .../>
```

End of Call Replication The state of the stateful session bean is replicated to all nodes in the cluster (with the same multicast address) at the end of each EJB method call. If the node loses power, then the state has already been replicated. The JVM termination replication mode does not guarantee state replication in the case of lost power.

Set the `replication` attribute of the `<session-deployment>` tag in the `orion-ejb-jar.xml` file to "`endOfCall`". This is shown below:

```
<session-deployment replication="EndOfCall" .../>
```

EJB Clustering Includes JNDI Namespace Replication

When EJB clustering is enabled, JNDI namespace replication is also enabled between the OC4J instances in a cluster. New bindings to the JNDI namespace in one OC4J instance are propagated to other OC4J instances in the cluster. Re-bindings and unbindings are not replicated. The 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*.

Configuring OC4J Instance-Specific Parameters

The manageability feature of the cluster causes the configuration to be replicated across all application server instances in the cluster, which is defined as a cluster-wide configuration. However, there are certain parameters where it is necessary to configure them separately on each OC4J instance. These parameters are referred to as OC4J instance-specific.

The following parameters are OC4J instance-specific parameters, which are not replicated across the cluster. You must modify these parameters on each application server instance.

The following are instance-specific parameters within each OC4J instance:

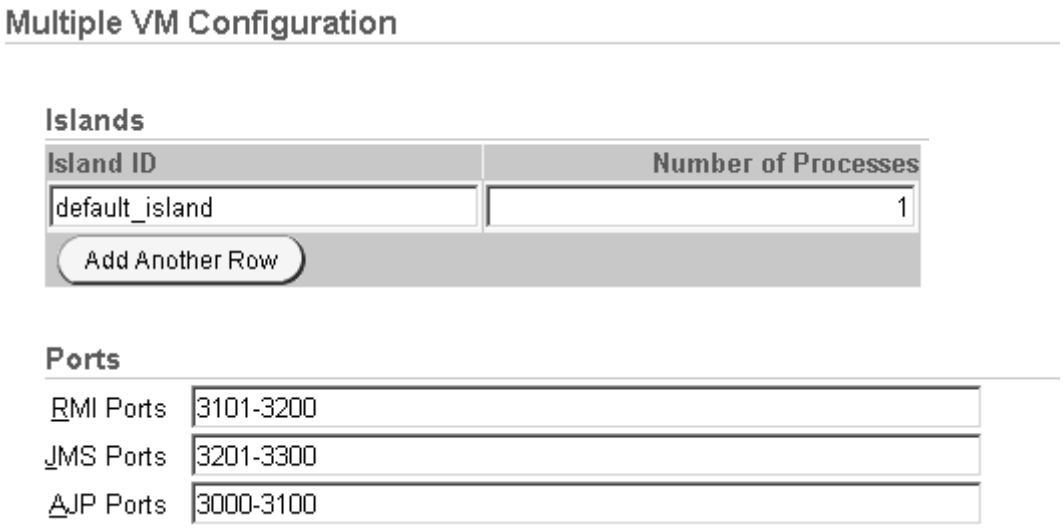
- islands

- number of OC4J processes
- port numbers

All other parameters are part of the cluster-wide parameters, which are replicated across the cluster.

Figure 8–8 shows the section where these parameters are modified. These sections are located in the Server Properties on the Administration page.

Figure 8–8 Non-Replicated Configuration



A

Additional Information

This appendix contains complete information about the following topics:

- Description of XML File Contents
- Elements in the server.xml File
- Elements in the application.xml File
- Elements in the orion-application.xml File
- Elements in the application-client.xml File
- Elements in the orion-application-client.xml File
- Configuration and Deployment Examples
- OC4J Command-Line Options and System Properties

Most of these sections discuss how to modify your XML files. Modify all XML files only through Oracle Enterprise Manager. Do not modify XML files on a single node.

Description of XML File Contents

OC4J uses configuration and deployment XML files. The following sections describe each of these files and their function.

OC4J Configuration XML Files

This section describes the following XML files, which are necessary for OC4J configuration:

- server.xml
- default-web-site.xml
- jazn-data.xml
- principals.xml
- data-sources.xml
- jms.xml
- rmi.xml
- httpds.conf
- mod_oc4j.conf
- workers.properties

server.xml

This file contains the configuration for the application server. The `server.xml` file is the root configuration file—it contains references to other configuration files. In this file, specify the following:

- Library path, which is located in the application deployment descriptor
- Global application, global Web application, and default Web site served
- Maximum number of HTTP connections the server allows
- Logging settings
- Java compiler settings
- Cluster ID
- Transaction time-out
- SMTP host

- Location of the `data-sources.xml` configuration
- Location of the configuration for JMS and RMI
- Location of the default and additional Web sites

  Specify these locations by adding entries that list the location of the Web site configuration files. You can have multiple Web sites. The `default-web-site.xml` file defines a default Web site; therefore, there is only one of these XML files. All other Web sites are defined in `web-site.xml` configuration files. Register each Web site within the `server.xml` file, as follows:

  ```
  <web-site path="./default-web-site.xml" />
  <web-site path="./another-web-site.xml" />
  ```

  Note: The path that is designated is relative to the `config/` directory.

- Pointers to all applications for the container to deploy and execute

  Specify the applications that run on the container in the `server.xml` file. You can have as many application directories as you want, and they do not have to be located under the OC4J installation directory.

default-web-site.xml

This file contains the configuration for a Web site. In the `default-web-site.xml` file, specify the following:

- Host name or IP address, virtual host settings for this site, listener ports, and security using SSL
- Default Web application for this site
- Additional Web applications for this site
- Access-log format
- Settings for user Web applications (for `/~user/ sites`)
- SSL configuration
- Restrict access to the site from one or more hosts

jazn-data.xml

This file contains security information for the OC4J server. It defines the user and group configuration for employing the default `JAZNUserManager`. In the `jazn-data.xml` file, specify the following:

- Username and password for the client-admin console
- Name and description of users/groups, and real name and password for users

principals.xml

This file contains security information for the OC4J server. It defines the user and group configuration for employing the `XMLUserManager`, which is no longer the default security manager. In the `principals.xml` file, specify the following:

- Username and password for the client-admin console
- Name and description of users/groups, and real name and password for users
- Optional X.509 certificates for users

data-sources.xml

This file contains configuration for the data sources that are used. In addition, it contains information on how to retrieve JDBC connections. In the `data-sources.xml` file, specify the following:

- JDBC driver
- JDBC URL
- JNDI paths to which to bind the data source
- User/password for the data source
- Database schema to use
- Inactivity time-out
- Thread policy
- Garbage collection granularity
- Maximum number of connections allowed to the database

Note: Database schemas are used to make auto-generated SQL work with different database systems. OC4J contains an XML file format for specifying properties, such as type-mappings and reserved words. OC4J comes with database schemas for MS SQL Server / MS Access, Oracle, and Sybase. You can edit these or make new schemas for your DBMS.

jms.xml

This file contains the configuration for the OC4J Java Message Service (JMS) implementation. In the `jms.xml` file, specify the following:

- Host name or IP address, and port number to which the JMS server binds
- Settings for queues and topics to be bound in the JNDI tree
- Log settings

rmi.xml

This file contains configuration for the Remote Method Invocation (RMI) system. It contains the setting for the RMI listener, which provides remote access for EJBs. In the `rmi.xml` file, specify the following:

- Host name or IP address, and port number to which the RMI server binds
- Remote servers to which to communicate
- Clustering settings
- Log settings

J2EE Deployment XML Files

The OC4J-specific deployment XML files contain deployment information for different components. If you do not create the OC4J-specific files, they are automatically generated when the application is deployed. You can edit OC4J-specific deployment XML files manually. OC4J uses these files to map environment entries, resources references, and security-roles to actual deployment-specific values.

This section describes the following XML files necessary for J2EE application deployment:

- The J2EE application.xml File

- The OC4J-Specific orion-application.xml File
- The J2EE ejb-jar.xml File
- The OC4J-Specific orion-ejb-jar.xml File
- The J2EE web.xml File
- The OC4J-Specific orion-web.xml File
- The J2EE application-client.xml File
- The OC4J-Specific orion-application-client.xml File

The J2EE application.xml File

This file identifies the Web or EJB applications that are contained within the J2EE application. See "Elements in the application.xml File" on page A-19 for a list of the elements.

The OC4J-Specific orion-application.xml File

This file configures the global application. In the `orion-application.xml` file, specify the following:

- Add files to the library path
- Whether to auto-create and auto-delete tables for CMP beans
- Which default data source to use with CMP beans
- Security role mappings
- Which user manager is the default for security
- JNDI namespace-access rules (authorization)

See "Elements in the orion-application.xml File" on page A-21 for a list of the elements.

The J2EE ejb-jar.xml File

This file defines the deployment parameters for the EJBs in this JAR file. See the Sun Microsystems EJB specification for a description of these elements.

The OC4J-Specific orion-ejb-jar.xml File

This file is the OC4J-specific deployment descriptor for EJBs. In the `orion-ejb-jar.xml` file, specify the following:

- Time-out settings
- Transaction retry settings
- Session persistence settings
- Transaction isolation settings
- CMP mappings
- OR mappings
- Finder method specifications
- JNDI mappings
- Minimum and maximum instance pool settings
- resource reference mappings

See the appendix in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for description of the elements.

The J2EE web.xml File

This file contains deployment information about the servlets and JSPs in this application. See the Sun Microsystems specifications for a description of these elements.

The OC4J-Specific orion-web.xml File

This is the OC4J-specific deployment descriptor for mapping Web settings. This XML file contains the following:

- Auto-reloading (including modification-check time-interval)
- Buffering
- Charsets
- Development mode
- Directory browsing
- Document root
- Locales
- Web timeouts
- Virtual directories

- Clustering
- Session tracking
- JNDI mappings
- Classloading priority for Web applications

See the appendix in the *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for description of the elements.

The J2EE application-client.xml File

This file contains JNDI information for accessing the server application and other client information. See "Elements in the application-client.xml File" on page A-28 for a list of the elements.

The OC4J-Specific orion-application-client.xml File

This OC4J-specific deployment file is for the client application. It contains JNDI mappings and entries for the client. See "Elements in the orion-application-client.xml File" on page A-32 for a list of the elements.

Elements in the server.xml File

The `server.xml` file is where you perform the following tasks:

- Configure OC4J
- Reference other configuration files
- Specify your J2EE application(s)

Configure OC4J

Configuring the OC4J server includes defining the following elements in the `server.xml` file:

- Library path
- Global application, global Web application, and default Web site
- Maximum number of HTTP connections the server allows
- Logging settings
- Java compiler settings

- Cluster ID
- Transaction time-out
- SMTP host

Reference Other Configuration Files

Referencing other configuration files in the `server.xml` file includes specifying the following:

- `data-sources.xml` location
- `jazn-data.xml` location
- `jms.xml` and `rmi.xml` locations

Several XML files and directories are defined in the `server.xml` file. The path to these files or directories can be relative or absolute. If relative, the path should be relative to the location of the `server.xml` file.

<application-server> Element Description of the server.xml file

The top level element of the `server.xml` file is the `<application-server>` element.

`<application-server>`

This element contains the configuration for an application server.

Attributes:

- `application-auto-deploy-directory="../applications/auto"` —Specifies the directory from which EAR files are automatically detected and deployed by the running OC4J server. In addition, it performs the Web application binding for the default Web site.
- `auto-start-applications="true|false"`—If set to `true`, all applications defined in the `<applications>` elements are automatically started when the OC4J server is started. If set to `false`, the applications are not started unless their `auto-start` attribute is set to `true`. The default for `auto-start-applications` is `true`.
- `application-directory="../applications"`— Specifies a directory in which to store applications (EAR files). If none is specified (the default), OC4J stores the information in `j2ee/home/applications`.

- deployment-directory="../application-deployments"—Specifies the master location where applications that are contained in EAR files are deployed. The location defaults to j2ee/home/application-deployments/.
- connector-directory—The location and file name of the oc4j-connectors.xml file.
- check-for-updates="true|false"—This attribute is only used for standalone OC4J.
- recovery-procedure="automatic|prompt|ignore"— Specifies how the EJB container reacts for recovery if an error occurred in the middle of a global transaction (JTA). If a CMP bean is in the middle of a global transaction, the EJB container saves the transactional state to a file. The next time OC4J is started, these attributes specify how to recover the JTA transaction.
  - automatic — automatically attempts recovery (the default)
  - prompt — prompts the user (system in/out)

    You may notice a prompt for recovery even if no CMP beans were executing. This is because the OC4J server asks for permission to see if there was anything to recover.
  - ignore — ignores recovery (useful in development environments or if you are never executing a CMP entity bean)
- taskmanager-granularity=*milliseconds*. The task manager is a background process that performs cleanup. However, the task manager can be expensive. You can manage when the task manager performs its duties through this attribute, which sets how often the task manager is kicked off for cleanup. Value is in milliseconds. Default is 1000 milliseconds.

Elements Contained Within <application-server> of the server.xml file

Within the <application-server> element, the following elements, which are listed alphabetically and not by DTD ordering, can be configured:

<application>

An application is a entity with its own set of users, Web applications, and EJB JAR files.

Attributes:

- auto-start="true|false" — Specifies whether the application should be automatically started when the OC4J server starts. The default is true. Setting

auto-start to false is useful if you have multiple applications installed and want them to start on demand. This can improve general server startup time and resource usage.

- deployment-directory="../application-deployments/myapp" — Specifies a directory to store application deployment information. If none is specified (the default), OC4J looks in the global deployment-directory, and if none exists there, it stores the information inside the EAR file. The path can be relative or absolute. If relative, the path should be relative to the location of the server.xml file.
- name="anApplication" — Specifies the name used to reference the application.
- parent="anotherApplication" — The name of the optional parent application. The default is the global application. Children see the namespace of its parent application. This is used to share services such as EJBs among multiple applications.
- path="../applications/myApplication.ear" /> — The path to the EAR file containing the application code. In this example, the EAR file is named myApplication.ear.

<compiler>

This element is deprecated for version 9.0.4 and forward. See the <java-compiler> element for the alternative. For previous releases, it specifies an alternative compiler (such as Jikes) for EJB/JSP compiling.

Attributes:

- classpath="/my/rt.jar" — Specifies an alternative/additional classpath when compiling. Some compilers need an additional classpath (such as Jikes, which needs the rt.jar file of the Java 2 VM to be included).
- executable="jikes" /> — The name of the compiler executable to use, such as Jikes or JVC.

<cluster>

Cluster settings for this server.

Attribute:

- id="123" /> — The unique cluster ID of the server.

<execution-order>

Defines the ordering of how the startup classes are executed. Value is an integer. OC4J loads from 0 on up. If duplicate numbers, OC4J chooses the ordering for those classes.

<global-application>

The default application for this server. This acts as a parent to other applications in terms of object visibility.

Attributes:

- `name="default"` — Specifies the application.
- `path=".../application.xml" />` — Specifies the path to the global `application.xml` file, which contains the settings for the default application. An `application.xml` file exists for each application as the standard J2EE application descriptor file, which is different than this file. This `application.xml` may have the same name, but it exists to provide global settings for all J2EE applications.

<global-thread-pool>

You can specify unbounded, one, or two thread pools for an OC4J process through this element. If you do not specify this element, then an infinite number of threads can be created. See "Thread Pool Settings" on page 3-42 for a full description.

Attributes:

- `min` —The minimum number of threads that OC4J can simultaneously execute. By default, a minimum number of threads are preallocated and placed in the thread pool when the container starts. Value is an integer. The default is 20. The minimum value you can set this to is 10.
- `max` —The maximum number of threads that OC4J can simultaneously execute. New threads are spawned if the maximum size is not reached and if there are no idle threads. Idle threads are used first before a new thread is spawned. Value is an integer. The default is 40.
- `queue` —The maximum number of requests that can be kept in the queue. Value is an integer. The default is 80.
- `keepAlive` —The number of milliseconds to keep a thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains alive. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one.

Value is a long. The default is 600000 milliseconds.

- `cx-min` —The minimum number of connection threads that OC4J can simultaneously execute. Value is an integer. The default is 20. The minimum value you can set this to is 10.
- `cx-max` —The maximum number of connection threads that OC4J can simultaneously execute. Value is an integer. The default is 40.
- `cx-queue` —The maximum number of connection requests that can be kept in the queue. Value is an integer. The default is 80.
- `cx-keepAlive` —The number of milliseconds to keep a connection thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains alive. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one.

  Value is a long. The default is 600000 milliseconds.
- `debug` —If true, print the application server thread pool information at startup. The default is false.

<global-web-app-config>

Attributes:

- `path`— The path where the `web-application.xml` file is located.

```
path=".../web-application.xml" />
```

<init-library>

Attributes:

- `path`— The path in which the startup and shutdown classes are located. The path indicates the directory in which the class resides or the directory and JAR filename of the JAR where the class is archived. If more than one directory or JAR file exists, then supply an `<init-library>` element for each directory and JAR filename.

```
<init-library path="../xxx">
```

<init-param>

Attributes:

- Defines the key-value pairs of the parameters to pass into the startup class.

<javacache-config>

Attributes:

- path—Specifies the path to the javacache.xml file.

```
<javacache-config path="../../../javacache/admin/javacache.xml" />
```

<java-compiler>

You can specify an alternative compiler—either in or out of process—for your JSP and EJB compilation. The default compiler is an out of process javac compiler found in the JDK bin directory.

Attributes:

- name—Specify the name of the compiler to use. Valid compiler names are as follows:
  - for in-process compilers—modern, classic, javac or ojc
  - for out-of-process compilers (forked)—modern, javac, ojc, or jikes

  These names are defined as follows:
  - javac—the standard compiler name for all JDKs.
  - classic—the standard compiler of JDK 1.1/1.2.
  - modern—the standard compiler of JDK 1.3/1.4.
  - jikes—the Jikes compiler.
  - ojc—The Oracle Java compiler.
- in-process—If true, the compiler is to run in process. If false, the compiler runs out of the process. Most compilers can execute both in and out of the process. The exceptions are as follows:
  - The classic compiler cannot run out of the process; thus, the in-process attribute is always true.
  - The jikes compiler cannot run in process; thus, the in-process attribute is always false.
- encoding—Specify the type of character encoding for the source file, such as UTF-8, EUCJIS, or SJIS. Encoding is only supported by the javac compiler. The default is determined by the language version of the JVM that is installed.

- bindir—Provide the absolute path to the compiler directory. You do not need to specify this attribute for javac, modern, or classic as the JDK bin directory is searched for this compiler.

  The syntax is specific to the operating system platform:

  * Sun Microsystems Solaris example—If you are using jikes, which is in /usr/local/bin/jikes, then specify the following:

    ```
    name="jikes"
    bindir="/usr/local/bin"
    ```

  * Windows example—To specify jikes, which is located in c:\jdk1.3.1\bin\jikes.exe, specify the following:

    ```
    name="jikes"
    bindir="c:\\jdk1.3.1\\bin"
    ```

- extdirs—Specifies extension directories that the compilation uses to compile against. The default is your JDK extension directories. Multiple directories can be specified, each separated by a colon. Each JAR archive in the specified directories are searched for class files. You can specify certain directories to be searched by modifying the -Djava.ext.dirs system property. The jikes compiler requires that extension directories are specified in either this attribute or in the -Djava.ext.dirs system property.

The following are four examples of how to define alternate compilers in this element:

```
<java-compiler name="jikes" bindir="C:\java\jikes\bin"
         in-process="false" />
<java-compiler name="ojc" bindir="C:\java\jdev\jdev\bin"
         in-process="false"/>
<java-compiler name="classic" in-process="true" />
<java-compiler name="modern" in-process="true" />
```

<jms-config>

Attribute:

- path— Specifies the path to the jms.xml file.

  ```
  path=".../jms.xml"
  ```

<log>

<file>

Attribute:

- `path="../log/server.log"` — Specifies a relative or absolute path to a file where log events are stored.

<mail>

An e-mail address where log events are forwarded. You must also specify a valid mail-session if you use this option.

Attribute:

- `address="my@mail.address"` — Specifies the mail address.

<odl>

The ODL log entries are each written out in XML format in its respective log file. The log files have a maximum limit. When the limit is reached, the log files are overwritten.

When you enable ODL logging, each message goes into its respective log file, named `log`*N*`.xml`, where N is a number starting at one. The first log message starts the log file, `log1.xml`. When the log file size maximum is reached, the second log file is opened to continue the logging, `log2.xml`. When the last logfile is full, the first log file, `log1.xml` is erased and a new one is opened for the new messages. Thus, your log files are constantly rolling over and do not encroach on your disk space.

Attributes:

- `path`: Path and folder name of the log folder for this area. You can use an absolute path or a path relative to where the configuration XML file exists, which is normally in the `j2ee/home/config` directory. This denotes where the log files will reside for the feature that the XML configuration file is concerned with. For example, modifying this element in the `server.xml` file denotes where the server log files are written.
- `max-file-size`: The maximum size in KB of each individual log file.
- `max-directory-size`: The maximum size of the directory in KB. The default directory size is 10 MB.

New files are created within the directory, until the maximum directory size is reached. Each log file is equal to or less than the maximum specified in the attributes.

<max-http-connections>

Used to define the maximum number of concurrent connections any given Web site can accept at a single point in time. If text exists inside the tag, it is used as a redirect-URL when the limit is reached.

Attributes:

- `max-connections-queue-timeout="10"` — When the maximum number of connections are reached, this is the number of seconds that can pass before the connections are dropped and a message is returned to the client stating that the server is either busy or connections will be redirected. The default is 10 seconds.
- `socket-backlog` — The number of connections to queue up before denying connections at the socket level. The default is 30.
- `value` — The maximum number of connections.

<rmi-config>

Attribute:

- `path`— Specifies the path to the `rmi.xml` file.

```
path=".../rmi.xml"
```

<sep-config>

The `<sep-config>` element in this file specifies the pathname, normally `internal-settings.xml`, for the server extension provider properties.

Attribute:

- `path`—The path of the server extension provider properties.

<sfsb-config>

Passivation for stateful session beans is automatically done, unless you set the `enable-passivation` attribute for this element to false. For more information on stateful session bean passivation, see the Advanced chapter in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.*

Attribute

- `enable-passivation`—Default is true, which means that stateful session bean passivation occurs. If you have a situation where your stateful session beans are not in a state to be passivated, set this attribute to false.

<shutdown-classes>

Shutdown classes can be defined by the user, and are executed after undeployment, but before the core services are stopped.

<shutdown-class>

Each startup class is defined within the `<startup-class>` element.

Attributes:

- `classname`—The classname of the user-defined startup class.

<startup-classes>

Startup classes can be defined by the user, and will be executed after the core services (JMS, RMI) are started, but before applications are deployed. The shutdown classes are executed after undeployment, but before the core services are stopped.

<startup-class>

Each startup class is defined within the `<startup-class>` element.

Attributes:

- `classname`—The classname of the user-defined startup class.
- `failure-is-fatal`—If true, if an exception is thrown, then OC4J logs the exception and exit. If false, OC4J logs the exception and then continues. Default is false.

<transaction-config>

Transaction configuration for the server.

Attribute:

- `timeout="30000"` — Specifies the maximum amount of time (in milliseconds) that a transaction can take to finish before it is rolled back due to a timeout. The default value is 30000. This timeout will be a default timeout for all transactions that are started in OC4J. You can change it by using the dynamic API—`UserTransaction.setTransactionTimeout(milliseconds)`.

<web-site>

Attribute:

- `path`— The path to a `*web-site.xml` file that defines a Web site. For each Web site, you must specify a separate `*web-site.xml` file. This example shows that a Web site is defined in the `my-web-site.xml` file.

```
path=".../my-web-site.xml"
```

Elements in the application.xml File

<application> Element Description of the application.xml file

The top level element of the `application.xml` file is the `<application>` element.

Elements Contained Within <application> of the application.xml file

Within the `<application>` element, the following elements, which are listed alphabetically and not by DTD ordering, can be configured:

`<alt-dd>path/to/dd</alt-dd>`

The `alt-dd` element specifies an optional URI to the post-assembly version of the deployment descriptor file for a particular J2EE module. The URI must specify the full pathname of the deployment descriptor file relative to the application's root directory. If alt-dd is not specified, the deployer must read the deployment descriptor from the default location and file name required by the respective component specification.

`<connector>context</connector>`

The `connector` element specifies the URI of a resource adapter archive file, relative to the top level of the aplication package.

`<context-root>thedir/</context-root>`

The `context-root` element specifies the context root of a web application.

`<description>A description.</description>`

The `description` element provides a human readable description of the application. The `description` element should include any information that the application assembler wants to provide the deployer.

`<display-name>The name.</display-name>`

The `display-name` element specifies an application name. The application name is assigned to the application by the application assembler and is used to identify the application to the deployer at deployment time.

`<ejb>pathToEJB.jar</ejb>`

The `ejb` element specifies the URI of a EJB JAR, relative to the top level of the application package.

<icon>

The icon element contains a small-icon and a large-icon element which specify the location within the application for a small and large image used to represent the application in a GUI tool.

<java>pathToClient.jar</java>

The java element specifies the URI of a Java application client module, relative to the top level of the application package.

<large-icon>path/to/icon.gif</large-icon>

The large-icon element contains the location within the application of a file containing a large (32x32 pixel) icon image. The image must be either GIF or JPEG format and the filename must end with the extension of ".gif" or ".jpg".

<module>

The module element represents a single J2EE module and contains an EJB, Java, or Web element, which indicates the module type and contains a path to the module file, and an optional alt-dd element, which specifies an optional URI to the post-assembly version of the deployment descriptor. The application deployment descriptor must have one module element for each J2EE module in the application package.

<role-name>nameOfRole</role-name>

The name of the role.

<security-role>

The security-role element contains the definition of a security role which is global to the application. The definition consists of a description of the security role, and the security role name. The descriptions at this level override those in the component level security role definitions and must be the descriptions tool display to the deployer.

<small-icon>path/to/icon.gif</small-icon>

The small-icon element contains the location within the application of a file containing a small (16x16 pixel) icon image. The image must be either GIF or JPEG format and the filename must end with the extension of ".gif" or ".jpg".

<web>

The web element contains the web-uri and context-root of a Web application module.

<web-uri>pathTo.war</web-uri>

The web-uri element specifies the URI of a web application file, relative to the top level of the application package.

Elements in the orion-application.xml File

<orion-application> Element Description of the orion-application.xml file

The top level element of the orion-application.xml file is the <orion-application> element.

Attributes:

- autocreate-tables - Whether or not to automatically create database tables for CMP beans in this application. The default is true.
- autodelete-tables - Whether or not to automatically delete old database tables for CMP beans when redeploying in this application. The default is false.
- default-data-source - The default data source to use if other than server default. This must point to a valid CMT data source for this application if specified.
- deployment-version - The version of OC4J that this JAR was deployed against, if it is not matching the current version then it will be redeployed. This is an internal server value; do not edit.
- treat-zero-as-null - Whether or not to treat read zero's as null's when they represent primary keys. The default is false.

Elements Contained Within <orion-application> of the orion-application.xml file

Within the <orion-application> element, the following elements, which are listed alphabetically and not by DTD ordering, can be configured:

<argument value="theValue" />

An argument used when invoking the client.

Attribute:

- value - The value of the argument.

<arguments>

A list of arguments to used when invoking the application client if starting it in-process (auto-start="true").

<client-module auto-start="true|false" deployment-time="073fc2ab513bc3ce" path="myappclient.jar" user="theUser">

An application client module of the application. An application client is a GUI or console-based standalone client that interacts with the server.

Attributes:

- `auto-start` - Whether or not to auto-start the client (in-process) at server startup. The default is false.
- `deployment-time` - Last deploy time attribute. Internal to OC4J; do not edit.
- `path` - The path (relative to the enterprise archive or absolute) to the application-client.
- `user` - User to run the client as if run in-process (autostart="true"). Must be specified if auto-start is activated.

<commit-coordinator>

Configure the two-phase commit engine.

<commit-class class="com.evermind.server.OracleTwoPhaseCommitDriver" />

Attribute:

- `class` - Configures the `OracleTwoPhaseCommitDriver` class for two-phase commit engines.

<connectors path="./oc4j-connectors.xml" />

Attribute:

- `path` - The name and path of the `oc4j-connectors.xml` file. If no `<connectors>` element is specified, then the default path is `$OC4J_HOME/connectors/`*`rarname`*`./oc4j-connectors.xml`.

<data-sources path="./data-sources.xml" />

Attribute:

- `path` - The path.

<description>A short description</description>

A short description of this component.

<ejb-module path="myEjbs.jar" remote="true|false" />

An EJB JAR module of the application.

Attributes:

- `path` - The path (relative to the enterprise archive or absolute) to the ejb-jar.
- `remote` - `true`/`false` value stating whether or not to activate the EJB instances on this node or to look them up remotely from another server (remote or inside a cluster). The default is false.

<file path="../log/server.log" />

A relative/absolute path to log events to.

Attribute:

- `path` - The path.

<group name="theGroup" />

A group that this security-role-mapping implies. That is, all members of the specified group are included in this role.

Attribute:

- `name` - The name of the group.

<jazn provider="XML" location="./jazn-data.xml" />

Configure the OracleAS JAAS Provider to use the XML-based provider type.

Attributes:

- `provider` - XML
- `location` - Path to file. For example: `./jazn-data.xml` This can be an absolute path, or a path relative to the `jazn.xml` file, where the OracleAS JAAS Provider first looks for the `jazn-data.xml` in the directory containing the `jazn.xml` file. Optional if `jazn.xml` file configured, otherwise Required
- `persistence` - Values can be `NONE` (Do not persist changes), `ALL` (Persist changes after every modification), `VM_EXIT` - (Default- Persist changes when VM exits)
- `default-realm` - A realm name. For example: `sample_subrealm`. Optional if only one realm is configured.

<jazn-web-app auth-method="SSO" runas-mode="false" doasprivileged-mode="true" />

The filter element of `JAZNUserManager`.

Attributes:

- `auth-method` - Set auth-method to SSO (single sign-on). If you do not set this parameter, it defaults to null.
- The `runas-mode` and `doasprivileged-mode` settings are described in Table A–1. See the *Oracle Application Server Containers for J2EE Security Guide* for more information.

Table A–1 runas-mode and doasprivileged-mode Settings

If runas-mode is Set To...	If doasprivileged-mode Is Set To...	Then...
true	true (default)	`Subject.doAsPrivileged` in a `privilegedExceptionAction` block that calls `chain.doFilter (myrequest,response)`
true	false	`Subject.doAs` in a `privilegedExceptionAction` block that calls `chain.doFilter (myrequest,response)`
false (default)	true	`chain.doFilter (myrequest,response)`
false	false	`chain.doFilter (myrequest,response)`

<library path="../lib/" />

A relative / absolute path / URL to a directory or a JAR / ZIP to add as a library-path for this server. Directories are scanned for JARS / ZIP files to include at startup.

Attribute:

- `path` - The path.

<log>

Logging settings.

<odl>

The ODL log entries are each written out in XML format in its respective log file. The log files have a maximum limit. When the limit is reached, the log files are overwritten.

When you enable ODL logging, each message goes into its respective log file, named `log`*N*`.xml`, where N is a number starting at one. The first log message starts the log file, `log1.xml`. When the log file size maximum is reached, the second log file is opened to continue the logging, `log2.xml`. When the last logfile is full, the first log file, `log1.xml` is erased and a new one is opened for the new messages. Thus, your log files are constantly rolling over and do not encroach on your disk space.

Attributes:

- `path`: Path and folder name of the log folder for this area. You can use an absolute path or a path relative to where the configuration XML file exists, which is normally in the `j2ee/home/config` directory. This denotes where the log files will reside for the feature that the XML configuration file is concerned with. For example, modifying this element in the `server.xml` file denotes where the server log files are written.
- `max-file-size`: The maximum size in KB of each individual log file.
- `max-directory-size`: The maximum size of the directory in KB. The default directory size is 10 MB.

New files are created within the directory, until the maximum directory size is reached. Each log file is equal to or less than the maximum specified in the attributes.

`<mail address="my@mail.address" />`

An e-mail address to log events to. A valid mail-session also needs to be specified if this option is used.

Attribute:

- `address` - The mail-address.

`<mail-session location="mail/TheSession"`
`smtp-host="smtp.server.com">`

The session SMTP-server host (if using SMTP).

Attributes:

- `location` - The location in the namespace to store the session at.
- `smtp-host` - The session SMTP-server host (if using SMTP).

`<namespace-access>`

Namespace (naming context) security policy for RMI clients.

<namespace-resource root="the/path">

A resource with a specific security setting.

Attribute:

- root - The root of the part of the namespaec that this rule applies to.

<password-manager>

Specifies the UserManager that is used for the lookup of hidden passwords. If omitted, the current UserManager is used for authentication and authorization. For example, you can use a OracleAS JAAS Provider LDAP UserManager for the overall UserManager, but use a OracleAS JAAS Provider XML UserManager for checking hidding passwords.

To identify a UserManager, provide a <jazn> element definition within this element, as follows:

```
<password-manager>
  <jazn ...>
</password-manager>
```

<persistence path="./persistence" />

A relative (to the application root) or absolute path to a directory where application state should be stored across restarts.

Attribute:

- path - The path (relative to the enterprise archive or absolute) to the persistence directory.

<principals path="principals.xml" />

Attribute:

- path - The path (relative to the enterprise archive or absolute) to the principals file.

<property name="theName" value="theValue" />

Contains a name/value pair initialization param.

Attributes:

- name - The name of the parameter.
- value - The value of the parameter.

<read-access>

The read-access policy.

<resource-provider>

Define a JMS resource provider. To add a custom <resource-provider>, add the following to your orion-application.xml file:

```
<resource-provider class="providerClassName" name="JNDI name">
  <description> description </description>
  <property name="name" value="value" />
</resource-provider>
```

In place of the user-replaceable constructs (those in italics) in the preceding code, do the following:

- Replace the value *providerClassName* of the `class` attribute with the name of the resource-provider class.
- Replace the value *JNDI name* of the `name` attribute with a name by which to identify the resource provider. This name will be used in finding the resource provider in the application's JNDI as "`java:comp/resource/name/`".
- Replace the value *description* of the `description` tag with a description of the specific resource provider.
- Replace the values *name* and *value* of the corresponding attributes with the same name in any property tags that the specific resource provider needs to be given as parameters.

<security-role-mapping impliesAll="true|false" name="theRole">

The runtime mapping (to groups and users) of a role. Maps to a security-role of the same name in the assembly descriptor.

Attributes:

- `impliesAll` - Whether or not this mapping implies all users. The default is false.
- `name` - The name of the role

<user name="theUser" />

A user that this security-role-mapping implies.

Attribute:

- `name` - The name of the user.

<user-manager class="com.name.of.TheUserManager" display-name="Friendly UserManager name">

Specifies an optional user-manager to use. For example, user-managers are `com.evermind.sql.DataSourceUserManager`, `com.evermind.ejb.EJBUserManager`, and so on. These are used to integrate existing systems and provide custom user-managers for Web applications.

Attributes:

- `class` - The fully qualified classname of the user-manager.
- `display-name` - A descriptive name for this `UserManager` instance.

<web-module id="myWebApp" path="myWebApp.war" />

A Web application module of the application. Each Web application can be installed on any site and in any context on those sites (for instance `http://www.myserver.com/myapp/`).

Attributes:

- `id` - The name used to reference this web-application when used in web-sites etc.
- `path` - The path (relative to the enterprise archive or absolute) to the web-application.

<write-access>

The write access policy.

Elements in the application-client.xml File

<application-client> Element Description of the application-client.xml file

The top level element of the `application-client.xml` file is the `<application-client>` element.

<application-client>

The `application-client` element is the root element of an application client deployment descriptor. The application client deployment descriptor describes the EJB components and external resources referenced by the application client.

Elements Contained Within <application-client> of the application-client.xml file

Within the `<application-client>` element, the following elements, which are listed alphabetically and not by DTD ordering, can be configured:

`<callback-handler>`

The `callback-handler` element names a class provided by the application. The class must have a no args constructor and must implement the `javax.security.auth.callback.CallbackHandler` interface. The class will be instantiated by the application client container and used by the container to collect authentication information from the user.

`<description>The description</description>`

A short description.

`<display-name>The name</display-name>`

The `display-name` element contains a short name that is intended to be displayed by tools.

`<ejb-link>EmployeeRecord</ejb-link>`

The `ejb-link` element is used in the `ejb-ref` element to specify that an EJB reference is linked to an enterprise bean in the encompassing J2EE Application package. The value of the `ejb-link` element must be the `ejb-name` of an enterprise bean in the same J2EE Application package.

`<ejb-ref>`

The `ejb-ref` element is used for the declaration of a reference to an enterprise bean's home. The declaration consists of an optional description; the EJB reference name used in the code of the referencing application client; the expected type of the referenced enterprise bean; the expected home and remote interfaces of the referenced enterprise bean; and an optional `ejb-link` information. The optional ejb-link element is used to specify the referenced enterprise bean.

`<ejb-ref-name>ejb/Payroll</ejb-ref-name>`

The `ejb-ref-name` element contains the name of an EJB reference. The EJB reference is an entry in the enterprise bean's environment. It is recommended that name is prefixed with "`ejb/`".

`<ejb-ref-type>Entity/Session</ejb-ref-type>`

The `ejb-ref-type` element contains the expected type of the referenced enterprise bean. The `ejb-ref-type` element must be one of the following: Entity Session

<env-entry>

The env-entry element contains the declaration of an Enterprise JavaBean's environment entries. The declaration consists of an optional description, the name of the environment entry, and an optional value.

<env-entry-name>minAmount</env-entry-name>

The env-entry-name element contains the name of an Enterprise JavaBean's environment entry.

<env-entry-type>java.lang.String</env-entry-type>

The env-entry-type element contains the fully-qualified Java type of the environment entry value that is expected by the enterprise bean's code. The following are the legal values of env-entry-type: java.lang.Boolean, java.lang.String, java.lang.Integer, java.lang.Double, java.lang.Byte, java.lang.Short, java.lang.Long, and java.lang.Float.

<env-entry-value>100.00</env-entry-value>

The env-entry-value element contains the value of an Enterprise JavaBean's environment entry.

<home>com.aardvark.payroll.PayrollHome</home>

The home element contains the fully-qualified name of the Enterprise JavaBean's home interface.

<icon>

The icon element contains a small-icon and large-icon element which specify the URIs for a small and a large GIF or JPEG icon image used to represent the application client in a GUI tool.

<large-icon>lib/images/employee-service-icon32x32.jpg</large-icon>

The large-icon element contains the name of a file containing a large (32 x 32) icon image. The file name is a relative path within the application client JAR file. The image must be either in the JPEG or GIF format, and the file name must end with the suffix ".jpg" or ".gif" respectively. The icon can be used by tools.

<remote>com.wombat.empl.EmployeeService</remote>

The remote element contains the fully-qualified name of the Enterprise JavaBean's remote interface.

<res-auth>Application/Container</res-auth>

The res-auth element specifies whether the Enterprise JavaBean code signs on programmatically to the resource manager, or whether the Container will sign on to the resource manager on behalf of the bean. In the latter case, the Container uses information that is supplied by the Deployer. The value of this element must be one of the two following: Application or Container

<resource-env-ref>

The resource-env-ref element contains a declaration of an application's reference to an administered object associated with a resource in the application's environment. It consists of an optional descrioption, the resource environment reference name, and an indication of the resource environment reference type expected by the application code.

<resource-env-ref-name>

The resource-env-ref-name element specifies the name of a resource environment entry name used in the application code.

<resource-env-ref-type>

The resource-env-ref-type element specifies the type of a resource environment reference.

<resource-ref>

The resource-ref element contains a declaration of Enterprise JavaBean's reference to an external resource. It consists of an optional description, the resource factory reference name, the indication of the resource factory type expected by the enterprise bean code, and the type of authentication (Bean or Container).

<res-ref-name>name</res-ref-name>

The res-ref-name element specifies the name of a resource factory reference.

<res-sharing-scope>Shareable</res-sharing-scope>

The res-sharing-scope element specifies whether connections obtained through the given resource manager connection factory reference can be shared. The value of this element, if specified, must be one of the following: Shareable or Unshareable. The default value is Shareable.

<res-type>javax.sql.DataSource</res-type>

The res-type element specifies the type of the data source. The type is specified by the Java interface (or class) expected to be implemented by the data source.

```
<small-icon>lib/images/employee-service-icon16x16.jpg
</small-icon>
```

The `small-icon` element contains the name of a file containing a small (16 x 16) icon image. The file name is a relative path within the application client JAR file. The image must be either in the JPEG or GIF format, and the file name must end with the suffix ".`jpg`" or ".`gif`" respectively. The icon can be used by tools.

Elements in the orion-application-client.xml File

<orion-application-client> Element Description

The top level element of the `orion-application-client.xml` file is the `<orion-application-client>` element.

`<orion-application-client>`

An `orion-application-client.xml` file contains the deploy time information for a J2EE application client. It complements the application client assembly information found in `application-client.xml`.

Elements Contained Within <orion-application-client>

Within the `<orion-application-client>` element, the following elements, which are listed alphabetically and not by DTD ordering, can be configured:

`<context-attribute name="name" value="value" />`

An attribute sent to the context. The only mandatory attribute in JNDI is the '`java.naming.factory.initial`,' which is the classname of the context factory implementation.

Attributes:

- `name` - The name of the attribute.
- `value` - The value of the attribute.

`<ejb-ref-mapping location="ejb/Payroll" name="ejb/Payroll" />`

The `ejb-ref` element is used for the declaration of a reference to another enterprise bean's home. The `ejb-ref-mapping` element ties this to a JNDI-location when deploying.

Attributes:

- `location` - The JNDI location to look up the EJB home from.

- `name` - The `ejb-ref` name. Matches the name of an ejb-ref in `application-client.xml`.

`<env-entry-mapping name="theName">deploymentValue</env-entry-mapping>`

Overrides the value of an `env-entry` in the assembly descriptor. It is used to keep the EAR (assembly) clean from deployment-specific values. The body is the value.

Attributes:

- `name` - The name of the context parameter.

`<lookup-context location="foreign/resource/location">`

The specification of an optional `javax.naming.Context` implementation used for retrieving the resource. This is useful when hooking up with third party modules, such as a third party JMS server for instance. Either use the context implementation supplied by the resource vendor or if none exists write an implementation which in turn negotiates with the vendor software.

Attributes:

- `location` - The name looked for in the foreign context when retrieving the resource.

`<resource-env-ref-mapping location="jdbc/TheDS" >`

The `resource-env-ref` element is used for the declaration of a reference to an external resource, such as a data source, JMS queue, mail session, or similar. The `resource-env-ref-mapping` ties that element to a JNDI location during deployment.

Attributes:

- `location` - The JNDI location to bind the resource to.

`<resource-ref-mapping location="jdbc/TheDS" name="jdbc/TheDSVar">`

The `resource-ref` element is used for the declaration of a reference to an external resource such as a data source, JMS queue, mail session or similar. The `resource-ref-mapping` ties this to a JNDI-location when deploying.

Attributes:

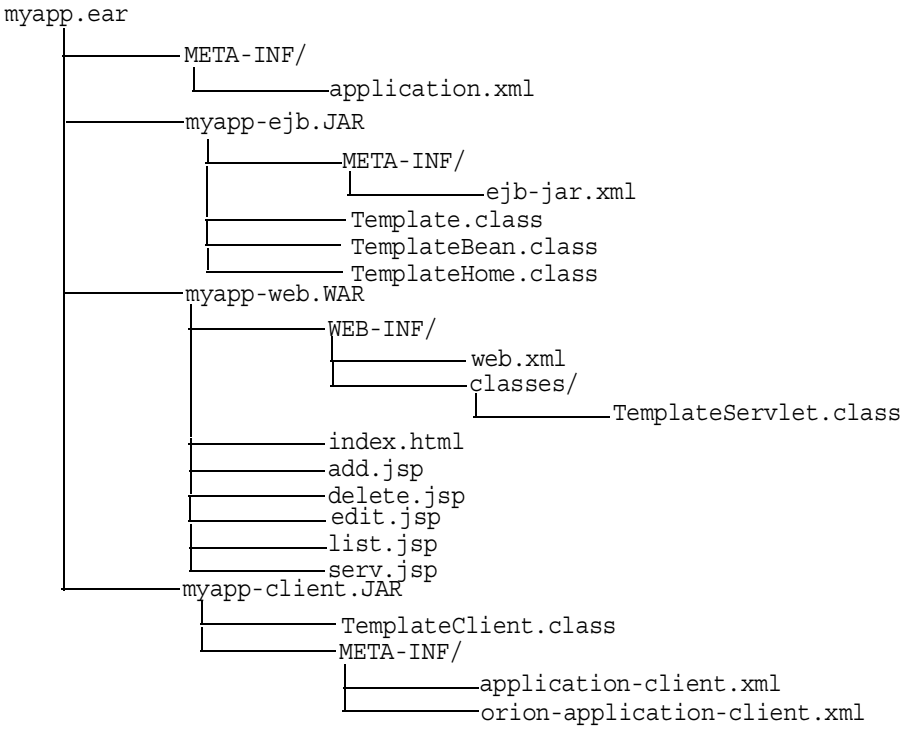
- `location` - The JNDI location to look up the resource home from.
- `name` - The `resource-ref` name. Matches the name of an `resource-ref` in `application-client.xml`.

Configuration and Deployment Examples

The following example shows how to configure and deploy a J2EE application within OC4J. See "Deploying Applications" on page 2-14 to learn how to modify the XML configuration files for the FAQ demo.

In this example, the `myapp` application contains a Java client, an EJB assembled into a JAR file, servlets and JSPs assembled into a WAR file, and an EAR file that contains both the EJB JAR file and the Web application WAR file. The tree structure showing the location of all the XML configuration files, the Java class files, and the JSP files is shown in Figure A–1. Notice that you can separate all the configuration files into logical directories within the application directory.

Figure A–1 Application EAR Structure



application.xml Example

The myapp/META-INF/application.xml file lists the EJB JAR and Web application WAR file that is contained in the EAR file using the <module> elements.

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE
Application 1.3//EN"
"http://java.sun.com/j2ee/dtds/application_1_3.dtd">
<application>
  <display-name>myapp j2ee application</display-name>
  <description>
     A sample J2EE application that uses a Container Managed
     Entity Bean and JSPs for a client.
  </description>
  <module>
    <ejb>myapp-ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>myapp-web.war</web-uri>
      <context-root>/myapp</context-root>
    </web>
  </module>
</application>
```

web.xml Example

The myapp/web/WEB-INF/web.xml file contains the class definitions for EJBs, servlets, and JSPs that are executed within the Web site. The myapp Web module specifies the following in its descriptor:

- The default page to be displayed for the application's root context (http://<host>:<port>/j2ee/myapp)
- Where to find the stubs for the EJB home and remote interfaces
- The JNDI name for the EJB
- The included servlets and where to find each servlet class
- How servlets are mapped to a subcontext using the <servlet-mapping> element (/template) off of the application root context

The Web server looks for the following:

- All servlet classes under `WEB-INF/classes/<package>.<class>`.
- All HTML and JSP from the root of the WAR file that is pointed to by `<web-app name="<warfile.war>">` in the `web-site.xml` file, which is packaged in the deployed corresponding application EAR file.
- OC4J compiles each JSP from `.java` into `.class` the first time it is used and caches it for subsequent use.

```
<web-app>
   <display-name>myapp web application</display-name>
   <description>
      Web module that contains an HTML welcome page, and 4 JSP's.
   </description>
   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>
   <ejb-ref>
      <ejb-ref-name>TemplateBean</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <home>TemplateHome</home>
      <remote>Template</remote>
   </ejb-ref>
   <servlet>
      <servlet-name>template</servlet-name>
      <servlet-class>TemplateServlet</servlet-class>
      <init-param>
         <param-name>length</param-name>
         <param-value>1</param-value>
      </init-param>
   </servlet>
</web-app>
```

ejb-jar.xml Example

The `ejb-jar.xml` file contains the definitions for a container-managed persistent EJB. The `myapp` EJB deployment descriptor contains the following:

- The entity bean uses container-managed persistence.
- The primary key is stored in a table. This descriptor defines the type and fields of the primary key.

- The table name is `TemplateBean`, and columns are named according to fields in the `ejb-jar.xml` descriptor and type mappings in `j2ee/home/config/database-schemas/oracle.xml`.
- The bean uses JDBC to access databases, as specified in `data-source.xml`, by `ejb-location` or by `default-data-source` in `orion-application.xml`.

```
<ejb-jar>
   <display-name>myapp</display-name>
   <description>
      An EJB app containing only one Container Managed Persistence
      Entity Bean
   </description>
   <enterprise-beans>
      <entity>
         <description>
            template bean populates a generic template table.
         </description>
         <display-name>TemplateBean</display-name>
         <ejb-name>TemplateBean</ejb-name>
         <home>TemplateHome</home>
         <remote>Template</remote>
         <ejb-class>TemplateBean</ejb-class>
         <persistence-type>Container</persistence-type>
         <prim-key-class>java.lang.Integer</prim-key-class>
         <reentrant>False</reentrant>
         <cmp-field><field-name>empNo</field-name></cmp-field>
         <cmp-field><field-name>empName</field-name></cmp-field>
         <cmp-field><field-name>salary</field-name></cmp-field>
         <primkey-field>empNo</primkey-field>
      </entity>
   </enterprise-beans>
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>TemplateBean</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>NotSupported</trans-attribute>
      </container-transaction>
      <security-role>
```

```
        <description>Users</description>
        <role-name>users</role-name>
      </security-role>
   </assembly-descriptor>
</ejb-jar>
```

server.xml Addition

When you deploy the application using the deployment wizard, this adds the location of the application EAR file to the `server.xml` file. This causes the application to be started every time that OC4J is started. If you do not want the application to be started with OC4J, change the `auto-start` variable to `FALSE`.

> **Note:** If you set `auto-start` to `FALSE`, you can manually start the application through Oracle Enterprise Manager or it is automatically started when a client requests the application.

```
<application name="myapp" path="../myapp/myapp.ear"
        auto-start="true" />
```

where

- The `name` variable is the name of the application.
- The `path` indicates the directory and filename for the EAR file.
- The `auto-start` variable indicates if this application should be automatically started each time OC4J is started.

default-web-site.xml Addition

The deployment wizard defines the root context for the Web application and binds the Web context and adds the following to the `default-web-site.xml` file:

```
<web-app application="myapp" name="myapp-web" root="/myapp" />
```

- The `name` variable is the name of the WAR file, without the `.WAR` extension.
- The `root` variable defines the root context for the application off of the Web site. For example, if you defined your Web site as "`http://<host>:7777/j2ee`", then to initiate the application, you would point your browser at "`http://<host>:7777/j2ee/myapp`".

Client Example

The application client that accesses the myapp application has a descriptor, which describes where to find the EJB stubs (home and remote interface) and its JNDI name.

The client XML configuration is contained in two files: application-client.xml and orion-application-client.xml.

The application-client.xml file contains a reference for an EJB, as follows:

```
<application-client>
    <display-name>TemplateBean</display-name>
    <ejb-ref>
         <ejb-ref-name>TemplateBean</ejb-ref-name>
         <ejb-ref-type>Entity</ejb-ref-type>
         <home>mTemplateHome</home>
         <remote>Template</remote>
    </ejb-ref>
</application-client>
```

The orion-application-client.xml file maps the EJB reference logical name to the JNDI name for the EJB. For example, this file maps the <ejb-ref-name> element, "TemplateBean," defined in the application-client.xml, to the JNDI name, "myapp/myapp-ejb/TemplateBean", as follows:

```
<orion-application-client>
  <ejb-ref-mapping name="TemplateBean"
location="myapp/myapp-ejb/TemplateBean" />
</orion-application-client>
```

JNDI Properties for the Client Set the JNDI properties for a regular client so it finds the initial JNDI context factory in one of the following manners:

- Set the JNDI properties within a Hashtable, then pass the properties to javax.naming.InitialContext.
- Set the JNDI properties within a jndi.properties file.

  If you provide the JNDI properties in the jndi.properties file, package the properties in myapp-client.jar to ensure that it is in the CLASSPATH.

  ```
  jndi.properties:
  ---------------
  java.naming.factory.initial=com.evermind.server.ApplicationClientInitialCont
  extFactory
  ```

```
java.naming.provider.url=
     opmn:ormi://<opmnhost>:<oc4j_instance>:7777/j2ee/myapp
java.naming.security.principal=admin
java.naming.security.credentials=welcome
```

Client Module—Standalone Java Client Invoking EJBs

Package your client module in a JAR file with the descriptor `META-INF/application-client.xml`.

Manifest File for the Client Package the client in a runable JAR with a manifest that has the main class to run and required `CLASSPATH`, as shown below. Check that the relative paths in this file are correct. Verify that you point to the relative location of the required OC4J class libraries.

```
manifest.mf
-----------
Manifest-Version: 1.0
Main-Class: myapp.myapp-client.TemplateClient
Name: "TemplateClient"
Created-By: 1.2 (Sun Microsystems Inc.)
Implementation-Vendor: "Oracle"
Class-Path: ../../../j2ee/home/oc4J.jar ../../../j2ee/home/jndi.jar
../../../j2ee/home/ejb.jar ../myapp-ejb.jar
```

Executing the Client To execute the client, perform the following:

```
% java -jar myapp-client.jar
TemplateClient.main(): start
Enter integer value for col_1: 1
Enter string value for col_2: BuyME
Enter float value for col_3: 99.9
Record added through bean
```

OC4J Command-Line Options and System Properties

You can set both system properties and command-line options on the OC4J command-line before startup. If OC4J is running, you must restart the instance for these to take effect. All system properties are prefaced with a `-D`. For example, `-Dhttp.session.debug`. All command-line options are prefaced with a hyphen (`-`). For example, `-help`.

- Table A–2 details OC4J command-line options.
- Table A–3 details general system properties.
- Table A–4 details debugging properties.

As described in "Configuring Server Properties" on page 3-2 and shown in Figure A–2, the `-D` system properties are entered on the Java Options line; the OC4J command-line options are entered on the OC4J Options line.

Figure A–2 EM Console to Modify Server Properties for an OC4J Instance

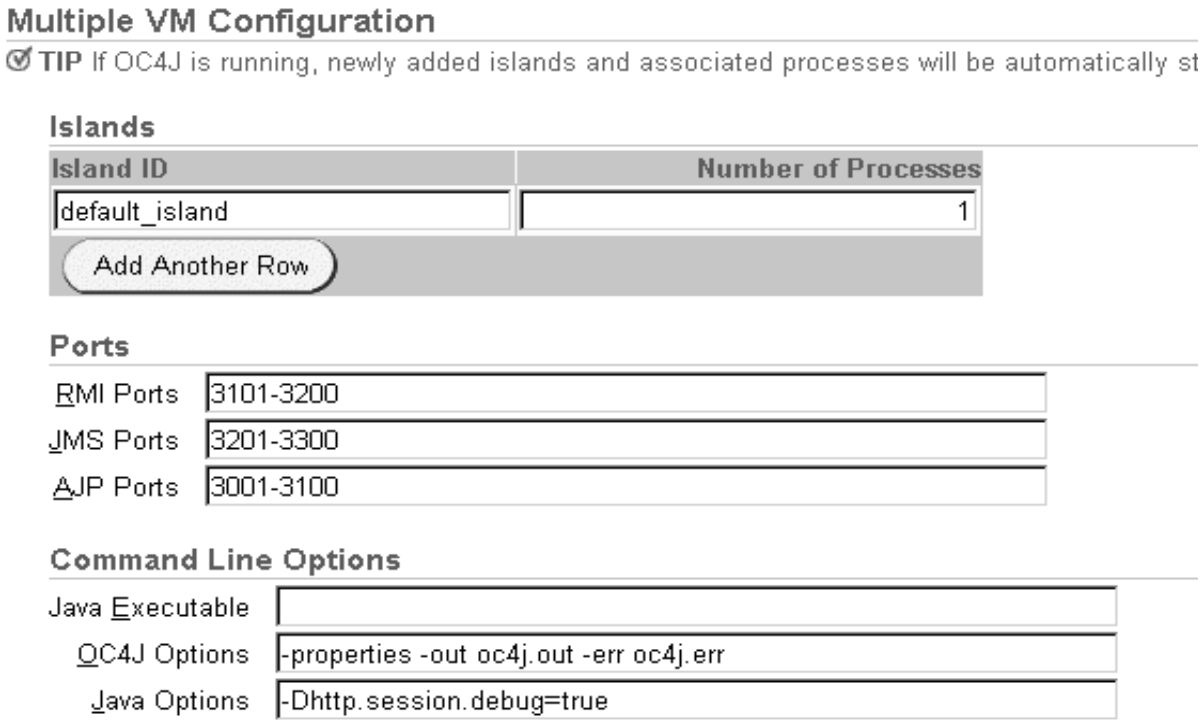


Table A–2 OC4J Command-Line Options

Command-Line Options	Description
`-install`	Installs the server, activates the Admin account, and rewrites text files to match the OS linefeed, and so on.
`-quiet`	Supress standard output.
`-config`	Specifies a location for the `server.xml` file.
`-rewriteXML`	Rewrites bad XML files (after prompting) as accurately as possible. Warning: If you have corrupt XML files, you may lose data when rewriting if they're badly misformed. Use this command with care.
`-out [file]`	Specifies a file to route standard output.
`-err [file]`	Specifies a file to route error output.

Table A–2 OC4J Command-Line Options (Cont.)

Command-Line Options	Description
`-monitorResourceThreads`	Enables backup debugging of thread resources. Enable this only if you have problems that relates to threads getting stuck in critical sections of code.
`-verbosity`	Define an integer between 1 and 10 to set the verbosity level of the message output. Example: `-verbosity 10`. See Example 3–5 for an example of this option.
`-version`	Prints the version and exits.
`-? -help`	Prints the help message.

Table A–3 -D General System Properties for OC4J

-D Option	Description
`java.home`	Sets the `JAVA_HOME` environment variable
`java.ext.dirs`	Sets the external directories to be searched for classes when compiling.
`java.io.tmpdir=<new_tmp_dir>`	Default is `/tmp/var`. To change the temporary directory for the deployment wizard. The deployment wizard uses 20 MB in swap space of the temp directory for storing information during the deployment process. At completion, the deployment wizard cleans up the temp directory of its additional files. However, if the wizard is interrupted, it may not have the time or opportunity to clean up the temp directory. Thus, you must clean up any additional deployment files from this directory yourself. If you do not, this directory may fill up, which will disable any further deployment. If you receive an `Out of Memory` error, check for space available in the temp directory.
`KeepIIOPCode=true/false`	Default is false. If true, keeps the generated IIOP stub / tie code.
`oracle.arraylist.deepCopy=true/false`	If true, then while cloning an array list, a deep copy is performed. If false, a shallow copy is performed for the array list. Default: true

Table A–3 -D General System Properties for OC4J (Cont.)

-D Option	Description
`dedicated.rmicontext= true/false`	Default is false. This replaces the deprecated `dedicated.connection` setting. When two or more clients in the same process retrieve an `InitialContext`, OC4J returns a cached context. Thus, each client receives the same `InitialContext`, which is assigned to the process. Server lookup, which results in server load balancing, happens only if the client retrieves its own `InitialContext`. If you set `dedicated.rmicontext=true`, then each client receives its own `InitialContext` instead of a shared context. When each client has its own `InitialContext`, then the clients can be load balanced. This parameter is for the client. You can also set this in the JNDI properties.
`associateUsingThirdTable= true/false`	For container-managed relationships in entity beans, you can designate if a third database table is used to manage the relationship. Set to false if you do not want a third association table. Default is true. See the "Entity Relationship Mapping" chapter in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for more information.

Table A–3 -D General System Properties for OC4J (Cont.)

-D Option	Description
`DefineColumnType=true/false`	`DefineColumnType=true/false`. The default is false. Set this to true if you are using an Oracle JDBC driver that is prior to 9.2. For these drivers, setting this variable to true avoids a round-trip when executing a select over the Oracle JDBC driver. This parameter should be set on the OC4J server. When you change the value of this option and restart OC4J, it is only valid for applications deployed after the change. Any applications deployed before the change are not affected. When true, the `DefineColumnType` extension saves a round trip to the database that would otherwise be necessary to describe the table. When the Oracle JDBC driver performs a query, it first uses a round trip to a database to determine the types that it should use for the columns of the result set. Then, when JDBC receives data from the query, it converts the data, as necessary, as it populates the result set. When you specify column types for a query with the `DefineColumnType` extension set to true, you avoid the first round trip to the Oracle database. The server, which is optimized to do so, performs any necessary type conversions.

Table A–3 -D General System Properties for OC4J (Cont.)

-D Option	Description
`oracle.mdb.fastUndeploy=<int>`	The `oracle.mdb.fastUndeploy` system property enables you to shutdown OC4J cleanly when you are running MDBs in a Windows environment or when the backend database is running on a Windows environment. Normally, when you use an MDB, it is blocked in a receive state waiting for incoming messages. However, if you shutdown OC4J while the MDB is in a wait state in a Windows environment, then the OC4J instance cannot be stopped and the applications are not undeployed since the MDB is blocked. However, you can modify the behavior of the MDB in this environment by setting the `oracle.mdb.fastUndeploy` system property. If you set this property to an integer, then when the MDB is not processing incoming messages and in a wait state, the OC4J container goes out to the database (requiring a database round-trip) and polls to see if the session is shut down. The integer denotes the number of seconds the system waits to poll the database. This can be expensive for performance. If you set this property to 60 (seconds), then every 60 seconds, OC4J is checking the database. If you do not set this property and you try to shutdown OC4J using CTRL-C, the OC4J process will hang for at least 2.5 hours.
`oracle.dms.sensors=[none, normal, heavy, all].`	You can set the value for Oracle Application Server built-in performance metrics to the following: none (off), normal (medium amount of metrics), heavy (high number of metrics), or all (every possible metric). The default is normal.This parameter should be set on the OC4J server. The previous method for turning on these performance metrics, `oracle.dms.gate=true/false`, is replaced by the `oracle.dms.sensors` variable. However, if you still use `oracle.dms.gate`, then setting this variable to false is equivalent to setting `oracle.dms.sensors=none`.

Table A–4 -D System Properties for Debugging

-D Debug System Properties	Description
ajp.debug	Default: false. If true, displays the AJP request (headers, mime types, URI etc) and response (status, error messages, and so on).
ajp.io.debug	Default: false. If true, displays the AJP post data, if any, and response data sent to the client.
KeepWrapperCode	Default: false. If true, keeps and debugs the generated wrapper code.
DBEntityHomeDebug	Default: false. If true, displays entity bean home interface debug messages.
DBEntityObjectDebug	Default: false. If true, displays entity bean object debug messages.
DBEntityWrapperDebug	Default: false. If true, displays entity bean pool debug messages.
iiop.runtime.debug	Default: false. If true, outputs IIOP debug messages.
NativeJDBCDebug	Default: false. Native JDBC debug messages.
http.cluster.debug	Default: false. HTTP clustering debug messages.
http.request.debug	Default: false. If true, provides information about each HTTP request to standard output.
http.redirect.debug	Default: false. If true, provides information about each HTTP redirects to standard output.
http.method.trace.allow	Default: false. If true, turns on the `trace` HTTP method.
http.session.debug	Default: false. If true, provides information about HTTP session events
http.error.debug	Default: false. If true, prints all HTTP errors
http.virtualdirectory.debug	Default:false. If true, print the enforced virtual directory mappings upon startup.
debug.http.contentLength	Default: false. If true, print explicit content-length calls as well as extra sendError information.
ejb.cluster.debug	Default: false. EJB clustering debug messages.
cluster.debug	Default: false. Clustering debug messages.
jms.debug	Default: false. JMS debug messages.
multicast.debug	Default: false. Multicast debug messages.
rmi.debug	Default: false. RMI debug messages.
transaction.debug	Default: false. If true, prints debug messages for JTA events.
rmi.verbose	Default: false. RMI verbose information.

Table A–4 -D System Properties for Debugging (Cont.)

-D Debug System Properties	Description
`datasource.verbose`	Default: false. If true, provides verbose information on creation of data source and connections using data sources and connections released to the pool, and so on,
`jdbc.debug`	Default: false. If true, provides very verbose information when JDBC calls are made
`ws.debug`	Default:false. If true, turns on OracleAS Web Services debugging
`javax.net.debug=[ssl|all]`	If ssl, turns on SSL debugging. If all, turns on SSL debugging with verbose messages.

For more information about debugging properties, see "OC4J Debugging" on page 3-52.

B

Third Party Licenses

This appendix includes a description of the Third Party Licenses for all the third party products included with Oracle Application Server.

Third-Party Licenses

Topics include:

- Apache HTTP Server
- Apache JServ

Apache HTTP Server

Under the terms of the Apache license, Oracle is required to provide the following notices. However, the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights. Notwithstanding anything to the contrary in the Oracle program license, the Apache software is provided by Oracle "AS IS" and without warranty or support of any kind from Oracle or Apache.

The Apache Software License

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
```

```
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
```

Apache JServ

Under the terms of the Apache license, Oracle is required to provide the following notices. However, the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights. Notwithstanding anything to the contrary in the Oracle program license, the Apache software is provided by Oracle "AS IS" and without warranty or support of any kind from Oracle or Apache.

Apache JServ Public License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- All advertising materials mentioning features or use of this software must display the following acknowledgment:

  This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (http://java.apache.org/).

- The names "Apache JServ", "Apache JServ Servlet Engine" and "Java Apache Project" must not be used to endorse or promote products derived from this software without prior written permission.
- Products derived from this software may not be called "Apache JServ" nor may "Apache" nor "Apache JServ" appear in their names without prior written permission of the Java Apache Project.
- Redistribution of any form whatsoever must retain the following acknowledgment:

  This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (http://java.apache.org/).

THIS SOFTWARE IS PROVIDED BY THE JAVA APACHE PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JAVA APACHE PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index

Symbols

A

B

C

D

E

M

N

O

P

R

S

T

U

W