Oracle® Application Server Containers for J2EE

Stand Alone User's Guide

10*g* (9.0.4)

Part No. B10323-01

September 2003

ORACLE

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

Part No. B10323-01

Primary Author: Sheryl Maring

Contributing Author: Brian Wright, Timothy Smith

Contents

2 Advanced Configuration, Development, and Deployment

3 Data Sources Primer

4 Servlet Primer

5 JSP Primer

6 EJB Primer

7 Security

A Additional Information

B Third Party Licenses

Index

Send Us Your Comments

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

Part No. B10323-01

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 Stand Alone 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) in standalone mode, 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 Stand Alone User's Guide* contains the following chapters and appendices:

Chapter 1, "Configuration and Deployment"

This chapter discusses how to install OC4J, how to configure the FAQ application, the popular J2EE demo application from Sun Microsystems, and how to deploy a Web application.

Chapter 2, "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 3, "Data Sources Primer"

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

Chapter 4, "Servlet Primer"

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

Chapter 5, "JSP Primer"

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

Chapter 6, "EJB Primer"

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

Chapter 7, "Security"

This chapter presents an overview of security features. It describes how to configure authorization and authentication for security.

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 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

Configuration and Deployment

This chapter demonstrates how to configure and execute OC4J as simply and quickly as possible. 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:

- Introduction to OC4J Standalone
- OC4J Installation
- Starting and Stopping OC4J
- Creating the Development Directory
- Configuring the FAQ Application Demo
- Deploying Applications
- Undeploying Web Applications

Introduction to OC4J Standalone

Oracle Application Server Containers for J2EE (OC4J) Standalone provides a complete Java 2 Enterprise Edition (J2EE) 1.3 environment written entirely in Java that executes on the Java virtual machine (JVM) of the standard Java Development Kit (JDK). You can run OC4J on the standard JDK that exists on your operating system or install it separately. Refer to the certification matrix on `http://otn.oracle.com`.

OC4J is J2EE 1.3 certified and provides all the containers, APIs, and services that J2EE specifies. OC4J is based on technology licensed from Ironflare Corporation, which develops the Orion server—one of the leading J2EE containers, so 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 OC4J 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	1.0
JAXP	1.1

OC4J Standalone is for use by development and small-medium scale production deployments. Specifically, OC4J Standalone supports HTTP and HTTPS natively without the use of Oracle HTTP Server. It does not have support for load balancing, clustering, or management through Oracle Enterprise Manager. To use those features, customers must install one of the Oracle Application Server installation

type, such as J2EE + WebCache. The standalone version is supported in a single instance, single JVM, and single machine configuration.

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 (1.3.1 or 1.4.1).
- You have installed OC4J.
- You have started OC4J.

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

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. After downloading the `oc4j_extended.zip` file from OTN, unzip this file to install OC4J. The following sections describe how to do this:

- Requirements
- Basic Installation
- Testing the Default Configuration

Requirements

You do not need 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 applications EAR, WAR, or `ejb-jar` files.

Basic Installation

OC4J is distributed within a ZIP file named `oc4j_extended.zip` on OTN. After unzipping this file, follow instructions listed in the `README.TXT`. Install this ZIP file in any directory that is in the path.

You must have a Java2 version Java executable in your $`PATH`, preferably version 1.3.1 or 1.4.1. To install OC4J, execute the following:

```
% cd /your_directory
% unzip oc4j_extended.zip
% cd j2ee/home
% java -jar oc4j.jar -install

Enter an administrator password
```

After the install is complete, the `j2ee/home` directory contains all the files necessary for running OC4J with a default configuration. The installation prompts you for an administration username and password, which is used for the administration console command-line tool.

Note: Instead of executing `oc4j.jar` from the `j2ee/home` directory, you can set a `$J2EE_HOME` variable (for UNIX) or the `%J2EE_HOME%` variable (for Windows NT) to `j2ee/home`, so that in the command line and execute `oc4j.jar` from *any* directory.

For example, in the UNIX environment use the following:

```
% java -jar $J2EE_HOME/oc4j.jar
```

Testing the Default Configuration

OC4J is installed with a default configuration that includes a default Web site and a default application. These are provided so you can start and test OC4J immediately.

Start OC4J by executing the following:

1. Change directory to the OC4J installation directory (`j2ee/home`), and issue one of the following commands:

   - `java -jar oc4j.jar`

     This starts OC4J using the default configuration files, which are located in `j2ee/home/config`.

   - `java -jar oc4j.jar -config /mypath/server.xml`

     This starts OC4J using the `server.xml` file located in `/mypath`.

   The server should output an initialization string with the version number.

2. Test OC4J by accessing "`http://`*`oc4j_host`*`:8888/`" from a Web browser. If you changed the default port number, access the Web server using "`http://`*`oc4j_host:oc4j_port`*`/`".

For example, test the Web server by connecting a Web browser to `http://oc4j_host:8888/servlet/HelloWorldServlet`, which should return a "`Hello World`" page.

For more information on starting and stopping OC4J, see "Starting and Stopping OC4J" on page 1-5. For more information on configuration, see "Deploying Applications" on page 1-19.

Starting and Stopping OC4J

- Starting OC4J
- Administering OC4J
- Shutting Down OC4J

Starting OC4J

OC4J is installed with a default configuration that includes a default Web site and a default application. Therefore, you can start OC4J immediately. To start OC4J in a standalone environment, issue the following command from the `j2ee/home/` directory:

```
java -jar oc4j.jar options
```

This command starts OC4J using the default configuration files, which you can find in the `j2ee/home/config` directory.

Options for this command are not necessary to start OC4J. However, if you want to exercise more control, use the options listed in "Options for the OC4J Server JAR" on page A-33 or issue the following command from the `j2ee/home` directory:

```
java -jar oc4j.jar -help
```

After OC4J launches, a message is displayed on the screen to note this fact.

Note: Instead of executing `oc4j.jar` from the `j2ee/home` directory, you can set a `$J2EE_HOME` variable (for UNIX) or the `%J2EE_HOME%` variable (for Windows NT) to `j2ee/home`, so that in the command line and execute `oc4j.jar` from *any* directory.

For example, in the UNIX environment use the following:

```
% java -jar $J2EE_HOME/oc4j.jar
```

Administering OC4J

After starting the OC4J server, you can administer the server using the `admin.jar` command-line tool, which is located in `<install_directory>/j2ee/home`. To use the `admin.jar` command, see the following syntax:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password options
```

where the variables are as follows:

- *oc4j_host:oc4j_ormi_port*—The host name and port of the OC4J server from which you want to deploy the application. The `admin.jar` tool uses the OC4J Remote Method Invocation (ORMI) protocol to communicate with the OC4J server. Therefore, the host name and port identified by these variables are defined in the `rmi.xml` file for the OC4J server to which you are directing the request.

  The default port number for the ORMI protocol is 23791. Configure both the host name and port number—if not using the default—in the `rmi.xml` file in the `<rmi-server>` element, as follows:

  ```
  <rmi-server port="oc4j_ormi_port" host="oc4j_host">
  ```

- *admin_id admin_password*—The administration identity and password. Specify this identity and password for the OC4J server in its `principals.xml` file.

"Options for the OC4J Administration Management JAR" on page 1-3 discusses the options for this tool.

Restarting OC4J

You can designate whether the task manager in OC4J automatically detects changes made to deployed applications. Once a change is detected, then OC4J reloads these

applications automatically. In this case, you do not need to restart the server when redeploying an application.

The `check-for-updates` attribute in the `<application-server>` element in the `server.xml` file defaults to true, which enables automatic deployment. If true, task manager checks for XML configuration file modifications. Thus, if you set this to false, you can disable automatic refreshing of the configuration to any new XML modifications. Also, setting this attribute to false stops the automatic deployment of any applications until you execute `admin.jar -updateConfig`. If set to false, you cause the XML configuration to refresh from the XML files and any necessary automatic deployment to occur by using the `admin.jar -updateConfig` option.

If you enable automatic deployment, then you do not have to restart the OC4J process each time you make a modification to the application. However, enabling automatic deployment also effects your performance. Thus, it is recommended that you enable automatic deployment only in a development environment, not in a production environment.

Even if you have automatic deployment enabled, it does not detect modifications in the global server XML configuration files. Thus, if you modify any of the container-level configuration files, such as `data-sources.xml`, `rmi.xml`, or `principals.xml`, you must restart the OC4J process for these modifications to be recognized.

To restart OC4J using the default parameters, change to the installation root directory, and execute the following:

```
% java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin
    admin_password -restart
```

This command connects to the OC4J RMI listener port and requests it to restart. It may not work if the JVM is not responding to signals or accepting RMI messages. In this case, stop the JVM in the UNIX environment with the following operating system command: `kill` *`process`*. In the Windows NT environment, access the Windows NT Task Manager to terminate the JVM.

Shutting Down OC4J

Shut down OC4J by executing the following:

```
% java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin
    admin-password -shutdown
```

This command provides a graceful shutdown of the container. If it does not shut down the container, force a rapid shutdown by passing the `force` argument, as follows:

```
% java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin
    admin-password -shutdown force
```

If this method does not work, then kill OC4J processes with your operating system commands or tool, depending on your system.

HTTP and RMI Communication

For HTTP applications, clients can send their requests directly to OC4J. The default port number is 8888. You can change this port number in the appropriate `*-web-site.xml` file, such as the `http-web-site.xml` file.

For RMI-based applications—such as EJB and JMS—clients should send their requests directly to OC4J. The default RMI port is 23791. Modify this port number in the `rmi.xml` file. See "Configuring a Listener" on page 2-10 for directions.

Quick Start for JSPs and Servlets

To deploy Web applications on OC4J, do one of the following:

- Place your servlet classes and JSP pages in the `j2ee/home/default-web-app` directory.
- Deploy J2EE applications using the `admin.jar` tool. The J2EE application must be archived in the EAR format.

Placing servlets and JSP pages in the `default-web-app` directory is the easiest method to deploy applications or to migrate J2EE applications from previous versions of OC4J.

Do the following for quick deployment of servlets or JSPs:

1. Place your servlet classes in the `j2ee/home/default-web-app/WEB-INF/classes` subdirectory—in a directory corresponding to their Java package. The servlet is accessible from URLs of the form: `http://oc4j_host:8888/servlet/class-name`

   For example, place the servlet class `my.HelloServlet`, as follows:

   ```
   j2ee/home/default-web-app/Web-INF/classes/my/HelloServlet.class
   ```

   Then it is accessible from the following URL:

```
http://oc4j_host:8888/servlet/my.HelloServlet
```

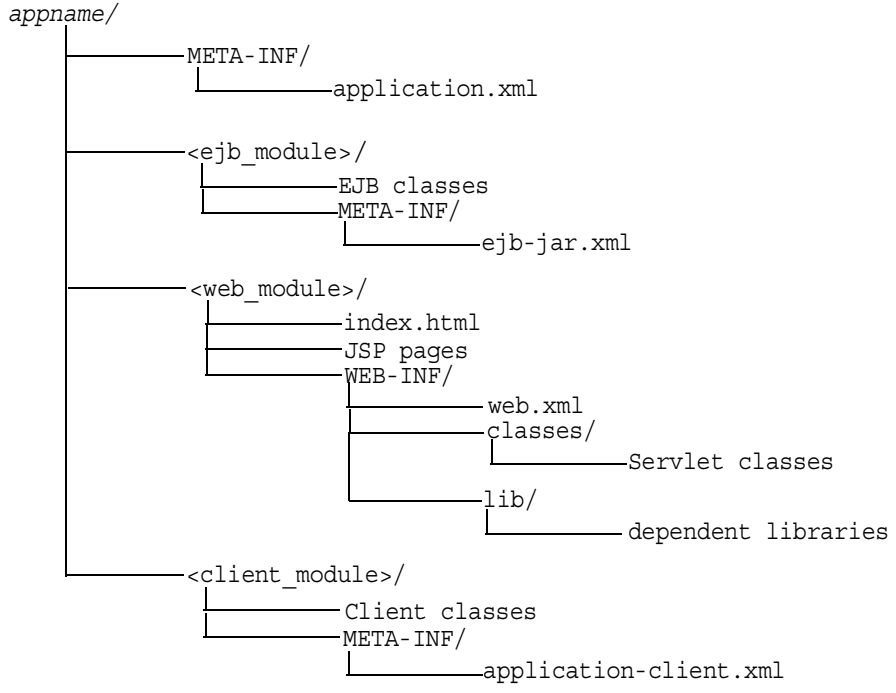
2. Place JSP pages anywhere in the `j2ee/home/default-web-app` directory. They are accessible with URLs of the form:
   `http://oc4j_host:8888/path-to-JSP`

   For example, a JSP page in `j2ee/home/default-web-app/examples/Hello.jsp` is accessible as `http://oc4j_host:8888/examples/Hello.jsp`.

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 1–1 demonstrates this structure.

Figure 1–1 Development Application Directory Structure



Consider the following points regarding Figure 1–1:

- 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. 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

Before you configure OC4J and deploy the FAQ demo, modify the back-end database to contain tables that are necessary for executing the FAQ demo.

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.

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 3, "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. 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. Deploy the FAQ application using by either copying the EAR file to the `j2ee/home/applications` directory or by the `admin.jar` tool. The following sections explain each method.
3. Start the OC4J server by executing `java -jar oc4j.jar`
4. Execute the FAQ application in your browser, where the default port is 8888.

   `http://`*`oc4j_host:8888`*`/FAQApp`

Deploy Using Automatic Deployment in a Development Environment

As discussed in "Restarting OC4J" on page 1-6, OC4J supports automatic deployment and redeployment of applications, which allows you to make changes to the application EAR file, which are picked up by the server without stopping and restarting OC4J. You enable this through the `check-for-updates` attribute in the `server.xml` file.

When automatic deployment is enabled, simply modify the XML configuration files, rearchive the application with its XML files into an EAR file, and copy the EAR file to the applications directory. The OC4J server notices the modified date and will redeploy the application, as necessary.

Warning: Automatic deployment should only be used in a development environment. The task manager that checks for updates can be time consuming. Turn off automatic deployment in a production environment by setting the `check-for-updates` attribute to false.

For the first deployment of the FAQ application (locally), do the following:

1. Copy the `<FAQApp_Home>/faq/dist/FAQApp.ear` file to the `j2ee/home/applications` directory.
2. Modify the `j2ee/home/config/server.xml` and `http-web-site.xml` files to register the FAQ application in the `j2ee/home/applications` directory, as follows:
   a. In the `j2ee/home/config/server.xml` file, add the `FAQApp` entry, as follows:

      ```
      <application name="FAQApp" path="../applications/FAQApp.ear" />
      ```

      This step deploys the FAQ application on OC4J. The path is relative to `j2ee/home/config`. Since the `FAQApp.ear` file is in `j2ee/home/applications`, this makes the path `../applications/FAQApp.ear`.

      For full details on the `server.xml` configuration file, see "Elements in the server.xml File" on page A-8.

   b. In the `j2ee/home/config/http-web-site.xml` file, bind the FAQ Web application by adding the `FAQApp` entry, as follows:

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

      This step makes FAQ accessible from the `/FAQApp` URL on the OC4J server.

      For full details on the `http-web-site.xml` configuration file, see the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*.

For more information, see "Manually Adding Applications in a Development Environment" on page 2-10, "OC4J Automatic Deployment for Applications" on page 2-15, and "What Happens When You Deploy?" on page 2-8.

Deploy Using the Admin.JAR Tool in All Environments

In the production environment, you should set the `check-for-updates` attribute to false (see "Restarting OC4J" on page 1-6) and then use the `admin.jar` tool for deploying all applications. The `admin.jar` tool deploys the application and modifies all of the appropriate XML files. This provides for remote deployment.

Use the `admin.jar` command-line tool for registration and deployment, as follows:

```
java -jar admin.jar
      ormi://oc4j_host:oc4j_ormi_port
      admin welcome -deploy
      -file d:\j2ee\home\FAQAPP_Home\faq\dist\FAQApp.ear
      -deploymentName FAQApp
```

The default port number for `oc4j_ormi_port` is 23791.

This step creates the entry in the `server.xml` file for the FAQ application. For a complete description of the `admin.jar` command-line tool, see "Options for the OC4J Administration Management JAR" on page A-34.

You can bind any Web application through the `admin.jar` tool, as follows:

```
java -jar admin.jar
   ormi://oc4j_host:23791
   admin welcome -bindWebApp
   FAQApp FAQAppWeb http_web_site /FAQApp
```

This creates the `<web-app>` entry in the `http-web-site.xml` configuration file. For a complete description of the `admin.jar` command-line tool, see "Options for the OC4J Administration Management JAR" on page A-34.

For more information on configuring and managing Web applications, see the `http-web-site.xml` file, see the *Oracle Application Server Containers for J2EE Servlet Developer's Guide*.

Deployment Details Explained

Although the development of J2EE applications is standardized and portable, the XML configuration files are not. You may have to configure multiple XML files before deploying any application to OC4J. 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 in the `data-sources.xml` file.

For basic applications, such as the FAQ demo, you configure the following OC4J XML files:

- `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.
- `server.xml` and `http-web-site.xml`—The application is registered in the `server.xml` file; the Web application and the context it uses are registered in the `http-web-site.xml` file (or any other `*-web-site.xml` file that you choose).
- `data-sources.xml`—You must configure the `DataSource` object in the `data-sources.xml` file for each database used within the application.

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 application XML configuration files.	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.
4. 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.
5. 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.
6. Register the application in the appropriate server XML files.	Modify the `server.xml` and `http-web-site.xml` files or use the `admin.jar` tool, which will modify these files for you.
7. Configure the database used.	Modify the `data-sources.xml` file.

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

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. All of the application XML files, such as `web.xml`, are provided for you in the ZIP file, correctly configured.
4. Update the standard J2EE application descriptor file. The `application.xml` in the FAQ demo application is provided for you in the ZIP file. OC4J uses the `application.xml` file as the standard J2EE application descriptor file.
5. Build an EAR file including the application. You can modify the FAQ demo application and rebuild it using the ANT command. To rebuild and deploy the FAQ demo, execute the following:

   ```
   ant deploy
   ```

   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/
   ```

   If you do not want to rebuild, you can copy the `FAQApp.ear` from the ZIP file into `j2ee/home/applications`. This step places the FAQ application in the OC4J server.
6. 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.
7. Register the J2EE application in the `server.xml` file and its Web application in the `http-web-site.xml` or use the `admin.jar` tool to deploy, which will modify these files for you.
8. Start OC4J by executing the following command from the `j2ee/home/` directory:

```
java -jar oc4j.jar
```

For a complete description of all the OC4J starting options, see "Starting OC4J" on page 1-5.

Open your Web browser and then specify the following URL:

`http://`*`oc4j_host`*`:8888/FAQApp`

See "Overview of OC4J and J2EE XML Files" on page 2-2 for more information on OC4J XML configuration files.

Deploying Applications

This section describes how to deploy a J2EE application to the OC4J server and how to bind that application to the server so that you can access the application from OC4J.

- Archive Application into an EAR File
- Deployment In a Production Environment Using ADMIN.JAR
- Deploy your Application Manually in a Development Environment
- Verifying Deployment

Archive Application into an EAR File

Your J2EE application can contain the following modules:

- Web applications

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

- EJB applications

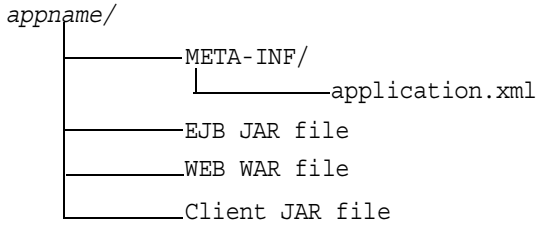
  The EJB applications module (EJB JAR files) includes Enterprise JavaBeans (EJBs).

- 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 1–2 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.

Deployment In a Production Environment Using ADMIN.JAR

OC4J contains a command-line deployment tool for deploying J2EE applications—the `admin.jar` command. The options for this command are listed in "Standalone OC4J Command-Line Options and Properties" on page A-33. Ensure that automatic deployment is disabled by setting the `check-for-updates` attribute to false (see "Restarting OC4J" on page 1-6).

To deploy a J2EE application with the EAR file to a remote node, execute `admin.jar`, as follows:

```
java -jar admin.jar ormi://host:port
             username password
             -deploy
                 -file filename -deploymentName app_name
                 -targetPath path/destination
```

where

- The *`host:port`* is the host and port of the OC4J server.
- The *`username password`* is the administration username and password for the OC4J server.
- The `-file` *`path/filename`* indicates the local directory and filename for the EAR file.

- The `-deploymentName` *`app_name`* variable is a user-defined name of the application.
- The `-targetPath` *`path/destination`* indicates what path on the server node in which to deploy the EAR file. Provide a target path to the directory where the EAR file is copied for deployment. The default path is the `applications/` directory. Oracle recommends that you provide a target path.

> **Note:** If you have a Web application within the EAR file, bind the Web application using the `admin.jar -bindWebApp` option.

This deployment step creates a new entry in `server.xml` for the application, as follows:

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

where

- The `name` attribute is the name of the application.
- The `path` indicates the directory and filename for the EAR file.
- The `auto-start` attribute 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.

Binding the Web Application in a Production Environment

To make your J2EE Web application accessible from the OC4J Web server, bind the Web application to the OC4J server using the `-bindWebApp` option as follows:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port
    username password
    -bindWebApp app_name web_app_name web_site_name context_root
```

where the following are the values for `-bindWebApp`:

- *`app_name`* is the application name, which is the same name used in `-deploymentName` on the `-deploy` option. In addition, note that this is the same name that is saved in the `<application name=`*`app_name`*` />` attribute in the `server.xml` file.

- *web_app_name* is the name of the WAR file contained within the EAR file—without the `.war` extension.
- *web_site_name* is the name of the `*-web-site.xml` file that denotes the Web site to which this Web application should be bound. This is the file that will receive the Web application definition.
- *context_root* is the root context for the Web module. The Web context defines how the Web application is accessed.

This step creates an entry in the OC4J `*-web-site.xml` configuration file that is denoted in the *web_site_name* variable. For a listing of all the options for `admin.jar`, see "Options for the OC4J Administration Management JAR" on page A-34.

Deploy your Application Manually in a Development Environment

To deploy your application in a development environment, you can modify the XML files by hand. Ensure that automatic deployment is enabled by setting `check-for-updates` attribute to true (see "Restarting OC4J" on page 1-6).

In `server.xml`, add a new or modify the existing `<application name=... path=... auto-start="true" />` entry for each J2EE application. The path should be the full directory path and EAR filename. For our employee example, add the following to the `server.xml` file:

```
<application name="employee"
          path="/private/applications/Employee.ear"
          auto-start="true" />
```

If you included a Web application portion, you must do the following to bind the Web application to the Web server. In `*-web-site.xml`, add a `<web-app ...>` entry for each Web application. The `application` attribute should be the same value as provided in the `server.xml` file. The `name` attribute should be the WAR file, without the WAR extension, for the Web application.

For Web application binding for the `employee` Web application, add the following:

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

Verifying Deployment

OC4J detects the addition of your application to `server.xml`. The OC4J server displays a message that your application has been deployed. After the message is displayed, you can invoke requests against your application.

Undeploying Web Applications

You can remove a J2EE Web application from the OC4J Web server using the `-undeploy` option with the `admin.jar` command-line tool. The syntax is as follows:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port
     admin adminpassword
    -undeploy applicationName -keepFiles
```

This command removes the deployed J2EE application known as `applicationName` 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. If you do not want these files to be removed, use the `-keepFiles` switch.

2

Advanced Configuration, Development, and Deployment

This chapter provides information for administering OC4J in standalone mode for development purposes. Chapter 1, "Configuration and Deployment", discusses the easiest method for configuring, developing, and deploying a J2EE application. However, if you want to use other services, such as JMS, you must know how to manipulate the XML configuration files.

This chapter discusses the following topics:

- Overview of OC4J and J2EE XML Files
- What Happens When You Deploy?
- Sharing Libraries
- Manually Adding Applications in a Development Environment
- Building and Deploying Within a Directory
- OC4J Automatic Deployment for Applications
- Changing XML Files After Deployment
- Designating a Parent of Your Application
- Developing Startup and Shutdown Classes
- Setting Performance Options
- Enabling OC4J Logging
- OC4J Debugging

Overview of OC4J and J2EE XML Files

This section contains the following topics:

- XML Configuration File Overview
- XML File Interrelationships

XML Configuration File Overview

Because OC4J is configured solely through XML files, you must understand the role and method for a set of XML files. Each XML file 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 2–1 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.

  OC4J server configuration files exist under the `j2ee/home/config/` directory — 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.

- 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:

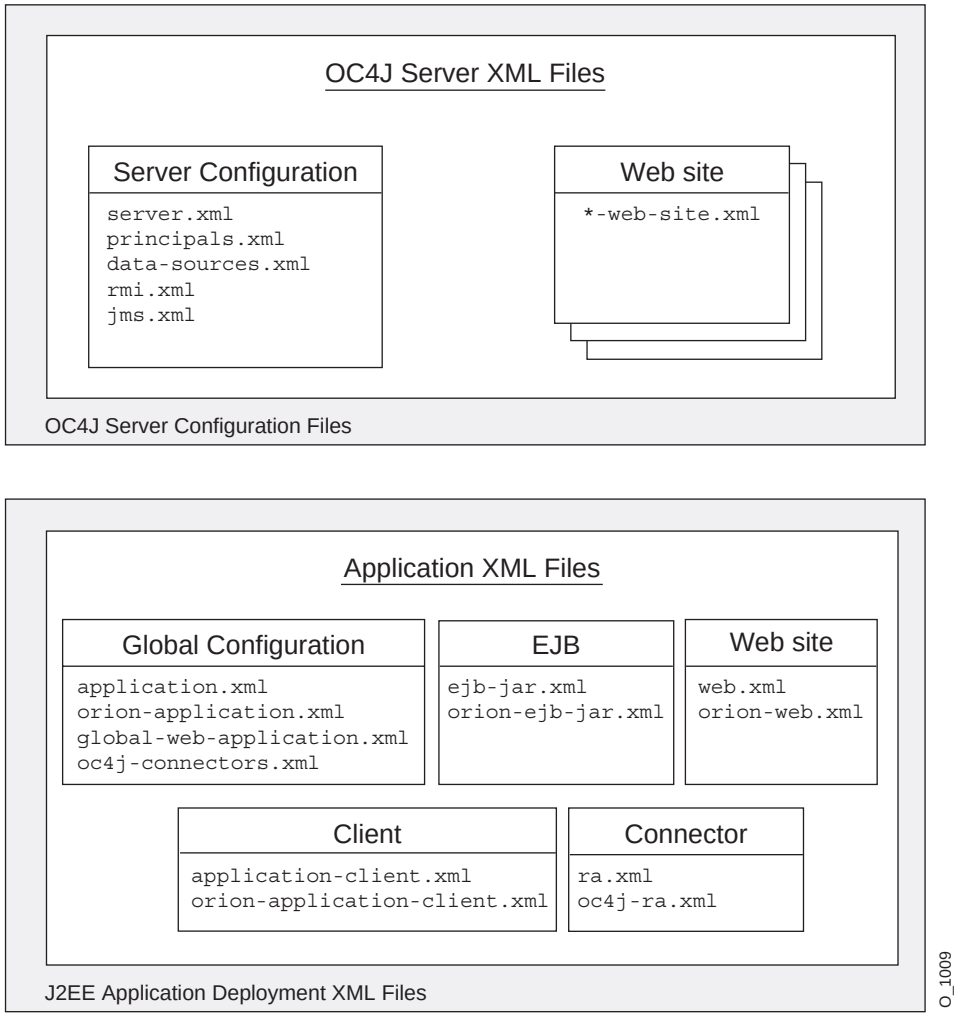
  – The `application.xml` as the global application configuration file that contains common settings for all applications in this OC4J instance.

  – 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 2–1 OC4J and J2EE Application Files



Note: Each deployed application uses an `application.xml` as the standard J2EE application descriptor file. That XML file is local to the application and separate from the `application.xml` that exists in the `j2ee/home/config` directory. The `j2ee/home/config/application.xml` file configures options that are applied to all applications deployed in this OC4J server instance.

Table 2–1 describes the role and function for each XML file that was displayed in the preceding figure.

Table 2–1 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.
`principals.xml`	OC4J security configuration for the type of 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.
`*-web-site.xml`	OC4J Web site definition. Each Web site is defined within its own XML file. It is a good practice to name each XML file based on the root element name, `<web-site>`. For example, `*-web-site.xml` could be `my-web-site.xml`. Normally, the global Web site definition is in `http-web-site.xml`. You must specify each Web site XML file in its own `web-site path` statement contained within the `server.xml` file.

Table 2–1 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—principals.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 2–1 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 2–3).

Here are the interrelated files:

- `server.xml`—contains references to the following:
  - All `*-web-site` files for each Web site for this OC4J server, including the default `http-web-site.xml` file.
  - The location of each of the other OC4J server configuration files, except `principals.xml`, which is defined in the global `application.xml`, shown in Figure 2–1
  - The location of each `application.xml` file for each J2EE application that has been deployed in OC4J
- `http-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 a reference to the `principals.xml` file.

The `server.xml` file is the keystone that contains references to most of the files used within the OC4J server. Figure 2–2 shows the XML files that can be referenced in the `server.xml` file:

Figure 2–2 XML Files Referenced Within server.xml

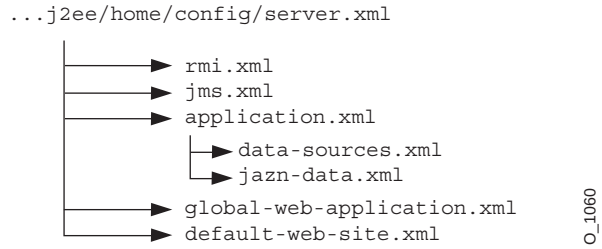
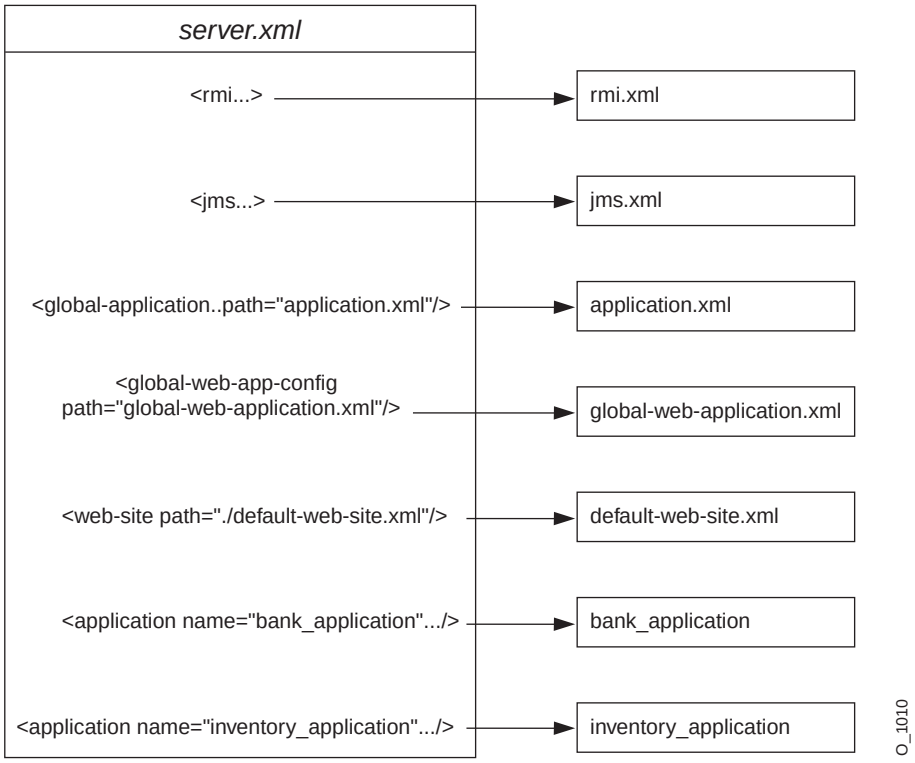


Figure 2–3 demonstrates how the `server.xml` points to other XML configuration files. For each XML file, the location can be the full path or a path relative to the location of where the `server.xml` file exists. In addition, the name of the XML file can be any name, as long as the contents of the file conform to the appropriate DTD.

- The `<rmi-config>` element denotes the name and location of the `rmi.xml` file.
- The `<jms-config>` element denotes the name and location of the `jms.xml` file.
- The `<global-application>` element denotes the name and location of the global `application.xml` file.
- The `<global-web-app-config>` element denotes the name and location of the `global-web-application.xml` file.
- The `<web-site>` element 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. You can deploy applications through the `admin.jar` command using the `-deploy` option or by modifying the `server.xml` file directly. Each deployed application is denoted by the `<application>` element. See "Manually Adding Applications in a Development Environment" on page 2-10 for more information on directly editing the `server.xml` file.

Figure 2–3 Server.xml File and Related XML Files



Other elements for `server.xml` are described in "Elements in the server.xml File" on page A-8.

What Happens When You Deploy?

Whether you deploy the application through the `admin.jar` command or by editing XML files, 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, OC4J writes out the new module deployment descriptors 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. However, you can designate where the "master" directory is by the `admin.jar -targetPath` option.

   Note: If you deploy this EAR file using `admin.jar` 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.

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.

Manually Adding Applications in a Development Environment

When you are in a development environment, it is sometimes easier to modify XML files than to use the admin.jar command for each iteration of development. The following sections help you understand how to modify your XML configuration files:

- Configuring a Listener
- Configuring J2EE Applications

Configuring a Listener

Each OC4J server is configured to listen on HTTP or RMI protocols for incoming requests. Each OC4J Web server is configured within its own *-web-site.xml file.

- HTTP protocol listener—HTTP clients can access an OC4J HTTP listener directly. This involves configuring an `http-web-site.xml` file, which indicates the HTTP listener port. The default HTTP port is 8888. The following shows the entry in the `http-web-site.xml` for an HTTP listener with a port number of 8888:

  ```
  <web-site host="oc4j_host" port="8888" protocol="http"
      display-name="Default OC4J WebSite">
  ```

- RMI protocol listener—EJB clients and the OC4J tools, such as `admin.jar`, access the OC4J server through a configured RMI port. This involves configuring the `rmi.xml` file. The default RMI port is 23791. The following shows the default RMI port number configured in the `rmi.xml` file:

  ```
  <rmi-server port="23791" >
  ```

Configuring J2EE Applications

To configure and deploy your J2EE applications, modify the `server.xml` and `http-web-site.xml` files with your application information.

- In `server.xml`, add a new or modify the existing `<application name=... path=... auto-start="true" />` entry for each application that you want automatically started when OC4J starts. The path points to either the location of the EAR file to be deployed or the exploded directory where the application has been built. See "Deployment In a Production Environment Using ADMIN.JAR" on page 1-20 or "Building and Deploying Within a Directory" on page 2-12 for more information.
- In `http-web-site.xml`, add a `<web-app...>` entry for each Web application you want bound to the Web site upon OC4J startup. Because the `name` attribute is the WAR filename (without the `.war` extension), you must have one line for each WAR file included in your J2EE application.

  For Web application binding using a WAR file, add the following:

  ```
  <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://oc4j_host:8888`", then to initiate the application, point your browser at "`http://oc4j_host:8888/myapp`".

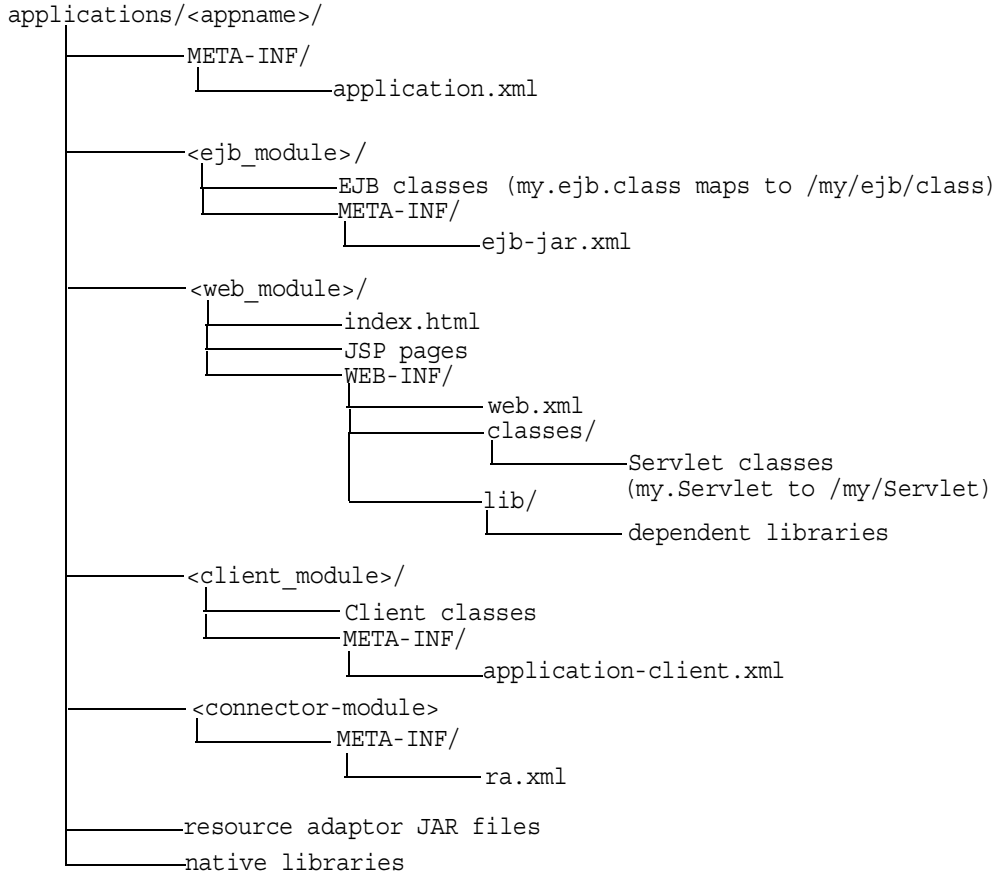
Note: Wait for automatic startup to complete before trying to access the client. The client fails on lookup if it tries to access before the completion of these processes.

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 2–4, 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 2–4 displays the necessary directory structure.

Figure 2–4 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 2–4 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 2–4 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 2–4, 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 2–4, 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 2–4, 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 2–4 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 an EAR file. During execution, the entire EAR file is loaded into memory and indexed. This is faster than reading in each class from the development directory when necessary.

OC4J Automatic Deployment for Applications

OC4J automatically deploys an application if the timestamp on an EAR file has changed. Restarting OC4J to deploy or redeploy applications is not necessary. Automatic deployment is not enabled in all cases, but deployment occurs in the following cases:

- changes to EAR files are checked

  If you change the EAR file, OC4J automatically detects the change. OC4J detects the timestamp change and redeploys the application.

- change in timestamp of certain XML files in the exploded directory format (The *appname* directory) that is discussed in "Building and Deploying Within a Directory" on page 2-12. For automatic deployment of exploded directory applications, you must do the following:

  1. Modify the classes in the `<module>` and touch its J2EE deployment descriptor to change the timestamp on the XML file. For example, if you

modify servlet classes, you must touch its `web.xml` file. This notifies OC4J that changes occurred in this `<module>`.

2. Touch the `application.xml` of this application. Changing the timestamp of the `application.xml` starts the automatic deployment. Once started, OC4J checks which modules to redeploy by noticing which module deployment descriptors have timestamp changes.

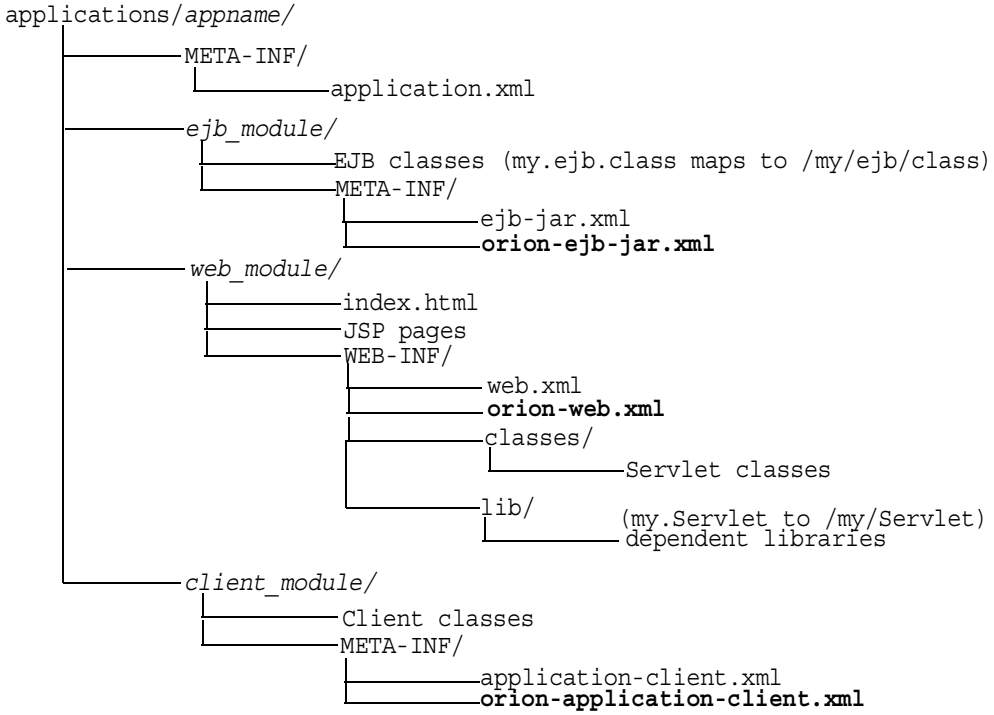
When OC4J does not check for updates, redeploy by either using the `admin.jar` command-line tool or restarting the OC4J server manually. See "Options for the OC4J Administration Management JAR" on page A-34 for a description of the `-deploy` option.

Changing XML Files After Deployment

Whenever you deploy an application, OC4J automatically generates the OC4J-specific XML files with the default elements. If you want to change these files or add to the existing XML files, you must copy the XML files to where your original development directory for the application and change it in this location. If you change the XML file within the deployed location, OC4J simply overwrites these changes when the application is deployed again. The changes only stay constant when changed in the development directories.

For all OC4J-specific XML files, you can add these files within the recommended development structure as shown in Figure 2–5.

Figure 2–5 Development Application Directory Structure



Designating a Parent of Your 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.

To set up an application as a parent of another, you can do one of the following:

- Use the `-parent` option of the `admin.jar` command when deploying the originating application. This option allows you to designate what application will be the parent of the deploying application.
- Specify the parent in the application definition line in the `server.xml` file. Each application is defined by an `<application>` element in the `server.xml` file. In this element, a `parent` attribute designates the parent application.

  ```
  <application ... parent="applicationWithCommonClasses" .../>
  ```

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. 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 2–1 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 2-18 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 "Standalone OC4J Command-Line Options and Properties" on page A-33 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 2–2 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.

Table 2–2 The Thread Pool Attributes (Cont.)

Thread Pool Attributes	Description
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 2–2 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 2–3 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 element 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 2–3. If there are multiple processes running for an OC4J instance, there is a multiple set of log files.

Table 2–3 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	`http-web-site.xml`

There are two types of log files:

- Text Log Files: The messages logged in these files are text-based and not in XML format. You can read these messages with 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 messages logged in these files use an XML format that can be read by a GUI tool, such as the Oracle Enterprise Manager GUI. We recommend that you use this format for your logging when you are using OC4J within Oracle Application Server.

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 XML files in Table 2–3. Text messaging is enabled in the `<file>` subelement the `<log>` element of the XML files, except the `http-web-site.xml` file. For the `http-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 2–4, but you can specify the location and filename within the path attribute of the `<log>` or `<access-log>` elements.

Table 2–4 shows the default location for the log files for a standalone OC4J. You can modify the location and names of these files by modifying the configuration files described in Table 2–3.

Table 2–4 OC4J Standalone Log File Locations

Log File	Default Location
`application.log`	*install_dir*/j2ee/home/application-deployments/<application-name>
`global-application.log`	*install_dir*/j2ee/home/log
`jms.log`	*install_dir*/j2ee/home/log
`rmi.log`	*install_dir*/j2ee/home/log
`server.log`	*install_dir*/j2ee/home/log
`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" />

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>
```

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 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 2–3, 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 2–3, except for the http-web-site.xml file.
- Uncomment the <odl-access-log> element in the http-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 <install-dir>/j2ee/home/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 <install-dir>/j2ee/home/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 `http-web-site.xml` file. For the `http-web-site.xml` file, turn off the text logging by commenting out the `<access-log>` element.

Redirecting Standard Out and Standard Error

Many developers use the `System.out.println()` and `System.err.println()` methods in their applications to generate debug information. Normally, the output from these method calls are printed to the console where the OC4J process is started. However, you can specify command-line options when starting OC4J to direct the `STDOUT` and `STDERR` output directly to files. The `-out` and `-err` parameters inform OC4J where to direct the error messages. The following startup command includes and example of the -out and -err parameters:

```
$ java -jar oc4j.jar -out d:\log-files\oc4j.out -err d:\log-files\oc4j.err
```

In this case, all information written to `STDOUT` and `STDERR` is printed to the files `d:\log-files\oc4j.out` and `d:\log-files\oc4j.err` respectively.

OC4J Debugging

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.

The following table provides 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 System Properties" on page A-44.

Table 2–5 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.error.debug`	Prints all HTTP errors
`http.method.trace.allow`	Default: false. If true, turns on the `trace` HTTP method.

Table 2–6 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 2–7 RMI Debugging Options

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

Table 2–8 OracleAS Web Services Debugging Options

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

For example, if you want to generate debug information on HTTP session events then you start OC4J, as follows:

```
java -Dhttp.session.debug=true -jar oc4j.jar
```

After OC4J is started with a specific debug option, debug information is generated and routed to standard output. In the above example, you would see HTTP session information on your OC4J console, as follows:

```
Oracle Application Server Containers for J2EE initialized
Created session with id '36c04d8a1cd64ef2b6a9ba6e2ac6637e' at Mon Apr 15
12:24:20 PDT 2002, secure-only: false
Created session with id '36c04d8a1cd64ef2b6a9ba6e2ac6637e' at Mon APR 15
12:36:06 PDT 2002, secure-only: false
Invalidating session with id '36c04d8a1cd64ef2b6a9ba6e2ac6637e' at Mon APR 15
12:44:32 PDT 2002 (created at Mon APR 15 12:24:23 PDT 2002) due to timeout
```

If you want to save this debug information, then you can redirect your standard output to a file using the `-out` or `-err` command-line options, as follows:

```
java -Dhttp.session.debug=true -jar oc4j.jar -out oc4j.out -err oc4j.err
```

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 OC4J command-line options section. The following examples show the output with and without verbosity:

Example 2–4 Error Messages Displayed Without Veribosity

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

Example 2–5 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.

```
java -Dhttp.session.debug=true -Ddatasource.verbose=true -jar oc4j.jar
```

Then, re-execute your servlet and see the following type of debug information in the standard output for the OC4J process:

```
DataSource logwriter activated... jdbc:oracle:thin:@localhost:1521:ORCL:
Started
jdbc:oracle:thin:@localhost:1521:ORCL: 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 OC4J Pooled
jdbc:oracle:thin:@localhost:1521:ORCL
null: Connection XA XA OC4J Pooled jdbc:oracle:thin:@localhost:1521:ORCL
allocated (Pool size: 0)
jdbc:oracle:thin:@localhost:1521:ORCL: Opened connection
Created new physical connection: Pooled
oracle.jdbc.driver.OracleConnection@5f18
Pooled jdbc:oracle:thin:@localhost:1521:ORCL: Connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 allocated (Pool size: 0)
Pooled jdbc:oracle:thin:@localhost:1521:ORCL: Releasing connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 to pool (Pool size: 1)
null: Releasing connection XA XA OC4J Pooled
jdbc:oracle:thin:@localhost:1521:ORCL to pool (Pool size: 1)
OC4J Pooled jdbc:oracle:thin:@localhost:1521:ORCL: Cache timeout, closing
connection (Pool size: 0)
com.evermind.sql.OrionCMTDataSource/default/jdbc/OracleDS: Cache timeout,
closing connection (Pool size: 0)
```

3

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 DataSources 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 exist in an XML file known as `data-sources.xml`.

Defining Location of the DataSource XML Configuration File

Your application can know about the data sources defined in this file only if the `application.xml` file knows about it. The `path` attribute in the `<data-sources>` element in the `application.xml` file must contain the name and path to your `data-sources.xml` file, as follows:

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

The `path` attribute of the `<data-sources>` element 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. Both the `application.xml` and `data-sources.xml` files are located in `j2ee/home/config/application.xml`. Thus, the path contains only the name of the `data-sources.xml` file.

Defining Data Sources

The `j2ee/home/config/data-sources.xml` file is pre-installed with a default data source. For most uses, this default is all you will need. However, you can also add your own customized data source definitions.

The 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.

This data source is extremely fast and efficient, because it does not require any JTA or XA operations. These would be necessary if you were to manage more than a single database.

The following is 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. 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*.

Alternatively, you can use the `admin.jar` command to install the data source as follows:

```
% java -jar admin.jar ormi://myhost admin welcome
     -application myapp -installDataSource -jar $ORACLE_HOME/jdbc/classes12.jar
     -url jdbc:oracle:thin:@myhost:1521:ORCL
     -connectionDriver oracle.jdbc.driver.OracleDriver
     -location jdbc/DefaultOracleDS -username hr -password hr
```

> **Note:** You must restart OC4J after you modify the `data-sources.xml` file.

See "Options for the OC4J Administration Management JAR" on page A-34 for a full description of the required parameters for this option.

The Data Sources chapter in the *Oracle Application Server Containers for J2EE Services Guide* fully describes all attributes.

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 in the `data-sources.xml` file.

   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 in the `data-sources.xml` file.

   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*.

4

Servlet Primer

Oracle Application Server Containers for J2EE (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 Standalone" 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

For more information, 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

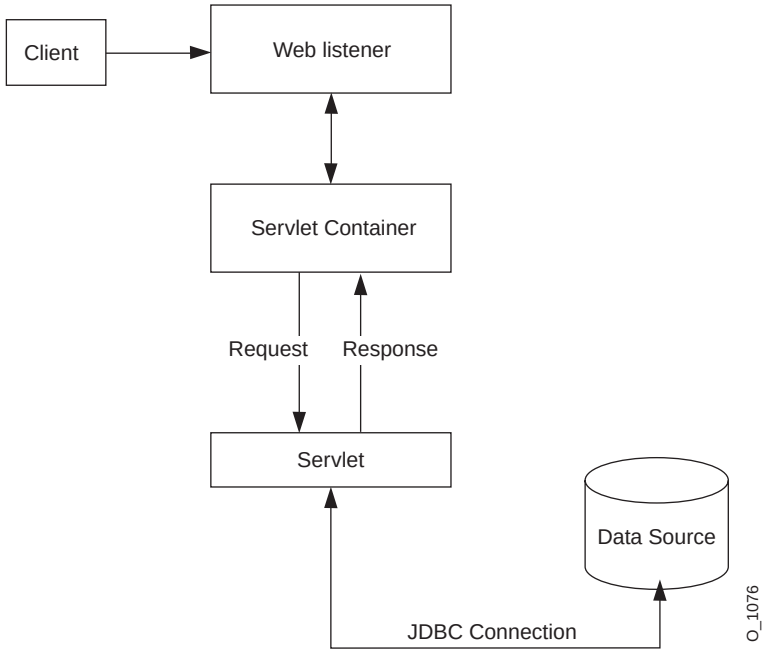
Unlike a Java client program, a servlet has no static `main()` method. Therefore, a servlet must execute under the control of an external container.

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 4–1 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 4–1 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

For convenience, simply place `HelloWorldServlet.java` and `HelloWorldServlet.class` in the

j2ee/home/default-web-app/WEB-INF/classes directory. This is the location in which the container finds servlets for the default Web application. (The default Web application is a WAR file that is automatically deployed when you install OC4J. It consists of various Web pages of static and dynamic content, so that OC4J has pages available to execute when first installed.)

Run the Hello World Servlet

Assuming you use the OC4J default Web application, which has a context path of "/", you can run the Hello World servlet with a URL such as the following:

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

By default, OC4J standalone uses port 8888.

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 Oracle9*i*.

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 3, "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

As for the Hello World example earlier in this chapter, you can place `GetEmpInfo.java` and `GetEmpInfo.class` into the `j2ee/home/default-web-app/WEB-INF/classes` director for the default Web application.

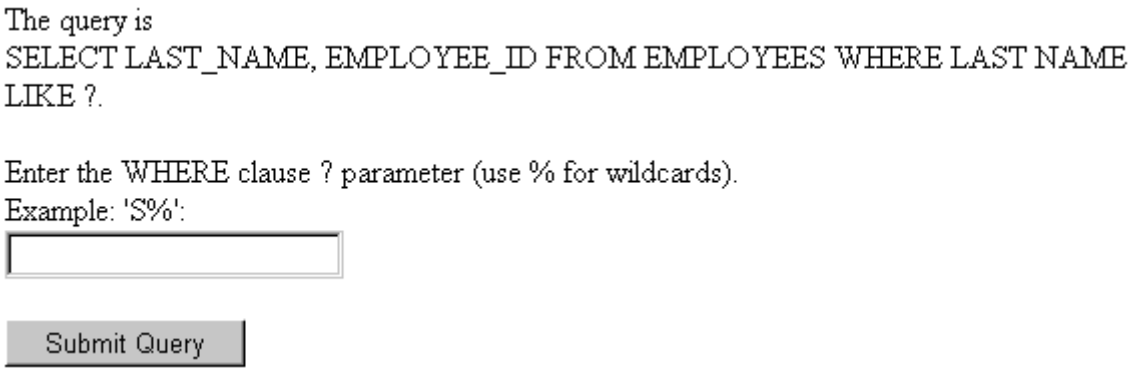
Place `EmpInfo.html` into the `/WEB-INF` directory.

Run GetEmpInfo

Assuming you use the OC4J default Web application, which has a context path of "/", you can access the front-end HTML page for the `GetEmpInfo` servlet with a URL such as the following:

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

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



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:

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.

5

JSP Primer

Oracle Application Server Containers for J2EE (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 Standalone" 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

For a complete description of Web application deployment, see "Deploying Applications" on page 1-19.

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! :-)
```

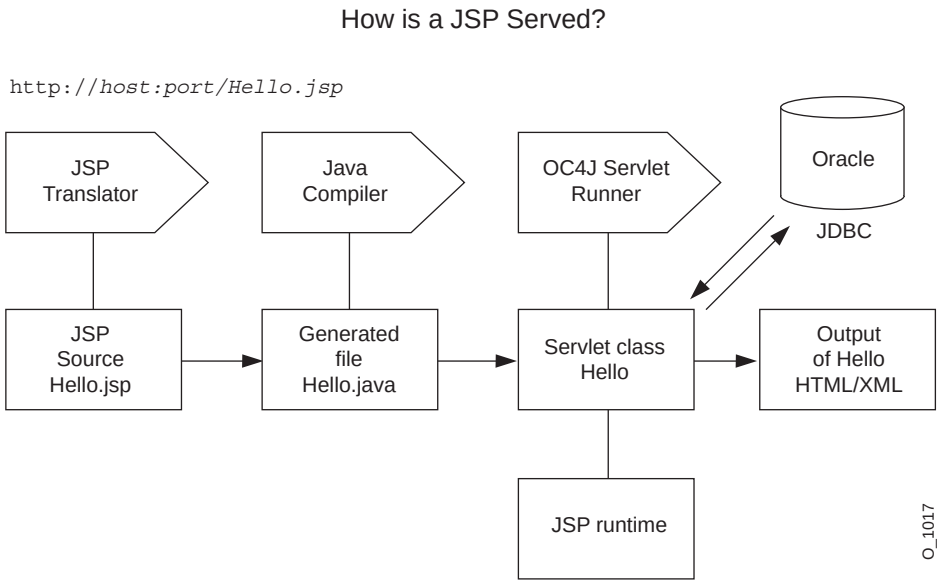
JSP Translation and Runtime Flow

Figure 5–1 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 5–1 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 5-2 into a file, and save it as `welcomeuser.jsp`.

For convenience, you can place this file under the OC4J default Web application, the base location of which is the `j2ee/home/default-web-app` directory. For example, you can place `welcomeuser.jsp` in the `j2ee/home/default-web-app/examples/jsp` directory.

Run welcomeuser.jsp

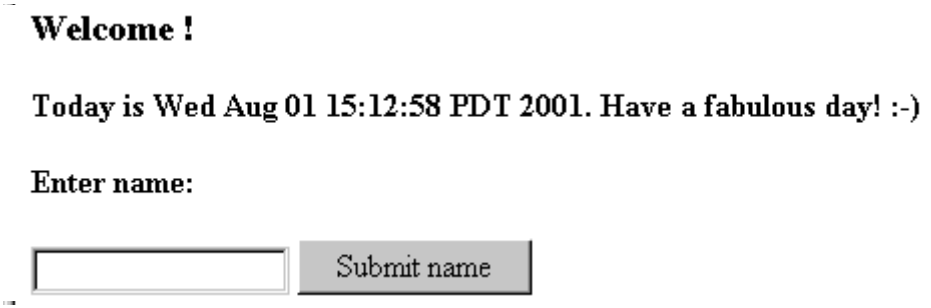
When specifying a URL to execute a JSP page in OC4J, note the following:

- By default, OC4J standalone uses port 8888.
- The context path for the OC4J default Web application is "/".
- The URL path maps to the directory path beneath the default Web application directory (or other Web application directory, as applicable).

For example, if you place `welcomeuser.jsp` in the `j2ee/home/default-web-app/examples/jsp` directory, you can run the page through the OC4J Web server with a URL such as the following:

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

and updates the page:

Welcome Amy!

Today is Wed Aug 01 15:14:29 PDT 2001. Have a fabulous day! :-)

Enter name:

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 `WEB-INF/classes` in the WAR file you use for deployment, 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

As with the earlier JSP example in this chapter, you can place `usebean.jsp` in the `j2ee/home/default-web-app/examples/jsp` directory.

Files for the JavaBean class—`NameBean.class` and optionally `NameBean.java`—should go under the `WEB-INF` directory, which is a standard J2EE Web application mechanism. In this case, this would be specifically as follows (for UNIX, or equivalently for Windows NT):

```
j2ee/home/default-web-app/WEB-INF/classes/beans
```

The `WEB-INF/classes` directory and any subdirectories are automatically in your Web application classpath. In this example, `NameBean.class` must be in the `beans` subdirectory, because that is the package name specified in the code. You must create the `beans` subdirectory.

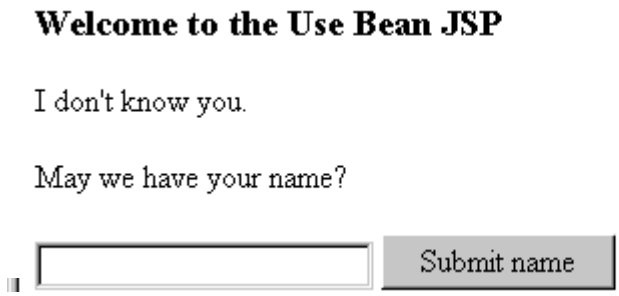
Run usebean.jsp

With `usebean.jsp` being in the `default-web-app/examples/jsp` directory, as with the earlier example in this chapter, you will use a similar URL to run it from a browser:

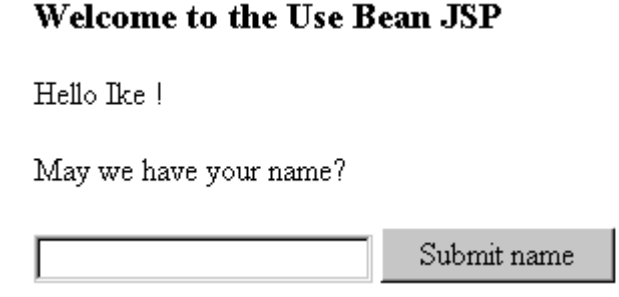
`http://host:port/examples/jsp/usebean.jsp`

This example, as before, uses *host* as the name of the system where OC4J and the application are installed. The default port is 8888.

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 5-7. 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 3, "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`. Verify that `ojsp.jar`, `ojsputil.jar`, `xmlparserv2.jar`, and `xsu12.jar` are available to the Web application.

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

As with the earlier JSP examples in this chapter, you can place `sqltagquery.jsp` in the `j2ee/home/default-web-app/examples/jsp` directory.

Run sqltagquery.jsp

With `sqltagquery.jsp` being in the `j2ee/home/default-web-app/examples/jsp` directory, as with earlier examples in this chapter, you will use a similar URL to run the page from a browser. This example, as before, uses *`host`* as the name of the system where OC4J and the application are installed:

`http://`*`host`*`:`*`port`*`/examples/jsp/sqltagquery.jsp`

This assumes that the OC4J Web server is running, and that you have run the standard SQL scripts for Oracle demos to set up a database. The default port is 8888.

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.

6

EJB Primer

After you have installed 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:

- Developing EJBs—Developing and testing an EJB module within the standard J2EE specification.
- Preparing 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.
- Deploying 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 *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.*

Developing EJBs

The development of EJB components for the OC4J environment is identical to development in any other standard J2EE environment. The steps for developing EJBs are as follows:

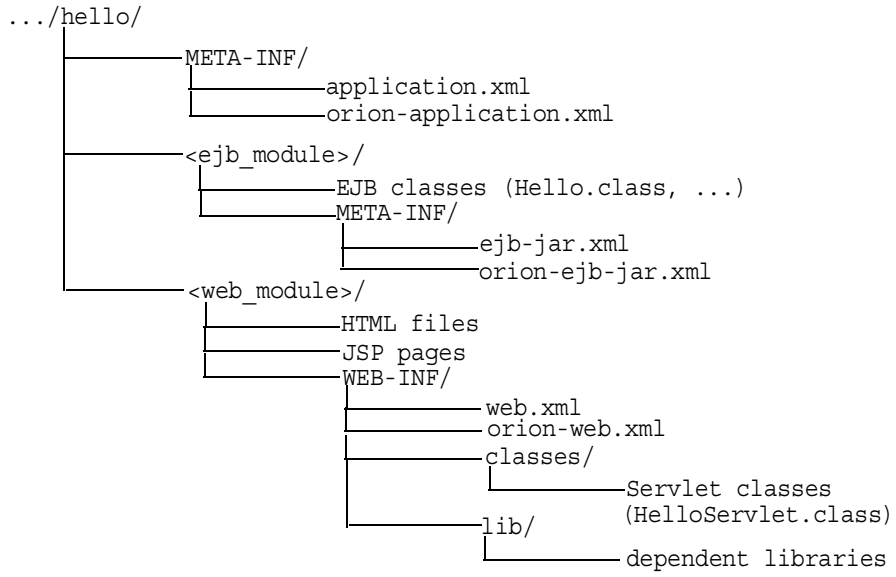
1. Creating the Development Directory—Create a development directory for the enterprise application (as shown in Figure 6–1).
2. Implementing the Enterprise JavaBeans—Develop your EJB with its home interface, remote interface, and bean implementation.
3. Accessing the EJB—Develop the client to access the bean through the remote or local interface.
4. Creating the Deployment Descriptor—Create the standard J2EE EJB deployment descriptor for all beans in your EJB application.
5. Archiving the EJB Application—Archive your EJB files into a JAR file.

Creating the Development Directory

You can develop your application in any manner. It is best 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 their own sub-directories.

The hello example was developed using the directory structure described in "Creating the Development Directory" on page 1-9. Notice in Figure 6–1 that the EJB and Web modules exist under the `hello` application parent directory and are developed separately in their own directory.

Figure 6–1 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`".

Implementing the Enterprise JavaBeans

When you implement an 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.

EJB Type	Create Parameters
Stateful Session Bean	One or more `create` methods, each with its own defined parameters.
Entity Bean	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 6–1 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 6–2 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 6–3 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 6–4 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 6–5 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`.

Accessing 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.

Example 6–6 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 6–7 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=ormi://myhost/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.

Setting JNDI Properties

If the client is collocated with the target, the client exists within the same application as the target, or the target exists within its parent, then you do not need a JNDI properties file. Else, you must initialize your JNDI properties either within a `jndi.properties` file, in the system properties, or within your implementation, before the JNDI call. The following sections discuss these three options:

- No JNDI Properties
- JNDI Properties File
- JNDI Properties Within the Implementation

To specify credentials within the JNDI properties, see the *Oracle Application Server Containers for J2EE Security Guide.*

Note: A full description of how to use JNDI, see the JNDI chapter in the *Oracle Application Server Containers for J2EE Services Guide.*

No JNDI Properties A servlet that is collocated with the target bean automatically accesses the JNDI properties for the node. Thus, accessing the EJB is simple: no JNDI properties are required.

```
//Get the Initial Context for the JNDI lookup for a local EJB
InitialContext ic = new InitialContext();
//Retrieve the Home interface using JNDI lookup
Object helloObject = ic.lookup("java:comp/env/ejb/HelloBean");
```

This is also true if the target bean is in the same application or an application that has been deployed as this application's parent. Use the `-parent` option of the `admin.jar` tool to set the parent of the application.

JNDI Properties File If setting the JNDI properties within the `jndi.properties` file, set the properties as follows. Make sure that this file is accessible from the `CLASSPATH`.

Factory

See "Using the Initial Context Factory Classes" on page 6-16 for discussion on the initial context factory to use.

```
java.naming.factory.initial=
          com.evermind.server.ApplicationClientInitialContextFactory
```

Location

Use the ORMI protocol for accessing EJBs in a standalone OC4J. The ORMI default port number is 23791, which can be modified in `config/rmi.xml`. Thus, set the provider URL in the `jndi.properties`, in one of the two ways:

```
java.naming.provider.url=ormi://<hostname>/<application-name>
```
or
```
java.naming.provider.url=ormi://<hostname>:23791/<application-name>\
```

When you access EJBs in a remote container, you must pass valid credentials to this container. Stand-alone clients define their credentials in the `jndi.properties` file deployed with the client's code.

```
java.naming.security.principal=<username>
java.naming.security.credentials=<password>
```

JNDI Properties Within the Implementation If you set the properties within the bean implementation, then set them with the same values, just with different syntax. For example, JavaBeans running within the container pass their credentials within the `InitialContext`, which is created to look up the remote EJBs.

To pass JNDI properties within the `Hashtable` environment, set these as shown below:

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
       "com.evermind.server.ApplicationClientInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "guest");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
Context ic = new InitialContext (env);
Object homeObject = ic.lookup("java:comp/env/ejb/HelloBean");

// Narrow the reference to a HelloHome.
HelloHome helloHome =
  (HelloHome) PortableRemoteObject.narrow(homeObject,
                                                  HelloHome.class);
```

Using the Initial Context Factory Classes

The type of initial context factory that you use depends on who the client is. The initial context factory creates the initial context class for the client.

- If the client is a pure Java client outside of the OC4J container, use the `ApplicationClientInitialContextFactory` class.
- If the client is an EJB or servlet client within the OC4J container, use the `ApplicationInitialContextFactory` class. The `ApplicationInitialContextFactory` class is the default class; thus, each time you create a new `InitialContext` without specifying any initial context factory class, your client uses the `ApplicationInitialContextFactory` class.
- If the client is an administrative class that is going to manipulate or traverse the JNDI tree, use the `com.evermind.server.RMIInitialContextFactory` class.
- If the client is going to use DNS load balancing, use the `RMIInitialContextFactory` class.

For example, if you have a pure Java client, then you set the initial context factory class (`"java.naming.factory.initial"`) to `ApplicationClientInitialContextFactory`. The following example sets the initial context factory in the environment, but you could also put this in the JNDI properties file.

```
env.put("java.naming.factory.initial",
      "com.evermind.server.ApplicationClientInitialContextFactory");
```

If the client is an EJB or a servlet calling an EJB in the same application, you can use the default by not setting the JNDI properties with a initial context factory and uses the `ApplicationInitialContextFactory` object by executing the following:

```
InitialContext ic = new InitialContext();
```

If you decide to use the `RMIInitialContextFactory` class, you must use the JNDI name in the lookup and not a logical name defined in the `<ejb-ref>` in your XML configuration file.

An Initial Context Factory Specific to DNS Load Balancing To use DNS as a method for your incoming load balancing, you must do the following:

1. Within DNS, map a single host name to several IP addresses. Each of the port numbers must be the same for each IP address. Set up the DNS server to return the addresses either in a round-robin or random fashion.
2. Within each client, use any initial context factory to create an initial context. You use the `ormi://` prefix in the provider URL.

3. Set the `dedicated.rmicontext` property to true.

Each time the lookup occurs on the DNS server, the DNS server hands back a one of the many IP addresses that are mapped to it.

Example 6–8 RMIInitialContextFactory Example

```
java.naming.factory.initial=
          com.evermind.server.rmi.RMIInitialContextFactory
java.naming.provider.url=ormi://myserver/applname
java.naming.security.principal=admin
java.naming.security.credentials=welcome
dedicated.rmicontext=true
```

Creating 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 6–1 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 6–9 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.

Archiving the EJB Application

Once you have finalized your implementation and have 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 "Building and Deploying Within a Directory" on page 2-12. Then, modify the `*-web-site.xml` file, and archive all Web application files into a WAR file.

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.

Preparing the EJB Application for Assembly

Before deploying, perform 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:

- Modifying Application.xml
- Creating the EAR File

Modifying Application.xml

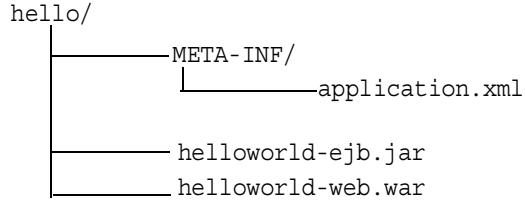
The `application.xml` file acts as the standard J2EE application descriptor file for the application and contains a list of the modules that are included within your enterprise application. You use each `<module>` element in the `application.xml` file to designate what comprises your enterprise application. Each module describes

one of three things: EJB JAR, Web WAR, and any client files. Respectively, modify the <ejb>, the <web>, and the <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 indicated in Figure 6–2, 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 only refers to the JAR and WAR files by name and relative path—and 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 6–2 Archive Directory Format



For example, the following example modifies the <ejb> and <web> 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>
```

Creating 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 that is shown in Figure 6–2:

```
% jar cvfM helloworld.ear .
```

This archives the `application.xml`, the `helloworld-ejb.jar`, and the `helloworld-web.war` files into the `helloworld.ear` file.

Deploying the Enterprise Application to OC4J

OC4J is aware of and deploys your application when it is configured within the `server.xml` file. There are two methods to provide application information within the `server.xml` file with a standalone OC4J instance:

- using `admin.jar` to modify the `server.xml` file
- updating the `server.xml` file manually

Although both methods accomplish the same result, using the `admin.jar` tool eliminates possible errors in manual updates.

Using ADMIN.JAR to Modify SERVER.XML

OC4J contains a command-line deployment tool for deploying J2EE applications—the `admin.jar` command. The options for this command are listed in the "Options for the OC4J Administration Management JAR" on page A-34.

To deploy a J2EE application with the EAR file to a remote node, execute `admin.jar`, as follows:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port
       username password -deploy -file path/filename
       -deploymentName app_name -targetPath path/destination
```

where

- The *oc4j_host:oc4j_ormi_port* is the host and port of the OC4J server.
- The *username password* is the administration username and password for the OC4J server.
- The `-file` *path/filename* indicates the local directory and filename for the EAR file.
- The `-deploymentName` *app_name* variable is the name of the application.
- The `-targetPath` *path/destination* indicates what path on the server node to deploy the EAR file into. The default path is the `applications/` directory. Oracle recommends that you provide a target path.

> **Note:** If you have a Web application within the EAR file, bind the Web application using the `admin.jar -bindWebApp` option.

Updating SERVER.XML Manually

In `server.xml`, add a new or modify the existing `<application name=... path=... auto-start="true" />` entry for each J2EE application. The path should be the full directory path and EAR filename. For our `hello` example, add the following to the `server.xml` file:

```
<application name="hello"
          path="/private/applications/helloworld.ear"
          auto-start="true" />
```

If you included a Web application portion, you must do the following to bind the Web application to the Web server. In `*-web-site.xml`, add a `<web-app ...>` entry for each Web application. The `application` attribute should be the same value as provided in the `server.xml` file. The `name` attribute should be the WAR file, without the WAR extension, for the Web application.

For Web application binding for the `hello` Web application, add the following:

```
<web-app application="hello" name="helloworld-web"
```

```
root="/hello" />
```

Verifying Deployment

OC4J detects the addition of your application to `server.xml`. The OC4J server displays a message that your application has been deployed. This is the extent of installation in OC4J. If the server does not notice your application in a timely manner, simply start (or restart) OC4J, and it will locate your application immediately.

7

Security

OC4J security employs a user manager to authenticate and authorize users and groups that attempt to access a J2EE application. User managers differ in performance and are employed based on the security you require. Confidentiality through encryption is supplied with SSL.

This chapter describes the following topics:

- Overview of Security Functions
- Authentication
- Authorization
- Plugging In a User Manager
- Confidentiality Through SSL

For a broader description of Oracle Application Server security, see the *Oracle Application Server 10g Security Guide* and the *Oracle Application Server Containers for J2EE Security Guide*.

Overview of Security Functions

OC4J security is based on a two-step process. First, a user or group attempting to access a J2EE application is authenticated, and then it is authorized. Authentication and authorization are provided under various user managers, such as the `JAZNUserManager` and `XMLUserManager` classes. The `JAZNUserManager` class is the default and offers the best security. The `XMLUserManager` is the simplest method for security. The `JAZNUserManager` leverages the OracleAS JAAS Provider as the security infrastructure for OC4J by using either the Lightweight Directory Access Protocol (LDAP)-based or the XML-based provider type. The `XMLUserManager` is configured using a file, so the passwords are visible.

See "Plugging In a User Manager" on page 7-11 for details on the OracleAS JAAS Provider, other provider types, and user managers. Also, see the *Oracle Application Server Containers for J2EE Security Guide* for details on security providers.

Note: The default user manager was changed from the `XMLUserManager` to the `JAZNUserManager`.

Authentication and authorization, along with OC4J confidentiality, are introduced below:

- Authentication: Verifies the identity and credentials of a user.

  Defines users and groups in a *user repository.* A user repository is employed by a *user manager* to verify the identity of a user or group attempting to access a J2EE application. A user repository can be a file or a directory server, depending on your environment. The OracleAS JAAS Provider-LDAP user manager and the `XMLUserManager` are two examples of user repositories.

  Although the J2EE application determines which client can access the application, it is the user manager, employing the user name and password, that verifies the client's identity, based on information in the user repository.

- Authorization: Permits or denies users and groups access to an application.

  Specifies authorization for users and groups (identities) in the J2EE *and* OC4J-specific deployment descriptors. J2EE and OC4J-specific deployment descriptors indicate what roles are needed to access the different parts of the application. *Roles* are the logical identities that each application uses to indicate access rights to its different objects. The OC4J-specific deployment descriptors provide a mapping between the logical roles and the users and groups known by OC4J.

- Confidentiality Through SSL: Ensures encrypted communications.

  Use Secure Sockets Layer (SSL) over HTTP for encrypted communication.

Authentication

Authentication verifies that the identity and credentials of a user are valid. The J2EE application determines which user can use the application. However, it is the user manager, employing the user name and password, that verifies the user's identity based on information in the user repository. Authentication is distinct from authorization, which is the process of giving a user access to a J2EE application, based on his identity.

OC4J security authenticates two types of clients: HTTP and Enterprise JavaBeans (EJBs). This section describes each of these along with setting up users and groups.

Specifying Users and Groups

OC4J supports the definition of users and groups—either shared by all deployed applications or specific to a given application.

- Shared users and groups are listed in the user repository, whose location is specified in the global `config/application.xml` file.
- Application-specific users and groups are listed in the application-specific user repository, whose location is specified in the `orion-application.xml` file of that application.

The way you define users and groups depends on what user manager you employ. For example, because the OracleAS JAAS Provider provider uses roles instead of groups, the `JAZNUserManager` XML-based user repository, `jazn-data.xml`, has a different structure from the `XMLUserManager` user repository, `principals.xml`. In addition, in a `JAZNUserManager` user repository, passwords are encrypted, unlike in `principals.xml`.

The following sections offer examples of how to specify users and groups under the `JAZNUserManager` and `XMLUserManager` classes. See "Plugging In a User Manager" on page 7-11 for additional details on these classes.

Example: Specifying Users and Groups in jazn-data.xml

The following XML from the `JAZNUserManager` user repository configuration file, `jazn-data.xml`, shows how to define OracleAS JAAS Provider roles (groups) and users. It defines a group named `allusers` and a user named `guest`.

```
<role>
  <name>allusers</name>
  <members>
   <member>
    <type>user</type>
    <name>guest</name>
   </member>
  </members>
</role>
```

Unlike the XML from the XMLUserManager user repository configuration file, principals.xml, you can encrypt the password under the JAZNUserManager.

```
<user>
  <name>guest</name>
  <description>The default user</description>
  <credentials>wEE6aA==</credentials>
</user>
```

> **Note:** See the *Oracle Application Server Containers for J2EE Security Guide* for more information on setting up the jazn-data.xml file.

Example: Specifying Users and Groups in principals.xml

The following XML from the principals.xml file (the user repository configuration file for the XMLUserManager class) shows how to define a group named allusers and a user named guest with password welcome. The guest user is made a member of the allusers group.

If you want to use the XMLUserManager class instead of the JAZNUserManager class, you must modify the global application.xml file, if modifying for all applications, or the orion-application.xml file, if using the XMLUserManager class only for a specific application. Add the following line:

```
<principals path="./principals.xml" />
```

where the path points to the location of the principals.xml file. Also, you must remove or comment out the <jazn provider> element in this file.

> **Note:** You can hide the password through password indirection. See the *Oracle Application Server Containers for J2EE Security Guide* for a description of password indirection.

```
<principals>
 <groups>
  <group name="allusers">
   <description>Group for all normal users</description>
   <permission name="rmi:login" />
   <permission name="com.evermind.server.rmi.RMIPermission" />
  </group>
....other groups...
 </groups>
 <users>
  <user username="guest" password="welcome">
   <description>Guest user</description>
   <group-membership group="allusers" />
  </user>
 </users>
</principals>
```

Authenticating HTTP Clients

OC4J requests the client to authenticate itself when accessing protected URLs. You can achieve authentication through a user name and password, or in the case of SSL, through an SSL certificate. Although in most cases where authentication is required, the user is prompted to enter a user name and password. If you decide to use an SSL certificate to authenticate the client, see "Confidentiality Through SSL" on page 7-19 for directions on how to set up your client certificate and server keystore.

Authenticating EJB Clients

When you access EJBs in OC4J, you must pass valid credentials to this server.

- Standalone clients define their credentials in the `jndi.properties` file, either deployed with the EAR file or in the `InitialContext` object.
- Servlets or JavaBeans running within OC4J pass their credentials within the `InitialContext` object, which is created to look up the remote EJBs.

Setting JNDI Properties

If the client exists within the same application as the target, or the target exists within its parent, you do not need a JNDI properties file. If not, you must initialize your JNDI properties either within a `jndi.properties` file, in the system properties, or within your implementation, before the JNDI call. The following sections discuss these three options:

- No JNDI Properties
- JNDI Properties File
- JNDI Properties Within Implementation

No JNDI Properties A servlet that exists in the same application with the target bean automatically accesses the JNDI properties for the node. Therefore, accessing the EJB is simple, because no JNDI properties are required.

```
//Get the Initial Context for the JNDI lookup for a local EJB
InitialContext ic = new InitialContext();
//Retrieve the Home interface using JNDI lookup
Object empObject = ic.lookup("java:comp/env/employeeBean");
```

This is also true if the target bean is in an application that has been deployed as this application's parent. To specify parents, use the `-parent` option of the `admin.jar` command when deploying the originating application.

JNDI Properties File If you are setting the JNDI properties within the `jndi.properties` file, set the properties as follows. Ensure that this file is accessible from the `CLASSPATH`.

Factory

```
java.naming.factory.initial=
    com.evermind.server.ApplicationClientInitialContextFactory
```

Location

The ORMI default port number is 23791, which you can modify in `j2ee/home/config/rmi.xml`. Therefore, set the URL in the `jndi.properties`, in one of two ways:

```
java.naming.provider.url=ormi://hostname/application-name
```

- or -

```
java.naming.provider.url=ormi://hostname:23791/application-name
```

Security

When you access EJBs in OC4J, you must pass valid credentials to this server. Standalone clients define their credentials in the `jndi.properties` file deployed with the code of the client.

```
java.naming.security.principal=username
```

```
java.naming.security.credentials=password
```

JNDI Properties Within Implementation Set the properties with the same values, but with different syntax. For example, JavaBeans running within the container pass their credentials within the `InitialContext`, which is created to look up the remote EJBs.

To pass JNDI properties within the `Hashtable` environment, set these as shown below:

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
      "com.evermind.server.ApplicationClientInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "guest");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
Context ic = new InitialContext (env);
Object homeObject = ic.lookup("java:comp/env/employeeBean");

// Narrow the reference to a TemplateHome.
EmployeeHome empHome =
 (EmployeeHome) PortableRemoteObject.narrow(homeObject,
                                                EmployeeHome.class);
```

Using the Initial Context Factory Classes

For most clients, set the initial context factory class to `ApplicationClientInitialContextFactory`. If you are not using a logical name defined in the `<ejb-ref>` in your XML configuration file, then you must provide the actual JNDI name of the target bean. In this case, you can use a different initial context factory class, the `com.evermind.server.RMIInitialContextFactory` class.

Example 7–1 Servlet Accessing EJB in Remote OC4J Instance

The following servlet uses the JNDI name for the target bean: `/cmpapp/employeeBean`. Thus, this servlet must provide the JNDI properties in an `RMIInitialContext` object, instead of the `ApplicationClientInitialContext` object. The environment is initialized as follows:

- The `INITIAL_CONTEXT_FACTORY` is initialized to a `RMIInitialContextFactory`.
- Instead of creating a new `InitialContext`, it is retrieved.

- The actual JNDI name is used in the lookup.

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "ormi://myhost/cmpapp");
env.put(Context.SECURITY_PRINCIPAL, "admin");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.evermind.server.rmi.RMIInitialContextFactory");

Context ic =
    new com.evermind.server.rmi.RMIInitialContextFactory().
        getInitialContext(env);

Object homeObject = ic.lookup("/cmpapp/employeeBean");

// Narrow the reference to a TemplateHome.
EmployeeHome empHome =
    (EmployeeHome) PortableRemoteObject.narrow(homeObject,
        EmployeeHome.class);
```

Authorization

Authorization is the process of granting or denying a user access to a J2EE application based on its identity. Authorization is distinct from authentication, which is the process of verifying that a user is valid.

Specify authorization for users and groups in the J2EE *and* OC4J-specific deployment descriptors. The J2EE deployment descriptor is where you specify the access rules for using logical roles. The OC4J-specific deployment descriptor is where you map logical roles to actual users and groups, which are defined in a user repository.

The following sections describe how to define users, groups, and roles:

- Specifying Logical Roles in a J2EE Application
- Mapping Logical Roles to Users and Groups

Specifying Logical Roles in a J2EE Application

Specify the logical roles that your application uses in the XML deployment descriptors. Depending on the application component type, update one of the following with the logical roles:

- `web.xml` for the Web component

- `ejb-jar.xml` for the EJB component
- `application.xml` for the application

In each of these deployment descriptors, the roles are defined by an XML element named `<security-role>`.

Example 7–2 EJB JAR Security Role Definition

The following steps describe the XML necessary to create a logical role named `VISITOR` in the `ejb-jar.xml` deployment descriptor.

1. Define the logical security role, `VISITOR`, in the `<security-role>` element.

```
<security-role>
 <description>A role for every user</description>
 <role-name>VISITOR</role-name>
</security-role>
```

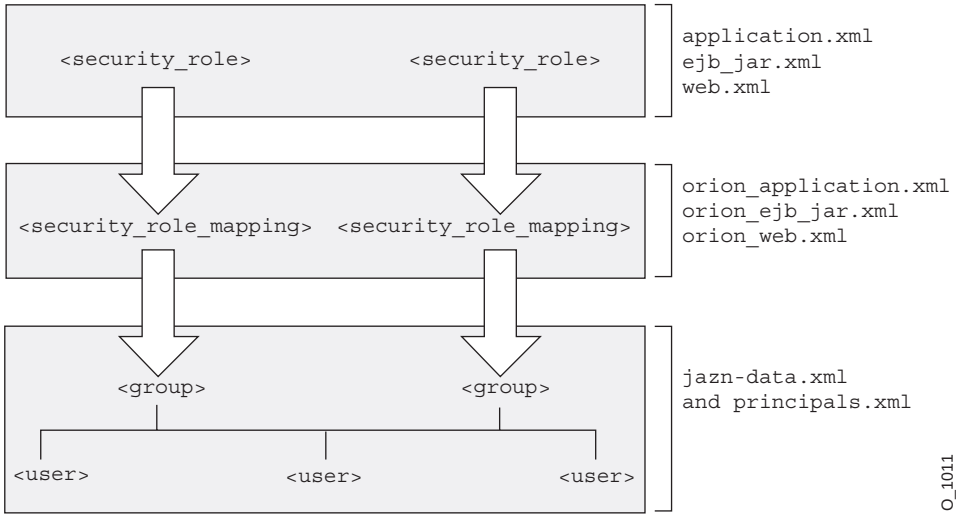
2. Define the bean and methods that this role can access in the `<method-permission>` element.

```
<method-permission>
 <description>VISITOR role needed for CustomerBean methods</description>
 <role-name>VISITOR</role-name>
 <method>
  <ejb-name>customerbean</ejb-name>
  <method-name>*</method-name>
 </method>
</method-permission>
```

Mapping Logical Roles to Users and Groups

Map logical roles defined in the application deployment descriptors to actual users and groups defined in a user repository. The mapping is specified in the OC4J-specific deployment descriptor with a `<security-role-mapping>` element. Figure 7–1 illustrates this mapping.

Figure 7–1 Mapping Logical Roles to Users and Groups Defined in jazn-data.xml



Note: The security role mapping layer, either defined in the `principals.xml` or `jazn-data.xml` file, is bypassed if the following conditions are true:

- The name of the security role and group (or roles, as in the case of `jazn-data.xml`) are the same.
- No security role mapping is specified.

Example 7–3 Mapping Logical Role to Actual Role

This example maps the logical role `VISITOR` to the `allusers` group in the `orion-ejb-jar.xml` file. Any user that can log in as part of this group is considered to have the `VISITOR` role and can therefore execute the methods of `customerbean`. This role is mapped to the `allusers` group, which is defined in the User Manager configuration file—the `jazn-data.xml` file.

```
<security-role-mapping name="VISITOR">
 <group name="allusers" />
</security-role-mapping>
```

Note: You can map a logical role to a single group or to several groups.

Plugging In a User Manager

Any user manager class providing OC4J security is an implementation of the `com.evermind.security.UserManager` interface. This includes any custom user managers you create. User manager classes manage users, groups, and passwords with such methods as `createUser()`, `getUser()`, and `getGroup()`. Table 7–1 lists the user managers that you can employ in OC4J security.

Table 7–1 User Managers and Their User Repositories Available to OC4J

User Manager	User Repository
`oracle.security.jazn.oc4j.`**`JAZNUserManager`**	■ using the XML-based provider type—`jazn-data.xml` ■ using the LDAP-based provider type—OID
`com.evermind.server.`**`XMLUserManager`**	`principals.xml`
Custom user manager	user-provided user repository

By default, OC4J reads the user names, groups, and passwords from the `JAZNUserManager` user repository, `jazn-data.xml`. In order for OC4J to employ any user manager, you must specify the name of the user manager class in one of the following XML files:

- `orion-application.xml`—file for a single application
- `config/application.xml`—global configuration file for all applications in the server

The following sections describe how to configure each User Manager type:

- Using the JAZNUserManager Class
- Using the XMLUserManager Class
- Creating Your Own User Manager

Using the JAZNUserManager Class

The primary purpose of the `JAZNUserManager` class is to leverage the OracleAS JAAS Provider as the security infrastructure for OC4J. For a complete description of the OracleAS JAAS Provider, see the *Oracle Application Server Containers for J2EE Security Guide*.

By integrating the OracleAS JAAS Provider with OC4J, the following benefits can be achieved:

- Single Sign-on (SSO)/`mod_osso` integration
- SSL/`mod_ossl` integration
- OID integration (using the LDAP-based provider type)
- Fine-grained access control using Java2 permissions
- `run-as` identity support, delegation support (from servlet to EJB)
- Secure file-based storage of passwords (using the XML-based provider type)

Use the `JAZNUserManager` class if you want OC4J security that has secure, centralized storage, retrieval, and administration of OracleAS JAAS Provider data. This data consists of realm (user and roles) and policy (permissions) information. Figure 7–2 illustrates the architecture of OC4J security under the `JAZNUserManager` class.

The `JAZNUserManager` class can use two types of OracleAS JAAS Providers for OC4J security. Use the provider type that is appropriate for your environment:

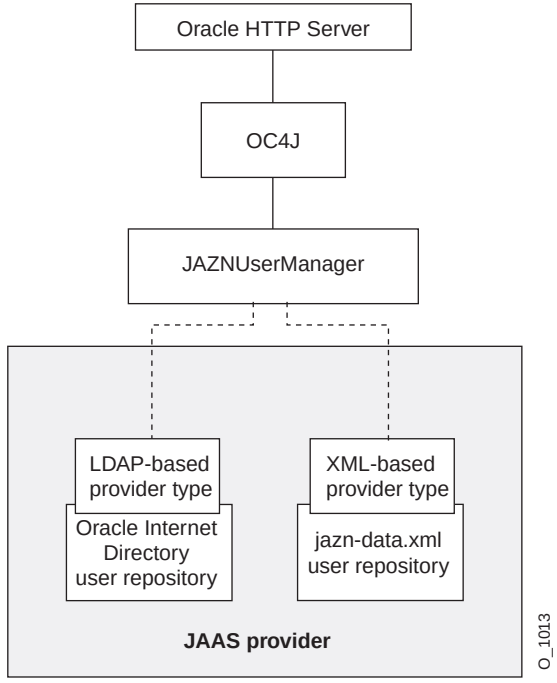
- LDAP-based

  For centralized storage of information in a directory. The user repository is OID.

- XML-based

  For lightweight storage of information in an XML file. The user repository is the `jazn-data.xml` file.

Figure 7–2 OC4J Security Architecture Under the JAZNUserManager Class



In OC4J, you can configure your application(s) to use the `JAZNUserManager` class by adding the `<jazn>` or `<user-manager>` element in your OC4J-specific configuration file (`config/application.xml` or `orion-application.xml`).

Using the JAZNUserManager Class with the LDAP-Based Provider Type

The LDAP-based provider type delegates user and group management functionality to the Delegated Administrative Service (DAS) from OID.

The following examples from an OC4J-specific configuration file have OC4J employ the `JAZNUserManager` class as the user manager with the LDAP-based provider type.

```
<jazn provider="LDAP" default-realm="sample_subrealm"
    location="ldap://myoid:389" />
```

- or -

```
<user-manager class="oracle.security.jazn.oc4j.JAZNUserManager">
```

```
    <property name="provider.type" value="LDAP" />
    <property name="realm.default" value="sample_subrealm" />
    <property name="ldap.service"  value="ldap://myoid:389" />
</user-manager>
```

Notes: If you specify both the `<user-manager>` element and the `<jazn>` element, then the `<jazn>` element is ignored.

Using the JAZNUserManager Class with the XML-Based Provider Type

The XML-based provider type is a fast, lightweight implementation of the OracleAS JAAS Provider API. This provider type uses XML to store user names and encrypted passwords.

The following examples from an OC4J-specific configuration file have OC4J employ the `JAZNUserManager` class as the user manager with the XML-based provider type. The user repository is located at `.../j2ee/home/jazn/config/`**`jazn-data.xml`**. Because there is only one realm in the data file, the specification of `realm.default` is not needed.

```
<jazn provider="XML"
    location=".../j2ee/home/config/jazn-data.xml" />
```

- or -

```
<user-manager class="oracle.security.jazn.oc4j.JAZNUserManager">
    <property name="provider.type" value="XML" />
    <property name="xml.store.fs.jazn"
        value=".../j2ee/home/config/jazn-data.xml" />
</user-manager>
```

Notes: If you specify both the `<user-manager>` element and the `<jazn>` element, then the `<jazn>` element is ignored.

Using the XMLUserManager Class

The `XMLUserManager` is a file-based security model, where all of your users, roles, groups, and passwords are stored in `principals.xml`. This is not secure as your passwords could be in the clear.

However, if you want to use the `XMLUserManager` class instead of the `JAZNUserManager` class, you must modify the global `application.xml` file, if

modifying for all applications, or the `orion-application.xml` file, if using the `XMLUserManager` class only for a specific application. Add the following line:

```
<principals path="./principals.xml" />
```

where the path points to the location of the `principals.xml` file. Also, you must remove or comment out the `<jazn>` element in this file. If you do not remove or comment out the `<jazn>` element, then whichever element is specified first is the User Manager for the applications. For example, if you have the following:

```
<principals path="./principals.xml" />
<jazn provider="XML"
    location=".../j2ee/home/config/jazn-data.xml" />
```

In this case, the `<principals>` element appears first, so the `XMLUserManager` is the security manager.

Creating Your Own User Manager

If none of the user managers supplied by OC4J are suitable for your specific user authentication needs, then you can create your own user manager and configure OC4J to use it.

To create your own user manager, complete the following steps:

1. Write a custom user manager.

   Your custom user manager class must implement the `com.evermind.security.UserManager` interface. Table 7–2 describes the methods of this interface.

Table 7–2 Methods of the UserManager Interface

Method	Description
`void addDefaultGroup (java.lang.String name)`	Adds a group to the set of default groups, of which all users of the user manager are members. ▪ `java.lang.String name` - the name of the group being added to the default group
`Group createGroup (java.lang.String name)`	Creates a new group. If the group already exists, a `java.lang.InstantiationException` is thrown. ▪ `java.lang.String name` - the name of the new group
`User createUser (java.lang.String username, java.lang.String password)`	Creates a new user. ▪ `java.lang.String username` - the new user name ▪ `java.lang.String password` - the new user password
`User getAdminUser()`	Returns the default admin user or `null` if there is none.
`User getAnonymousUser()`	Returns the default anonymous user or `null` if none exists.
`java.util.Set getDefaultGroups()`	Returns the set of default groups for the user manager.
`Group getGroup(java.lang.String name)`	Returns the group with the specified name or `null` if none exists. ▪ `java.lang.String name` - the name of the specified group
`int getGroupCount()`	Returns the number of users contained in the user manager. Throws `UnsupportedOperationException` if not supported.
`java.util.List getGroups (int start,int max)`	Returns a list of groups (between the specified indexes) contained in the user manager. Throws `UnsupportedOperationException` if not supported.
`UserManager getParent()`	Returns the parent manager of the user manager.
`User getUser (java.lang.String username)`	Returns the user with the specified user name or `null` if there is no match.
`User getUser (java.lang.String issuerDN, java.math.BigInteger serial)`	Returns the user associated with this certificate or `null` if either certificates are not supported or there is no user associated with this certificate.

Table 7–2 Methods of the UserManager Interface (Cont.)

Method	Description
`User getUser (java.security.cert.X509Certificate certificate)`	Returns the user associated with this certificate or `null` if either certificates are not supported or there is no user associated with this certificate.
`int getUserCount()`	Returns the number of users contained in this manager. Throws `UnsupportedOperationException` if not supported.
`java.util.List getUsers (int start,int max)`	Returns a list of users (between the specified indexes) contained in this manager. Throws `UnsupportedOperationException` if not supported.
`void init (java.util.Properties properties)`	Instantiates the user manager with the specified settings. Throws `java.lang.InstantiationException` if any errors occur.
`boolean remove(Group group)`	Removes the specified group from the user manager and returns `true` if the operation is successful.
`boolean remove(User user)`	Removes the specified user from the user manager and returns `true` if the operation is successful.
`void setParent (UserManager parent)`	Sets the parent user manager if one exists. This method is called only on a nested user manager. A user manager can delegate work to its parent user manager.

2. Plug the user manager into your application.

   For a single application, specify the custom user manager in the `<user-manager>` element of the `orion-application.xml` file. For all applications in the server, specify the custom user manager in the `<user-manager>` element of the `config/application.xml` file.

3. Define your users and groups.

   See "Specifying Users and Groups" on page 7-3.

4. Create security constraints in your Web application.

   See "Authorization" on page 7-8.

Example 7–4 Using the DataSourceUserManager Class

The following example of the `DataSourceUserManager` class is a custom user manager and it implements the `UserManager` interface. Within its methods, the

DataSourceUserManager class manages the users in a database specified by the DataSource interface.

To configure a custom user manager, you specific the classname in the class attribute of the <user-manager> element in either the global application.xml file or the orion-application.xml file. Then, you can specify input parameters and values through the name/value attributes of one or more <property> elements.

For our DataSourceUserManager example, it requires that the table name and columns are defined in the <property> element name/value pairs. This example sets up the following input parameters:

- Data source that specifies the database where the tables reside
- Table for user names and passwords
- Table for user and group association

A typical registration of the user manager for an application can be specified in orion-application.xml, as follows:

```
<user-manager class="com.evermind.sql.DataSourceUserManager">
    <property name="dataSource" value="jdbc/OracleCoreDS" />
    <property name="table" value="j2ee_users" />
    <property name="usernameField" value="username" />
    <property name="passwordField" value="password" />
    <property name="groupMembershipTableName" value="second_table" />
    <property name="groupMembershipGroupFieldName" value="group" />
    <property name="groupMembershipUserNameFieldName" value="userId" />
</user-manager>
```

The <user-manager> property elements define the input parameters into the UserManager class. It assumes that the tables that these refer to already exist in the database.

The user manager is a hierarchical implementation with a parent-child relationship. The parent of the DataSourceUserManager class is the default file-based XMLUserManager class, which uses the principals.xml user repository. However, you can change the parent with the setParent() method. The sample DataSourceUserManager class invokes parent.getGroups() to retrieve all the available groups from its parent, the XMLUserManager.

Confidentiality Through SSL

OC4J supports Secure Socket Layer (SSL) communication between the client and a standalone OC4J, using HTTPS.

The following sections document SSL in detail:

- Overview of Using SSL for OC4J Standalone
- Configuration of OC4J for SSL
- HTTPS Common Problems and Solutions

Overview of Using SSL for OC4J Standalone

The following sections describe security features and discuss how to use them with OC4J standalone:

- Overview of SSL Keys and Certificates
- Using Certificates with OC4J Standalone

Overview of SSL Keys and Certificates

In SSL communication between two entities, each entity (or at least the server) has an associated *public key* and a *private key*. During communication, each entity uses its own private key, together with the public key of the other party, to ensure that they can communicate with each other. If one entity encrypts data using its private key, then the other party can decrypt the data by only by using the public key of the originating entity. If one entity encrypts data using the public key of the other party, then that party can decrypt the data only by using its own private key.

Each key is a number, with the private key of an entity being kept secret by that entity, and the public key of an entity being publicized to any other parties with which secure communication might be necessary.

A *certificate* is a digitally signed statement from a recognized issuer that verifies the public key of an entity. Such an issuer is referred to as a *certificate authority* (CA). An issued certificate is typically associated with a *root certificate*. This association, or "chaining", of certificates establishes a chain of trust. An issuer might have its own root certificate, and will chain any certificates it issues to its own root certificate.

Functionally, a certificate acts as a container for keys, holding private keys (as applicable), public keys, and associated signatures. A single certificate file can contain an entire chain of certificates.

A *keystore* is used for storage of certificates, including the certificates of all trusted parties. Through its keystore, an entity, such as OC4J, can authenticate itself to other parties.

A keystore is a `java.security.KeyStore` instance that you can create and manipulate using the `keytool` utility, provided with the Sun Microsystems JDK. Go to the following site for information about `keytool`:

```
http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html
```

During secure communication between the client and OC4J, the following functionality is executed:

- The link (all communications) between the two is encrypted.
- OC4J is authenticated to the client through a security challenge and response. A "secret key" is securely exchanged and used for the encryption of the link.
- Optionally, if OC4J is in client-authentication mode, the client is authenticated to OC4J.

Using Certificates with OC4J Standalone

The steps for using keys and certificates for SSL communication in OC4J are as follows. These are server-level steps, typically executed prior to deployment of an application that will require secure communication, perhaps when you first set up OC4J.

1. Use `keytool` to generate a private key, public key, and unsigned certificate.You can place this information into either a new keystore or an existing keystore.
2. Obtain a signature for the certificate, using either of the following two approaches.
   - You can generate your own signature by using `keytool` to "self-sign" the certificate. This is appropriate if your only clients will trust you as, in effect, your own certificate authority.
   - You can obtain a signature from a recognized certificate authority through the following steps:
     a. Using the certificate from Step 1, use `keytool` to generate a *certificate request*, which is a request to have the certificate signed by a certificate authority.
     b. Submit the certificate request to a certificate authority.

c. Receive the signature from the certificate authority and import it into the keystore, again using `keytool`. In the keystore, the signature will be matched with the associated certificate.

The process for requesting and receiving signatures is up to the particular certificate authority you use. Because that is outside the scope and control of Oracle Application Server, it is not covered in Oracle Application Server documentation. You can go to the Web site of any certificate authority for information. Any browser should have a list of trusted certificate authorities. Here are the Web addresses for VeriSign and Thawte (acquired by VeriSign), for example:

```
http://www.verisign.com/

http://www.thawte.com/
```

In addition, Oracle provides a certificate authority, where each certificate is recognized only by Oracle applications. The Oracle Certificate Authority (OCA) allows customers to create and issue certificates for themselves and their users, although these certificates would likely be unrecognized outside a customer's organization without prior arrangements. See the *Oracle Application Server 10g Security Guide* for information about OCA.

Configuration of OC4J for SSL

For secure communication between a client and OC4J, configuration is required on OC4J standalone. You are required to provide a certificate on the client-side only if you configure client-authentication.

In the `http-web-site.xml` file of OC4J (or other Web site XML file, as appropriate), you must specify appropriate SSL settings under the `<web-site>` element.

1. Turn on the `secure` flag to specify secure communication, as follows:

   ```
   <web-site ... protocol="http" secure="true" ... >
      ...
   </web-site>
   ```

   Setting `secure="true"` specifies that the HTTP protocol is to use an SSL socket.

2. Use the `<ssl-config>` subelement and its `keystore` and `keystore-password` attributes to specify the directory path and password for the keystore, as follows:

```
<web-site ... secure="true" ... >
   ...
   <ssl-config keystore="path_and_file" keystore-password="pwd" />
</web-site>
```

The `<ssl-config>` element is required whenever the `secure` flag is set to `"true"`.

The *`path_and_file`* value can indicate either an absolute or relative directory path and includes the file name.

Note: You can hide the password through password indirection. See *Oracle Application Server Containers for J2EE Security Guide* for a description of password indirection.

---

3. Optionally, turn on the `needs-client-auth` flag, an attribute of the `<ssl-config>` element, to specify that client authentication is required, as follows:

```
<web-site ... secure="true" ... >
   ...
   <ssl-config keystore="path_and_file" keystore-password="pwd"
               needs-client-auth="true" />
</web-site>
```

This step sets up a mode where OC4J accepts or rejects a client entity for secure communication, depending on its identity. The `needs-client-auth` attribute instructs OC4J to request the client certificate chain upon connection. If the root certificate of the client is recognized, then the client is accepted.

The keystore specified in the `<ssl-config>` element must contain the certificates of any clients that are authorized to connect to OC4J through HTTPS.

4. Optionally, specify each application in the Web site as shared. The `shared` attribute of the `<web-app>` element indicates whether multiple bindings (different Web sites, or ports, and context roots) can be shared. Supported values are `"true"` and `"false"` (default).

Sharing implies the sharing of everything that makes up a Web application, including sessions, servlet instances, and context values. A typical use for this mode is to share a Web application between an HTTP site and an HTTPS site at the same context path, when SSL is required for some but not all of the

communications. Performance is improved by encrypting only sensitive information, rather than all information.

If an HTTPS Web application is marked as shared, then instead of using the SSL certificate to track the session, the cookie is used to track the session. This is beneficial in that the SSL certificiate uses 50K to store each certificate when tracking it, which sometimes results in an "out of memory" problem for the session before the session times out. This could possibly make the Web application less secure, but might be necessary to work around issues such as SSL session timeouts not being properly supported in some browsers.

5. Optionally, set the cookie domain if `shared` is true and the default ports are not used. When the client interacts with a Web server over separate ports, the cookie believes that each separate port denotes a separate Web site. If you use the default ports of 80 for HTTP and 443 for HTTPS, the client recognizes these as two different ports of the same Web site and creates only a single cookie. However, if you use non-default ports, the client does not recognize these ports as part of the same Web site and will create separate cookies for each port, unless you specify the cookie domain.

   Cookie domains track the client's communication across multiple servers within a DNS domain. If you use non-default ports for a shared environment with HTTP and HTTPS, set the `cookie-domain` attribute in the `<session-tracking>` element in the `orion-web.xml` file for the application. The cookie-domain attribute contains the DNS domain with at least two components of the domain name provided.

   ```
   <session-tracking cookie-domain=".oracle.com" />
   ```

Example 7–5 HTTPS Communication With Client Authentication

The following configures a Web site for HTTPS secure communication with client authentication:

```
<web-site display-name="OC4J Web Site" protocol="http" secure="true" >
   <default-web-app application="default" name="defaultWebApp" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome"
               needs-client-auth="true" />
</web-site>
```

Only the portions in bold are specific to security. The protocol value is always `"http"` for HTTP communication, whether or not you use secure communication. A protocol value of `http` with `secure="false"` indicates HTTP protocol; `http` with `secure="true"` indicates HTTPS protocol.

Then, configures the news application to accept both HTTP and HTTPS connections:

```
<web-app application="news" name="news-web" root="/news" shared="true" />
```

This Web site uses the default port numbers for HTTP and HTTPS communication. If it did not, you would also add the `cookie-domain` attribute.

```
<session-tracking cookie-domain=".oracle.com" />
```

For more information about elements and attributes of the `<web-site>`, `<web-app>`, and `<session-tracking>` elements, see the XML Appendix in the *Oracle Application Server Containers for J2EE Servlet Developer's Guide.*

Example 7–6 Creating an SSL Certificate and Configuring HTTPS

The following example uses `keytool` to create a test certificate and shows all of the XML configuration necessary for HTTPS to work. To create a valid certificate for use in production environments, see the `keytool` documentation.

1. Install the correct JDK

   Ensure that JDK 1.3.x is installed. This is required for SSL with OC4J. Set the JAVA_HOME to the JDK 1.3 directory. Ensure that the JDK 1.3.x `JAVA_HOME/bin` is at the beginning of your path. This may be achieved by doing the following:

   UNIX

   ```
   $ PATH=/usr/opt/java130/bin:$PATH
   $ export $PATH
   $ java -version
   java version "1.3.0"
   ```

   Windows

   ```
   set PATH=d:\jdk131\bin;%PATH%
   ```

   Ensure that this JDK version is set as the current version in your Windows registry. In the Windows Registry Editor under HKEY_LOCAL_MACHINE / SOFTWARE / JavaSoft / Java Development Kit, set 'CurrentVersion' to 1.3 (or later).

2. Request a certificate

   a. Change directory to `ORACLE_HOME/j2ee`

b. Create a keystore with an RSA private/public keypair using the keytool command. In our example, we generate a keystore to reside in a file named 'mykeystore', which has a password of '123456' and is valid for 21 days, using the 'RSA' key pair generation algorithm with the following syntax:

```
keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456
-validity 21
```

Where:

- the `keystore` option sets the filename where the keys are stored
- the `storepass` option sets the password for protecting the keystore
- the `validity` option sets number of days the certificate is valid

The `keytool` prompts you for more information, as follows:

```
keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456
-validity 21

What is your first and last name?
  [Unknown]:  Test User
What is the name of your organizational unit?
  [Unknown]:  Support
What is the name of your organization?
  [Unknown]:  Oracle
What is the name of your City or Locality?
  [Unknown]:  Redwood Shores
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Test User, OU=Support, O=Oracle, L=Reading, ST=Berkshire, C=GB>
correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

Note: To determine your 'two-letter country code', use the ISO country code list at the following URL: `http://www.bcpl.net/~jspath/isocodes.html`.

The mykeystore file is created in the current directory. The default alias of the key is mykey.

3. If you do not have a secure-web-site.xml file, then copy the http-web-site.xml to $J2EE_HOME/config/secure-web-site.xml.

4. Edit secure-web-site.xml with the following elements:

   a. Add secure="true" to the <web-site> element, as follows:

   ```
   <web-site port="8888" display-name="Default Oracle Application Server
   Containers for J2EE Web Site" secure="true">
   ```

   b. Add the following new line inside the <web-site> element to define the keystore and the password.

   ```
   <ssl-config keystore="<Your-Keystore>"
   keystore-password="<Your-Password>" />
   ```

   Where <Your-Keystore> is the full path to the keystore and <Your-Password> is the keystore password. In our example, this is as follows:

   ```
   <!-- Enable SSL -->
   <ssl-config keystore="../../keystore" keystore-password="123456"/>
   ```

   Note: The keystore path is relative to where the XML file resides.

   c. Change the web-site port number, to use an available port. For example, the default for SSL ports is 443, so change the Web site port attribute to port="4443". To use the default of 443, you have to be a super user.

   d. Now save the changes to secure-web-site.xml.

5. If you did not have the secure-web-site.xml file, then edit server.xml to point to the secure-web-site.xml file.

   a. Uncomment or add the following line in the file server.xml so that the secure-web-site.xml file is read.

   ```
   <web-site path="./secure-web-site.xml" />
   ```

   Note: Even on Windows, you use a forward slash and not a back slash in the XML files.

b. Save the changes to `server.xml`.

6. Stop and re-start OC4J to initialize the `secure-web-site.xml` file additions. Test the SSL port by accessing the site in a browser on the SSL port. If successful, you will be asked to accept the certificate, since it is not signed by an accepted authority.

When completed, OC4J listens for SSL requests on one port and non-SSL requests on another. You can disable either SSL requests or non-SSL requests, by commenting out the appropriate `*web-site.xml` in the `server.xml` configuration file.

```
<web-site path="./secure-web-site.xml" /> - comment out this to remove SSL
<default-site path="./http-web-site.xml" /> - comment out this to
                                                        remove non-SSL
```

Requesting Client Authentication with OC4J Standalone

OC4J supports a "client-authentication" mode in which the server explicitly requests authentication from the client before the server will communicate with the client. In this case, the client must have its own certificate. The client authenticates itself by sending a certificate and a certificate chain that ends with a root certificate. OC4J can be configured to accept only root certificates from a specified list in establishing a chain of trust back to the client.

A certificate that OC4J trusts is called a trust point. This is the first certificate that OC4J encounters in the chain from the client that matches one in its own keystore. There are three ways to configure trust:

- The client certificate is in the keystore.
- One of the intermediate certificate authority certificates in the client's chain is in the keystore.
- The root certificate authority certificate in the client's chain is in the keystore.

OC4J verifies that the entire certificate chain up to and including the trust point is valid to prevent any forged certificates.

If you request client authentication with the `needs-client-auth` attribute, perform the following:

1. Decide which of the certificates in the client's chain is to be your trust point. Ensure that you either have control of the issue of certificates using this trust point or that you trust the certificate authority as an issuer.

2. Import the intermediate or root certificate in the server keystore as a trust point for authentication of the client certificate.
3. If you do not want OC4J to have access to certain trust points, make sure that these trust points are not in the keystore.
4. Execute the preceding steps to create the client certificate, which includes the intermediate or root certificate installed in the server. If you wish to trust another certificate authority, obtain a certificate from that authority.
5. Save the certificate in a file on the client.
6. Provide the certificate on the client initiation of the HTTPS connection.
   a. If the client is a browser, set the certificate in the client browser security area.
   b. If the client is a Java client, you must programmatically present the client certificate and the certificate chain when initiating the HTTPS connection.

HTTPS Common Problems and Solutions

The following errors may occur when using SSL certificates:

Keytool Error: java.security.cert.CertificateException: Unsupported encoding

Cause: You cannot allow trailing whitespace in the keytool.

Action: Delete all trailing whitespace. If the error still occurs, add a new line in your certificate reply file.

Keytool Error: KeyPairGenerator not available

Cause: You are probably using a keytool from an older JDK.

Action: Use the keytool from the latest JDK on your system. To ensure that you are using the latest JDK, specify the full path for this JDK.

Keytool Error: Failed to establish chain from reply

Cause: The keytool cannot locate the root CA certificates in your keystore; thus, the keytool cannot build the certificate chain from your server key to the trusted root certificate authority.

Action: Execute the following:

```
keytool -keystore keystore -import -alias cacert -file cacert.cer (keytool
-keystore keystore -import -alias intercert -file inter.cer)
```

If you use an intermediate CA keytool, then execute the following:

```
keystore keystore -genkey -keyalg RSA -alias serverkey keytool -keystore
keystore -certreq -file my.host.com.csr
```

Get the certificate from the Certificate Signing Request, then execute the following:

```
keytool -keystore keystore -import -file my.host.com.cer -alias serverkey
```

No available certificate corresponds to the SSL cipher suites which are enabled

Cause: Something is wrong with your certificate.

IllegalArgumentException: Mixing secure and non-secure sites on the same ip + port

Cause: You cannot configure SSL and non-SSL web-sites to listen on the same port and IP address.

Action: Check to see that different ports are assigned within `secure-web-site.xml` and `http-web-site.xml` files.

Keytool does not work on HP-UX

Cause: On HP-UX, it has been reported that the 'keytool' does not work with the RSA option.

Action: Generate the key on another platform and FTP it to the HP-UX server.

General SSL Debugging

You can get more debug information from the JSSE implementation. To get a list of options, start OC4J with:

```
java -Djavax.net.debug=help -jar oc4j.jar
```

Or, if you want to turn on full verbosity, use:

```
java -Djavax.net.debug=all -jar oc4j.jar
```

Both options will display:

- Browser request header
- Server HTTP header
- Server HTTP body (HTML served)
- Content length (before and after encryption)
- SSL version

For UNIX, you could use the startup scripts in NOTE 150215.1 'Scripts to Administer OC4J on Unix Platforms', and amend these.

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
- Standalone OC4J Command-Line Options and Properties
- Configuration and Deployment Examples

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
- http-web-site.xml
- jazn-data.xml
- principals.xml
- data-sources.xml
- jms.xml
- rmi.xml

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
- 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 `http-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="./http-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.

http-web-site.xml

This file contains the configuration for a Web site. In the `http-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

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 passwords
- Name and description of users, groups, and roles

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
- Username/password for the data source
- Database schema to use
- Inactivity time-out
- 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
- 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-18 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:

- 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-20 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
- 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-31 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, the global Web application, and the default Web site
- Maximum number of HTTP connections the server allows
- Logging settings
- Java compiler settings
- 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

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"`—Default in standalone OC4J is "`true`". If true, task manager checks for XML configuration file modifications. Thus, if you set to false, you can disable automatic refreshing of the configuration to any new XML modifications. Also, setting this attribute to false stops the automatic deployment of any applications until you execute `admin.jar -updateConfig`. If set to false, you cause the XML configuration to refresh from the XML files and any necessary automatic deployment to occur by using the `admin.jar -updateConfig` option.
- `recovery-procedure="automatic|prompt|ignore"`— Specifies how the EJB container recovers a global transaction (JTA) if an error occurs in the middle of the transaction. If a CMP bean is in the middle of a global transaction when an error occurs, then 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 caused by the OC4J server asking for permission to see whether there is 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>

Within the `<application-server>` element, you can configure the following elements, which are listed alphabetically and not by DTD ordering:

`<application>`

An application is an 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 setting 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), then 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 setting 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 later. 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 or 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.

<execution-order>

This element defines the order in which the startup classes are executed. The value is an integer. OC4J loads from 0 on up. If there are duplicate numbers, then OC4J chooses the ordering for those classes.

<global-application>

The default application for this server. This element acts as a parent to other applications for 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 from this file. Although this `application.xml` file has the same name, 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 2-23 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 one minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one.

  Value is a long. The default is 600,000 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 one minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one.

  Value is a long. The default is 600,000 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 runs in the 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 supported only 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"
    ```

  * Microsoft 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. You can specify multiple directories where each directory is separated by a colon. Each `JAR` archive in the specified directories is 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 you specify the extension directories in either this attribute or in the `-Djava.ext.dirs` system property.

The following four examples 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>

Defines the maximum number of concurrent connections any given Web site can accept at a single point in time. If text exists inside the element, 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

The top level element of the `application.xml` file is the `<application>` element.

Elements Contained Within <application>

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

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>

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.The default is false.

<file path="../log/server.log" />

A relative/absolute path to log events to.

Attribute:

- `path` - The path to the log file.

<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` element 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 elements 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

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>

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.

Attribute:

- 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`.

Standalone OC4J Command-Line Options and Properties

You start OC4J through `oc4j.jar`. You manage OC4J through the `admin.jar` tool. The following sections describe the options for each JAR.

- Options for the OC4J Server JAR
- Options for the OC4J Administration Management JAR

Options for the OC4J Server JAR

The `oc4j.jar` command-line options enable you to start, stop, and install OC4J.

Table A–2 lists all the `oc4j.jar` command-line options:

Table A–2 OC4J Command-Line Options

Command-Line Options	Description
`-install`	Installs the server and activates the admin account. Rewrites text files to match the operating system line feed. This should be used only the first time.
`-quiet`	Supress standard output.
`-config`	Specifies a location for the `server.xml` file.
`-out [file]`	Specifies a file to route the standard output to. The file contains messages that are printed to `System.out`, as well as the messages sent to output through the servlet logging interface. If not specified, all output is written to standard out.
`-err [file]`	Specifies a file to route standard error to. The file contains messages that are printed to `System.err`. If not specified, all errors are written to standard error.
`-verbosity`	Define an integer between 1 and 10 to set the verbosity level of the message output. Example: `-verbosity 10`. See Example 2–5 for an example of this option.
`-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.
`-userThreads`	Enables context lookup support from user-created threads.
`-version`	Prints the version and exits.
`-? -help`	Prints the help message.

Options for the OC4J Administration Management JAR

The `admin.jar` command-line tool enables you to administer any stand alone OC4J from a client-admin console using a command line.

The syntax is as follows:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password options
```

The options for the `admin.jar` command-line tool cover the four subjects below:

- General OC4J administration described in Table A–3.
- Application deployment described in Table A–4.
- Web site administration described in Table A–5.
- Data source administration described in Table A–6.

General OC4J Administration

Table A–3 lists the `admin.jar` options for general OC4J administration. For example, the following command shuts down the OC4J server:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -shutdown
```

Table A–3 Options for OC4J Administration

Option	Description
`-shutdown [ordinary | force] [reason]`	Shuts down the OC4J server. The default is "`ordinary`." Ordinary allows each thread to terminate normally. Force terminates all threads immediately. The reason is a string that is logged with the termination.
`-restart [reason]`	Restarts the OC4J server. The container must have been started with `oc4j.jar`. The reason is a string that is logged with the restart.

Application Deployment

Table A–4 lists the `admin.jar` options for OC4J application administration. For example, the following command structure is used to deploy an application:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -deploy -file path/filename
    -deploymentName app_name -targetPath deploy_dir
```

The following command structure is used to bind a Web application:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -bindWebApp app_name web_app_name
                                web_site_name context_root
```

Table A–4 Options for Application Deployment

Option	Description
`-deploy`	Deploy (redeploy) an application. Supply application information in the following subswitches: `-file` *`path/filename`*: Required. The path and filename of the EAR file to deploy. `-deploymentName` *`app_name`*: Required. The user-defined application deployment name. This same name is used to identify the application within OC4J. It is also provided when you want to undeploy the application. `-targetPath` *`deploy_dir`*: Optional. The path on the server node to deploy archive into. Default is the `applications/` directory. It is best to provide a target path to the directory where the EAR file is copied for deployment. If `-targetPath` is not specified, the EAR file is copied to the `applications/` directory. OC4J maintains a unique name for the EAR file. Thus, when you redeploy the EAR file, OC4J renames the file by prepending an underscore character ('_') in front of the name to ensure that another application's EAR file is not overwritten. Each successive deployment will cause another underscore character to be prepended to the EAR file. However, if it is the same application, the `applications/` directory contains a separate EAR file for each deployment. If you provide a target path, this problem does not occur. `-parent` *`parent_appname`*: Optional. The parent application of this application. When deployed, any method within the child application can invoke any method within the parent application. This is a means to enable methods in one JAR to see EJBs that have been deployed in another JAR. This is useful to deploy all service EJBs in a single JAR file, where its users declare the service application as its parent. The default is the global application. `-deploymentDirectory` *`path`*: Optional. If not specified, the application is deployed into the `application-deployments/` directory. To change where the application is deployed, provide a path with this option. If you supply the string "[NONE]", the deployment configurations are always read from the EAR file each time the application is deployed.

Table A–4 Options for Application Deployment (Cont.)

Option	Description
`-bindWebApp` *app_name web_app_name web_site_name context_root*	Bind a Web application to the specified site and root. ■ *app_name* is the application name, which is the same name used in `-deploymentName` on the `-deploy` option. Also note that this is the same name that is saved in the `<application name=`*app_name*`>` attribute in the `server.xml` file. ■ *web_app_name* is the name of the WAR file contained within the EAR file—without the `.WAR` extension. ■ *web_site_name* is the name of the *name*`-web-site.xml` file that denotes the Web site that this Web application should be bound to. This is the file that will receive the Web application definition. ■ *context_root* is the root context for the Web module. This option creates an entry in the OC4J *name*`-web-site.xml` configuration file that was denoted in the *web_site_name* variable.
`-updateConfig`	If you have set `check-for-updates` to false, then OC4J does not automatically refresh modifications of the XML files. You have to execute this flag to have OC4J upload all of the new changes to these files.
`-undeploy` *app_name*	Removes the deployed J2EE application from the OC4J Web server. The *app_name* is the name provided on the `-deploymentName` subswitch. This results in the following: ■ Application is removed from the OC4J runtime and the `server.xml` file. ■ Bindings for all the application's Web modules are removed from all the Web sites to which the Web modules were bound. ■ Application files are removed from both the `applications` and `application-deployments` directories. `-keepFiles`: Optional subswitch that prevents application files from being removed. However, the application is removed from the runtime and the Web modules are unbound.
`-deploymentDirectory "[NONE]"`	If you specify this flag as "[NONE]", then OC4J uses the `orion-ejb-jar.xml` deployment descriptor in the current deployment to be used instead of the deployment descriptor from a previous deployment within the `application-deployments` directory.

Table A–4 Options for Application Deployment (Cont.)

Option	Description
`-iiopClientJar`	You can convert an EJB to use RMI/IIOP, making it possible for EJBs to invoke one another across EJB containers. See the RMI/IIOP chapter in the *Oracle Application Server Containers for J2EE Services Guide* for full details.

Adding Web Sites

The `-site` option enables you to add Web site configuration to the XML files. Table A–5 lists all the subswitches for the `-site` option of the `admin.jar` command-line tool.

For example, the following command structure installs a new Web site:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -site -add -host hostname -port portnumber
      -display-name name -virtual-hosts virtual_host
```

Table A–5 Options for Web Site Administration

-site options	Description
`-site -add`	Installs a new Web site. Supply information with the following subswitches: `-host` *`hostname`*: The host where the web site exists. `-port` *`portnum`*: The Web site port. `-display-name` *`name`*: The name of the Web site. `-virtual-hosts` *`virtual_hosts`*: The virtual hosts of the Web site. `-secure true|false`: The value is `true` if the Web site is secure, otherwise the value is `false`. `-factory` *`factory_name`*: The name of the `SSLServerSocketFactory` class if you are not using the Java Secure Socket Extension (JSSE). The JSSE defines a provider interface that other security providers can implement. Sun Microsystems provides its own implementation in `com.sun.net.ssl.internal.ssl.Provider`. `-keystore` *`keystore`*: The relative or absolute path to a keystore. `-storepass` *`password`*: The keystore password. `-provider` *`provider`*: The provider used if using JSSE, defaults to `com.sun.net.ssl.internal.ssl.Provider`. `-needs-client-auth true|false`: If set to `true`, a client that wants to access a J2EE Web site needs to identify itself with a digital certificate. If set to `false`, a client does not need to identify itself with a digital certificate. The default is `false`.
`-site -remove`	Removes an existing Web site. Supply the host and port of this Web site with the following subswitches: `-host` *`hostname`*: The Web site host to be removed. `-port` *`portnum`*: The Web site port to be removed.
`-site -test`	Tests an existing Web site. Supply the host and port of the Web site to be tested with the following subswitches: `-host` *`hostname`*: The Web site host to be tested. `-port` *`portnum`*: The Web site port to be tested.
`-site -list`	Lists all existing Web sites.

Table A–5 Options for Web Site Administration (Cont.)

-site options	Description
`-site -update`	Updates an existing Web site. Supply information with the following subswitches: `-oldHost` *`hostname`*: The old host of the Web site. You can change the Web site host and port with the "old" and "new" subswitches. `-oldPort` *`portnum`*: The old port of the Web site. `-newHost` *`hostname`*: The new host of the Web site. `-newPort` *`portnum`*: The new port of the Web site. `-display-name` *`name`*: The new display name of the Web site. `-virtual-hosts` *`vhosts`*: The new virtual hosts of the Web site. `-secure true|false`: If set to `true`, the Web site is secure. If set to `false`, the Web site is not secure. The default is `false`. `-factory` *`classname`*: The new name of the `SSLServerSocketFactory` class if you are not using JSSE. `-keystore` *`path`*: The new relative or absolute path to a keystore. `-storepass` *`password`*: The new keystore password. `-provider` *`provider`*: The new provider used if you are not using JSSE. `-needs-client-auth true|false`: If set to `true`, a client that wants to access a J2EE Web site needs to identify itself with a digital certificate. If set to `false`, a client does not need to identify itself with a digital certificate. The default is `false`.

DataSource And Application Options

Table A–6 lists the `-application` option subswitches for the `admin.jar` command-line tool. The `-application` takes in a name of an application before the subswitch command. This *`name`* can be one of the following:

- The global application name, installed originally as `default`, specified in the `name` attribute of the `<global-application>` element in the `server.xml` file.
- A specific application name defined within an `<application>` element in the `server.xml` file.

This name, while a string, should not be enclosed in quotes. For example, the following command lists all data source objects defined:

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -application default -listDataSource
```

Table A–6 Options For Application And Data Source Management

-application Option	Description
`-application` *`name`* `-restart`	Restarts the application. This triggers auto-deployment if enabled and a file has been touched.
`-application` *`name`* `-addUser` *`username password`*	Adds a user to the security file (`principals.xml`).
`-application` *`name`* `-dataSourceInfo`	Retrieves the dynamic usage information about the installed `DataSource` objects.
`-application` *`name`* `-listDataSource`	Retrieves the statically configured information about each installed `DataSource` object.
`-application` *`name`* `-testDataSource`	Tests an existing `DataSource`. Supply information with the following subswitches: `-location` *`location`*: The namespace location for the `DataSource`. For example, `jdbc/DefaultDS`. Required. `-username` *`username`*: The username you use to login along with a password. Optional. `-password` *`password`*: The password to log in with. Optional.

Table A–6 Options For Application And Data Source Management (Cont.)

-application Option	Description
`-application` *name* `-installDataSource`	Installs a new `DataSource`. Supply information within the following subswitches: `-jar` *JARfile*: The JAR file containing the driver that is to be added to the library of the server. `-url` *URL*: The JDBC database URL. `-location` *JNDIlocation*: The namespace location for the raw source. For example, `"jdbc/DefaultPooledDS"`. Required. `-pooledLocation` *JNDIlocation*: The namespace location for the pooled source. For example, `"jdbc/DefaultPooledDS"`. `-xaLocation` *JNDIlocation*: The namespace location for the XA source. For example, `"jdbc/xa/DefaultXADS"`. Requied if `-ejbLocation` is specified. `-ejbLocation` *JNDIlocation*: The namespace location for the container-managed transactional data source. This is the only data source that can perform global JTA transactions. For example, `"jdbc/DefaultDS"`. `-username` *username*: The username to log in with. `-password` *password*: The password to log in with. `-connectionDriver` *driverClass*: The JDBC database driver class. `-classname` *DSclass*: The data source class name, such as `com.evermind.sql.DriverManagerDataSource`. Required. `-sourceLocation` *jndiDS*: The underlying data source of this specialized data source. `-xaSourceLocation` *jndiXADS*: The underlying XA data source of this specialized data source.
`-application` *name* `-removeDataSource`	Remove an existing `DataSource`. Supply information with the following subswitches: `-location` *JNDIlocation*: The namespace location for the `DataSource`. For example, `jdbc/DefaultDS`. Required.

Table A–6 Options For Application And Data Source Management (Cont.)

-application Option	Description
`-application` *name* `-updateDataSource`	Update an existing `DataSource`. Supply information with the following subswitches: `-oldLocation` *JNDIlocation*: The old namespace location for the `DataSource`. For example, `jdbc/DefaultDS`. Required. `-newLocation` *JNDIlocation*: The new namespace location for the `DataSource`. For example, `jdbc/DefaultDS`. `-jar` *JAR*: The JAR file containing the driver to add to the library of the server. `-url` *URL*: The JDBC database URL. `-pooledLocation` *JNDIlocation*: The namespace location for the pooled source. For example, `jdbc/DefaultPooledDS`. `-xaLocation` *JNDIlocation*: The namespace location for the XA `DataSource`. For example, `jdbc/xa/DefaultXADS`. Required if `-ejbLocation` is specified. `-ejbLocation` *JNDIlocation*: The namespace location for the data source for container-managed transactions. This is the only data source that can perform global JTA transactions. For example, `jdbc/DefaultDS`. `-username` *username*: The username you use to login. `-password` *password*: The password you use to login. `-connectionDriver` *driverClass*: The JDBC database driver class. For example, `com.mydb.Driver`. `-className` *dsClass*: The data source class name. For example, `com.evermind.sql.DriverManagerDataSource`. `-sourceLocation` *jndiDS*: The underlying data source of this specialized data source. `-xaSourceLocation` *jndiXADS*: The underlying XA data source of this specialized data source.

OC4J System Properties

You can set system properties 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, `-DGenerateIIOP`.

- Table A–7 details general system properties.
- Table A–8 details debugging properties.

Table A–7 -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.
`GenerateIIOP=` `true/false`	Default is false. If true, enables IIOP stub generation.
`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–7 -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.
`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.

Table A–7 -D General System Properties for OC4J (Cont.)

-D Option	Description
`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`.
`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 false. See the "Entity Relationship Mapping" chapter in the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for more information.

Table A–7 -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–8 -D System Properties for Debugging

-D Debug System Properties	Description
`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.

Table A–8 -D System Properties for Debugging (Cont.)

-D Debug System Properties	Description
`http.request.debug`	Default: false. If true, provides information about each HTTP request directed 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.
`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.
`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 2-31.

Configuration and Deployment Examples

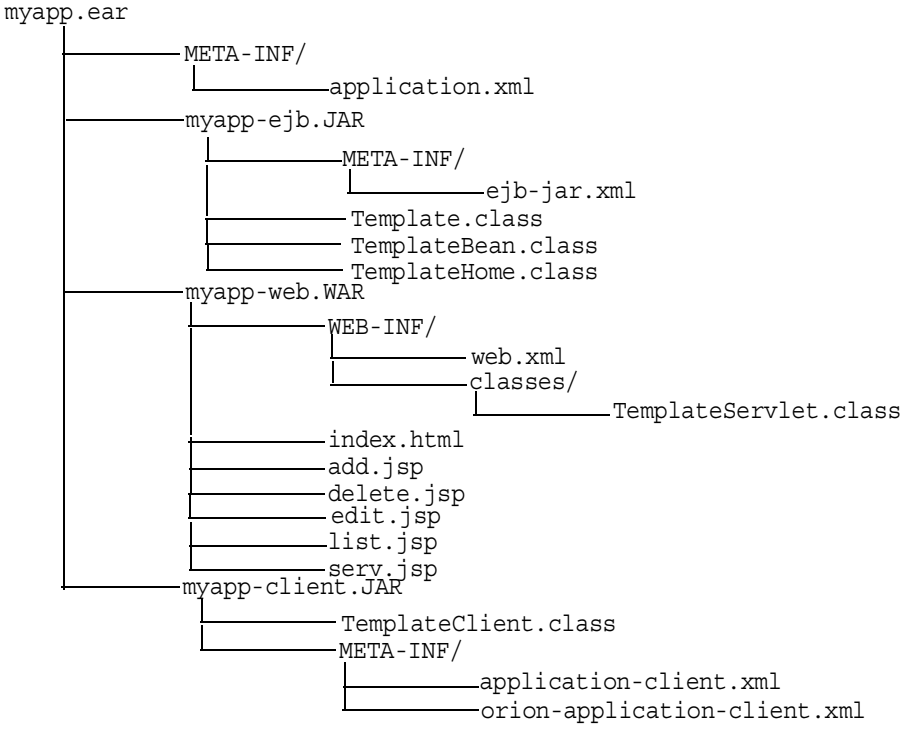
The following examples show how to configure and deploy a J2EE application within OC4J. See "Configuring the FAQ Application Demo" on page 1-11 to learn how to modify the XML configuration files for the FAQ application demo.

- J2EE Application XML Configuration Example
- Deploying Example

J2EE Application XML Configuration Example

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 as specified using the `admin.jar bind` command (`http://`*`oc4j_host:port`*`/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 `admin.jar -deploy` option, 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` attribute to `FALSE`.

> **Note:** If you set `auto-start` to `FALSE`, you can manually start the application using the `admin.jar` tool or it is automatically started when a client requests the application.

```
<application name="myapp" path="../myapp/myapp.ear"
         auto-start="true" />
```

where

- The `name` attribute is the name of the application.
- The `path` indicates the directory and filename for the EAR file.
- The `auto-start` attribute indicates if this application should be automatically started each time OC4J is started.

http-web-site.xml Addition

You must designate the WAR file name and define the root context for the Web application, which was deployed in the WAR file. You can either bind the Web context through the `admin.jar -bindWebApp` option or edit the `http-web-site.xml` file and add the following:

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

- 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://`*`oc4j_host`*`:8888`", then to initiate the application, you would point your browser at "`http://`*`oc4j_host`*`:8888/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=ormi://oc4j_host:23791/myapp
  java.naming.security.principal=admin
  java.naming.security.credentials=welcome
  ```

Deploying Example

After developing your J2EE application, assemble the different modules of your J2EE application (EJB, Web, and client) into an EAR file. This section provides an example of a J2EE application with a EJB, Web, and client sections.

To deploy this application from the client using the `admin.jar` command-line tool, perform the following from the `myapp` directory. Notice that it defines the EAR file

in the -file option and the target path for copying the EAR file into in the -targetPath option. Because the path where the EAR resides and the target path is the same, no copying occurs.

```
% java -jar $J2EE_HOME/admin.jar ormi://oc4j_host admin welcome
  -deploy -file ./myapp.ear -deploymentName myapp
Auto-deploying myapp (New server version detected)...
Auto-deploying myapp-ejb.jar (ejb-jar.xml had been touched since the
previous deployment)... done.
Auto-deploying myapp web application (New server version detected)...
```

Note: The EJB JAR file is immediately unpacked; the WAR file is unpacked when you navigate to /myapp on the Web server.

EJB Module

When you deployed the EJB module, the following messages were received:

```
Auto-deploying myapp (New server version detected)...
Auto-creating table: create table TemplateBean (col_1 NUMBER not null
primary key, col_2 VARCHAR2(255) null, col_3 FLOAT null)
Auto-deploying myapp-ejb.jar (Class 'myapp.myapp-ejb.Template' had been
updated)... done.
```

OC4J created the TemplateBean table for you; however, you must first install a data source. You can use the admin.jar command-line tool to install the data source, as follows:

```
% java -jar admin.jar ormi://oc4j_host admin welcome
      -installDataSource -jar $ORACLE_HOME/jdbc/classes12.jar
      -url jdbc:oracle:thin:@oc4j_host:1521:orcl
      -connectionDriver oracle.jdbc.driver.OracleDriver
      -location jdbc/DefaultOracleDS -username scott -password tiger
```

Web Module—Servlet and JSP Calling EJBs

To bind the Web component (WAR file) of a J2EE application (EAR file) on a Web site, do the following:

```
% java -jar admin.jar ormi://oc4j_host admin welcome
    -bindWebApp myapp myapp-web http-web-site /myapp
```

This adds the following to http-web-site.xml:

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

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/lib/jndi.jar
../../../j2ee/home/lib/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
```

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

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

X