Oracle® Application Server Containers for J2EE

Standalone User's Guide

10*g* Release 2 (10.1.2)

Part No. B14361-02

July 2005

ORACLE®

Oracle Application Server Containers for J2EE Standalone User's Guide, 10*g* Release 2 (10.1.2)

Part No. B14361-02

Primary Authors: Sheryl Maring, Dan Hynes

Contributing Author: Brian Wright

Contents

2 Advanced Configuration and Development

3 Configuring Security

A Troubleshooting OC4J

B Additional Information

C Third Party Licenses

Index

Preface

This preface introduces you to the *Oracle Application Server Containers for J2EE Standalone 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. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

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

Related Documentation

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 Security Guide*
- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

The following documentation may also be helpful in understanding OC4J:

- *Oracle Application Server Performance Guide*
- *Oracle Application Server High Availability Guide*
- *Oracle9i JDBC Developer's Guide and Reference*
- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server DMS API Reference*

Conventions

The following text conventions are used in this document:

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

1

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 andEJBs. 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

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).

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

Table 1–1 (Cont.) OC4J J2EE Support

J2EE 1.3 Standard APIs	Version Supported
Java Mail	1.1.2
Java Database Connectivity (JDBC)	2.0 Extension
Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider	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 10*g*. To use those features, customers must install one of the Oracle Application Server installation types, 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.

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) 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://`*`hostname`*`:8888/`" from a Web browser. If you changed the default port number, access the Web server using "`http://`*`hostname:portnumber`*`/`".

For example, test the Web server by connecting a Web browser to `http://`*`hostname`*`: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 1-3. For more information on configuration, see "Deploying Applications" on page 1-13.

Starting and Stopping 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.

Important: Due to a logging implementation dependency issue, OC4J fails to start when using JDK 1.3. The solution to this problem is to remove or comment out the following entry in the `ORACLE_HOME/j2ee/home/config/server.xml` configuration file:

```
<j2ee-logging-config path="./j2ee-logging.xml" />
```

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 B-26 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) 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 B-26 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 environment, access the Windows 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 2-7 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
   ```

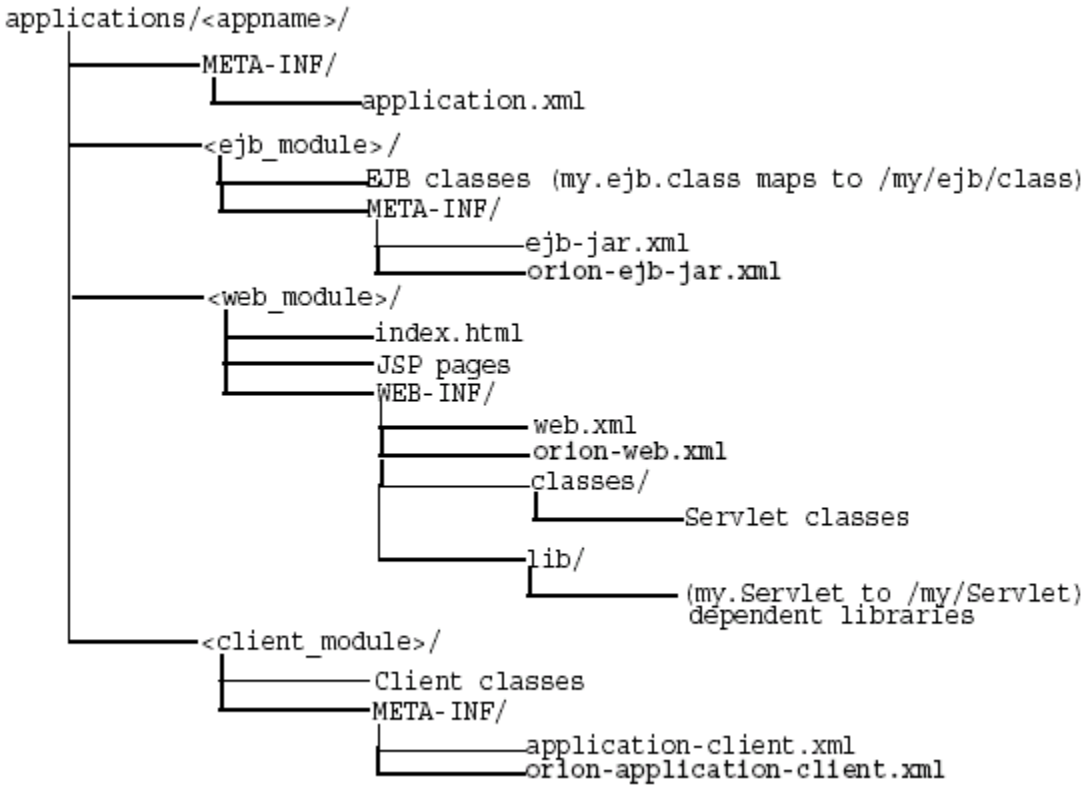
2. Place JSP pages anywhere in the `j2ee/home/default-web-app` directory. They are accessible with URLs of the form: `http://`*`oc4j_host`*`:8888/`*`path-to-JSP`*

   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, it is good practice to 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 OracleAS Web Services 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 (JAAS) Provider
- 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, you need to modify the back-end database to contain tables that are necessary for executing the FAQ demo.

Oracle Database

Create the database tables for the FAQ demo by executing the SQL table creation script CreateTables.sql script, which is located at `http://<hostname>:8888/FAQApp/CreateTables.sql` or can be downloaded

with the rest of the FAQ application from OTN at `http://www.oracle.com/technology/tech/java/oc4j/demos/904/index.html` 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/MYSERVICE, and uses the username/password of faq/faq, then the j2ee/home/config/data-sources.xml file is modified to point to the database service at the URL jdbc:oracle:thin:@myhost:1521/MYSERVICE, 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/MYSERVICE"
  inactivity-timeout="30"
/>
```

Security Configuration

The FAQ demo uses Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider 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 Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider as your security provider.

Deploy the FAQ Demo

Download the FAQ demo application from OTN at `http://www.oracle.com/technology/tech/java/oc4j/demos/904/index.html` 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://`*`your_machine_name:8888`*`/FAQApp`

Deploy Using Automatic Deployment in a Development Environment

As discussed in "Restarting OC4J" on page 1-5), 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 B-6.

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 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 2-7, "OC4J Automatic Deployment for Applications" on page 2-10, and "What Happens When You Deploy?" on page 1-17.

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 1-5) 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\applications\FAQApp.ear
      -deploymentName FAQApp
      -targetPath applications/
```

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 B-26.

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

```
java -jar admin.jar
   ormi://oc4j_host:oc4j_ormi_port
   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 B-26.

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
Create or obtain the application.	Download the `FAQApp.zip` from OTN
Make any necessary server environment changes.	Set the `JAVA_HOME` variable
Modify any application XML configuration files.	All of the application XML files are provided for you in the FAQ ZIP file.
Update the application standard J2EE application descriptor file.	The `application.xml` file is included in the `FAQApp.EAR` file.
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.
Register the application in the appropriate server XML files.	Modify the `server.xml` and `http-web-site.xml` files.
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.
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 1-3.

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 2-1 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

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:

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 B-26. Ensure that automatic deployment is disabled by setting the check-for-updates attribute to false (see "Restarting OC4J" on page 1-5).

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.

> **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 B-6.

Binding the Web Application

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 B-26.

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.

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.

Impact of Undeploying/Redeploying an Application

Undeploying a J2EE application from an OC4J instance results in the following:

- The application is removed from the OC4J runtime and is no longer accessible.
- All bindings for the Web applications are removed from all the Web sites to which the Web modules were bound.
- All application files are removed from both the `applications/` and `application-deployments/` directories.

During a redeployment, OC4J removes the existing application (EAR/WAR) before redeploying the new EAR. This means, for example, that attempts to access an HTML file that was included in the previous application, but not the new one, will result in "File Not Found" errors.

Also note that a redeployed WAR file overlays the previously expanded WAR, meaning that some older files may persist in the new deployment and will need to be deleted. For example, static HTML files from the previous deployment that are not included in the new WAR may continue to reside in the expanded WAR directory structure, and we would have to be manually deleted.

Impact of "Hot Deploying" an Application

The term *hot deployment* refers to the process of deploying archive files - EARs, WARs, JARs, etc. - and their associated XML descriptor files on a production application server without re-starting or "bouncing" the server.

When an EAR is redeployed or "hot redeployed" on a running OC4J instance, the status of the classes loaded in the JVM from the previous application may vary. In some cases a classloader may recognize that a class or JAR file in the file system has changed, and reload the class or libary. In other cases whether a new class definition is loaded may depend on whether the JVM tuning allows the garbage collector to flush the existing class definition.

Issues may also exist with respect to serialized objects containing session data. If the class related to a session object changes, it may not be possible to cast the generic session object back to the class, since the class has changed and its variables may occupy a different memory footprint. This may result in lost session data.

Deploying a new Web module to an active OC4J instance also has a negative impact on exising sessions. Specifically, the HTTP sessions for every Web application running within the server instance will be lost by default.

You can avoid this issue on non-clustered OC4J instances by defining a "persistence directory" in each Web application's `orion-web.xml` file. Existing HTTP sessions will be stored in this temporary location across application deployments.

Specify a relative path to this directory as the value of the `persistence-path` attribute in the root `<orion-web-app>` element within each `orion-web.xml` file. For example:

```
<orion-web-app ...
  persistence-path="persistDir"
```

```
  ...>
</orion-web-app>
```

Note that this feature is not applicable for OC4J instances within a clustered environment.

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.

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 and Development

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
- 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

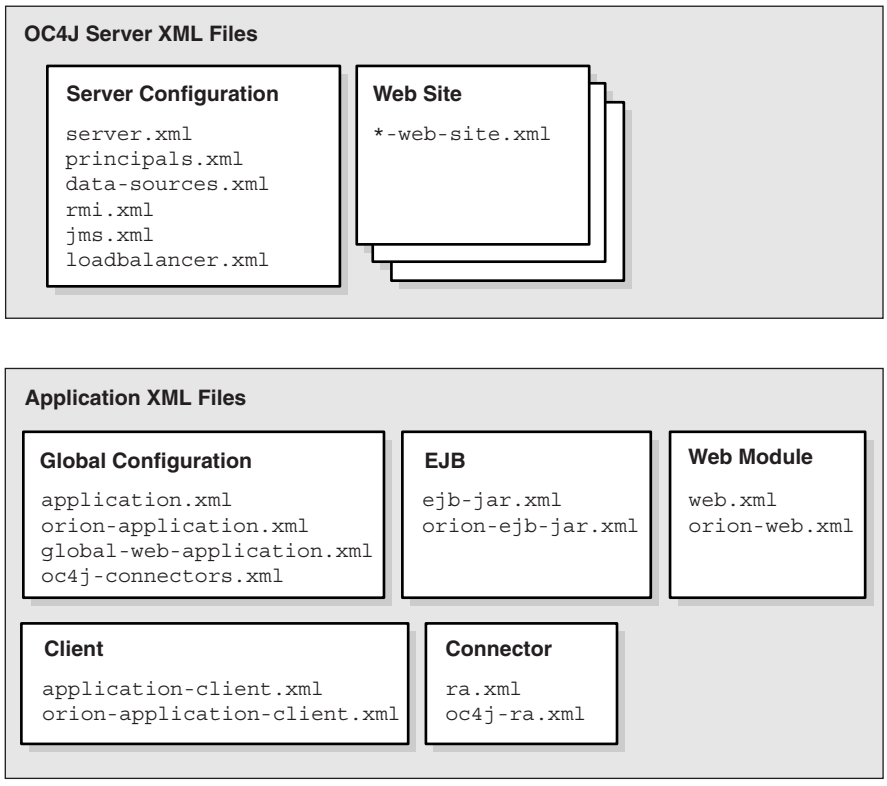
Because OC4J is configured solely through XML files, you must understand the role and method for a set of XML files. Each XML file exists to satisfy a certain role; thus, if you have need of that role, you will understand which XML file to modify and maintain.

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.
`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.

Table 2–1 (Cont.) OC4J Features and Components

XML Configuration File	Features/Components

`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.

`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

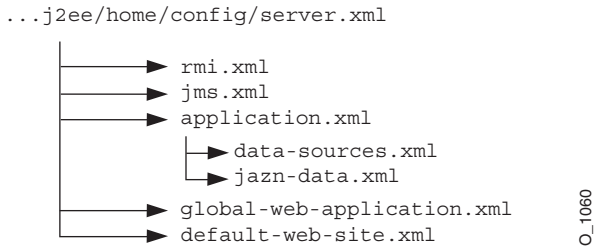
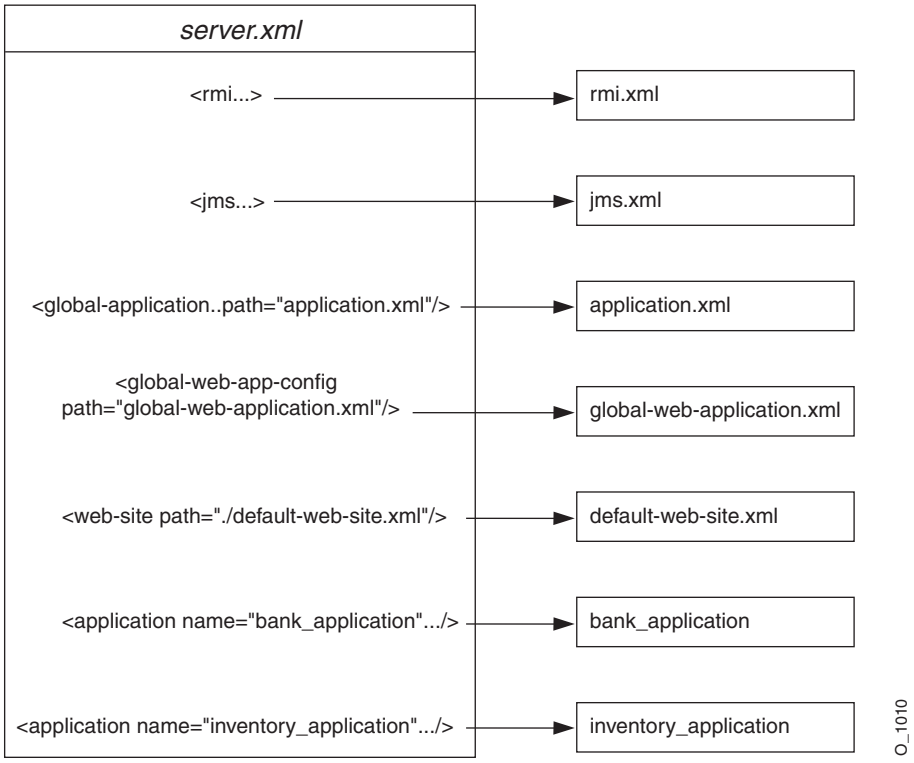


Figure 2–3 demonstrates how the `server.xml` points to other XML configuration files. For each XML file, the location can be the full path or a path relative to the location of where the `server.xml` file exists. In addition, the name of the XML file can be any name, as long as the contents of the file 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 2-7 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 B-6.

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 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 1-14 or "Building and Deploying Within a Directory" on page 2-8 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`".

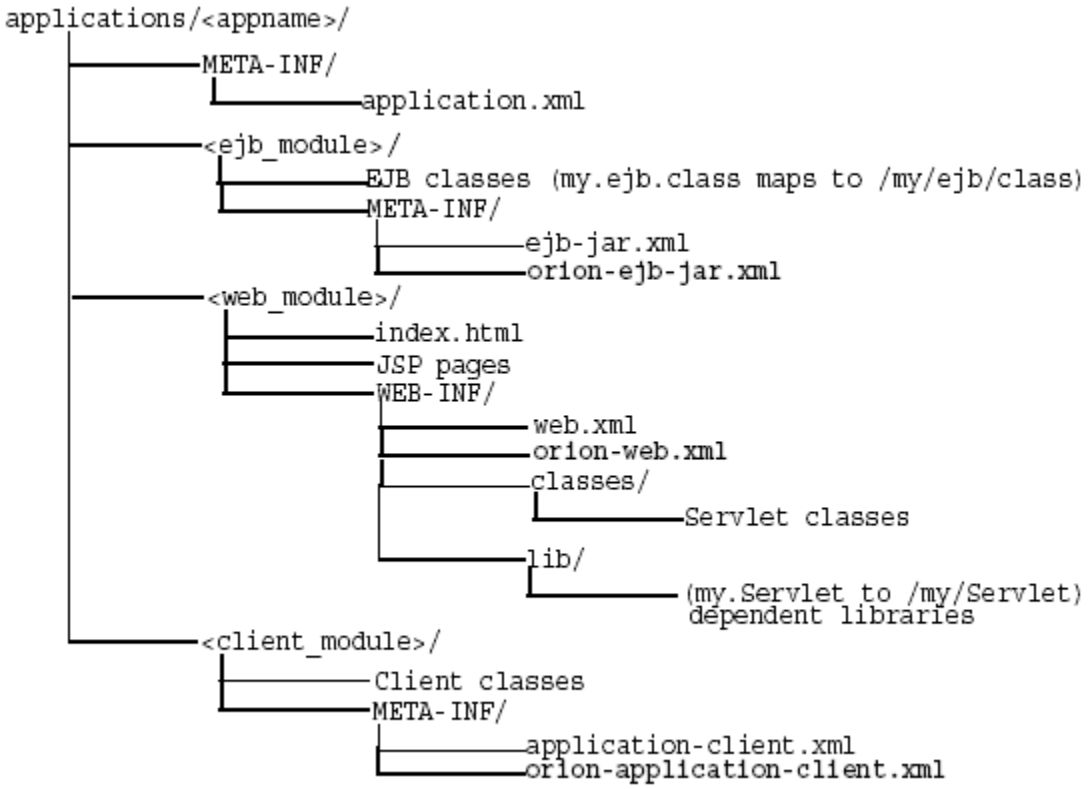
Note: Wait for automatic startup to complete before trying to access the client. The client fails on lookup if it tries to access before the completion of these processes.

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 Development Application Directory Structure, 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



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 2-8. 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.

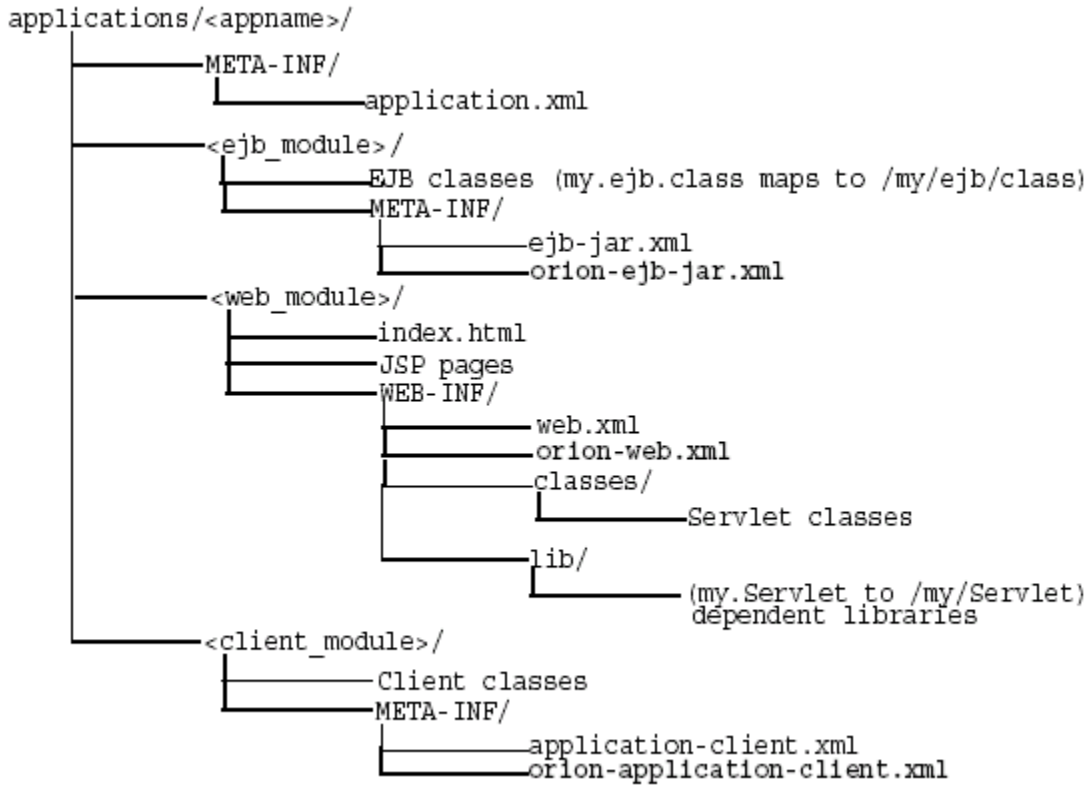
When OC4J does not check for updates, redeploy by either using the `admin.jar` command-line tool or restarting the OC4J server manually. See "Options for the OC4J Administration Management JAR" on page B-26 for a description of the `-deploy` option.

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

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 2-13 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 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 B-26 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

Thread pools create and maintain a "queue" of threads for use by an OC4J process. Re-using existing threads rather than creating new threads on demand improves performance and reduces the burden on the JVM and underlying operating system.

By default, a single thread pool is created at OC4J startup. New threads are created and added to the pool on an as-needed basis. As each thread is released, it is returned to the pool to wait until it is needed by a new request. Idle threads In the pool are used first before a new thread is spawned.

Threads in the pool are automatically destroyed after 10 minutes of inactivity. There is no limit on the number of threads that can be created in this configuration.

The default configuration should by sufficient for most OC4J usage scenarios. However, you can optionally modify the single thread pool created by default through the `min`, `max`, `queue`, and `keepAlive` attributes of the `<global-thread-pool>` element in the `server.xml` file.

Alternatively, you can create two thread pools using `<global-thread-pool>`, with different types of threads divided among the pools:

- The *worker thread pool* contains worker threads used in processing RMI , HTTP and AJP requests, as well as MDB listener threads. These are process-intensive and use database resources.
- The *connection thread pool* contains threads such as listener threads, JDBC connection threads, RMI server and HTTP server connection threads, and background threads. These threads are typically not process intensive.

To create two pools, you must configure the `min`, `max`, `queue`, and `keepAlive` attributes for the worker thread pool and the `cx-min`, `cx-max`, `cx-queue`, and `cx-keepAlive` attributes for the connection thread pool. All of these attributes must be configured if creating pools; otherwise you will see the following error message:

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

See Table 2–2 on page 2-17 for descriptions of the attributes of `<global-thread-pool>`.

The following example initializes two thread pools for the OC4J process. Each contains a minimum of 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>
```

Table 2–2 below describes the attributes of the `<global-thread-pool>` element. Note that this element is not included in `server.xml` by default.

Table 2–2 Attributes of <global-thread-pool>

Attribute	Description
`min`	The minimum number of threads to create in the pool. By default, a minimum number of threads are preallocated and placed in the thread pool when the container starts. If you add the `<global-thread-element>` element to `server.xml`, the default value is set to `20`. The minimum value that can be specified is `10`.
`max`	The maximum number of threads that can be created in the pool. 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. The default is `40`.
`queue`	The maximum number of requests that can be kept in the queue. The default is `80`.

Table 2–2 (Cont.) Attributes of <global-thread-pool>

Attribute	Description
keepAlive	The length of time, in milliseconds, to keep a thread alive (idle) while waiting for a new request. After the timeout is reached, the thread is destroyed. To never destroy threads, set to -1. The default is 600000 milliseconds (10 minutes), which is also the minimum value allowed if not -1.
cx-min	The minimum number of threads to create in the connection thread pool. The minimum value that can be specified is 10.
cx-max	The maximum number of threads that can be created in the connection pool. The default is 40.
cx-queue	The maximum number of threads that can be kept in the queue in the connection pool. The default is 80.
cx-keepAlive	The length of time, in milliseconds, to keep a thread alive (idle) while waiting for a new request. After the timeout is reached, the thread is destroyed. To never destroy threads, set to -1. The default is 600000 milliseconds (10 minutes), which is also the minimum value allowed if not -1.
debug	If true, print the application server thread pool information to the console at startup. The default is false.

Additional notes on thread pool configuration:

- 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. However, this number should be small; the more threads you have, the more burden you put on the operating system and the garbage collector.
- The cx-min and cx-max attributes are relative to the number of the physical connections you have at any point in time. The cx-queue handles bursts in connection traffic.

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 2–2 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 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 10*g* 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:

- Uncommend the `<odl>` element within the `<log>` element in all XML files listed in Table 2–3, except for the `http-web-site.xml` file.
- Add 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.

Disabling Access Logging for a Web Application

In the OC4J 10.1.2 implementation, there is new functionality in Web site XML files to disable OC4J access logging (used to log requests to the Web site) on a per-module basis.

In general, text-based access logging is enabled for a Web site through the `<access-log>` subelement of the `<web-site>` element in the Web site XML file (such as `default-web-site.xml` or `http-web-site.xml`). Alternatively, Oracle Diagnostic Logging ("ODL-based access logging") is enabled for a Web site through the `<odl-access-log>` subelement of the `<web-site>` element.

As of the 10.1.2 release, you can disable text-based logging or ODL-based logging (as applicable) for a particular Web application (module) through the new `access-log` attribute of the `<web-app>` element for that Web application. The `<web-app>`

element is another subelement of <web-site>. A setting of access-log="false" for a Web application overrides any <access-log> or <odl-access-log> element to disable logging during the operation of that Web application.

The following example disables logging for the dms0 module of the default application, but leaves text-based logging enabled for the admin_web module:

```
<web-site ... >
...
   <web-app application="default" name="dms0" root="/dms0" access-log="false" />
   <web-app application="default" name="admin_web" root="/adminoc4j" />
   <access-log path="../log/http-web-access.log" />
...
</web-site>
```

> **Notes:** The default setting is access-log="true". With this setting, functionality is the same as for previous releases, with logging being determined solely through the presence or absence of an <access-log> or <odl-access-log> element.
>
> If there is no <access-log> or <odl-access-log> element in a Web site XML file, then logging is disabled anyway and access-log="false" for an application has no additional effect.

See the 10.1.2 release of the *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for related information about access logging.

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 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 B-33.

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 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. The following examples show the output with and without verbosity:

Example 2–3 Error Messages Displayed Without Veribosity

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

Example 2–4 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/MYSERVICE:
Started
jdbc:oracle:thin:@localhost:1521/MYSERVICE: 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/MYSERVICE
null: Connection XA XA OC4J Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE
allocated (Pool size: 0)
jdbc:oracle:thin:@localhost:1521/MYSERVICE: Opened connection
Created new physical connection: Pooled
oracle.jdbc.driver.OracleConnection@5f18
Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE: Connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 allocated (Pool size: 0)
Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE: 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/MYSERVICE to pool (Pool size: 1)
OC4J Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE: Cache timeout, closing
connection (Pool size: 0)
com.evermind.sql.OrionCMTDataSource/default/jdbc/OracleDS: Cache timeout,
closing connection (Pool size: 0)
```

3

Configuring 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 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 3-8 for details on the OracleAS JAAS Provider, provider types, and user managers. Also, see the Oracle Application Server Containers for J2EE Security Guide for details on the OracleAS JAAS Provider and provider types.

> **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 Oracle Application Server Java Authentication and Authorization Service (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 Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider(OracleAS JAAS 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 3-8 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 3-13 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 3–1 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 3–2 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>
```

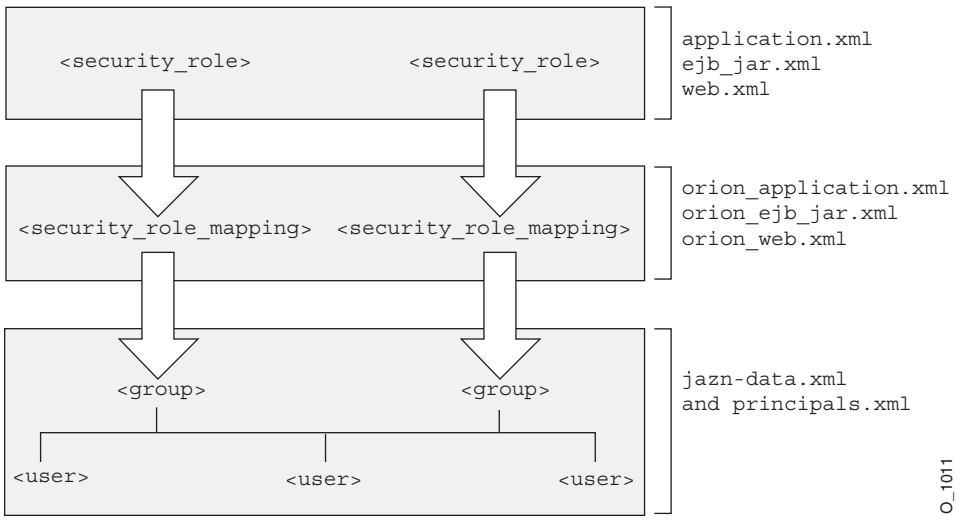
2. Define the bean and methods that this role can access in the `<method-permission>` element.

```
<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 3–1 illustrates this mapping.

Figure 3–1 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 3–3 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 3–1 lists the user managers that you can employ in OC4J security.

Table 3–1 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 OracleAS JAAS Provider policy (permissions) information. Figure 3–2 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:

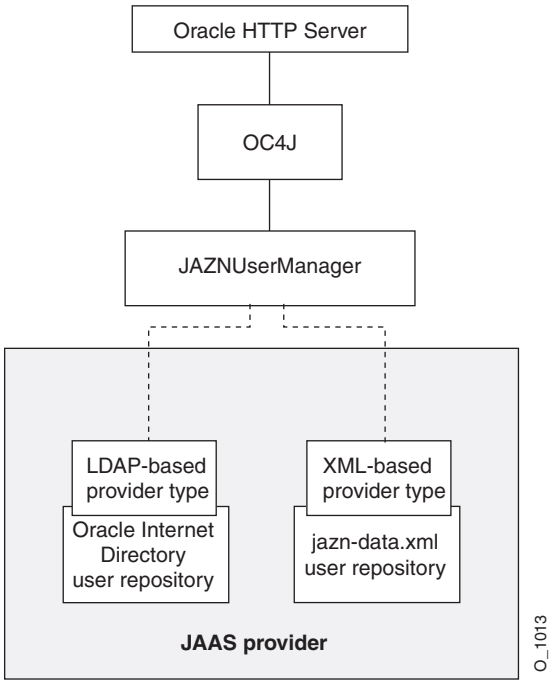
- LDAP-based

  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 3–2 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 3–2 describes the methods of this interface.

Table 3–2 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.

Table 3–2 (Cont.) Methods of the UserManager Interface

Method	Description
`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.
`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 3-2.

4. Create security constraints in your Web application.

   See "Authorization" on page 3-6.

Example 3–4 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. 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 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 3–5 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, configure 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 3–6 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
```

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

Troubleshooting OC4J

This appendix describes common problems that you may encounter when using OC4J and explains how to resolve them. It includes the following topics:

Problems and Solutions

This section describes common problems and solutions. It contains the following topics:

Unable to Start OC4J When Using JDK 1.3

Problem

OC4J fails to start when using JDK 1.3.

Solution

The failure to start is caused by a logging implementation dependency issue. The solution to this problem is to remove or comment out the following entry in `ORACLE_HOME/j2ee/home/config/server.xml`:

```
<j2ee-logging-config path="./j2ee-logging.xml" />
```

Unable to Restart OC4J After Abnormal Termination When OracleAS JMS is Active

Problem

When persistence is enabled in OracleAS JMS, the JMS server creates persistent queues/topics. It also creates lock files (`.lock`) associated with these queues/topics in the `/persistence` directory. If the JVM is terminated abnormally, such as with `kill -9`, the lock files are not deleted. This creates a condition in which OC4J cannot be restarted.

Solution

Manually delete all `.lock` files from the `/persistence` directory.

Stateful Replication Not Consistent Across OC4J instances

Problem

The common scenario is that failover is seen from OC4J instance A to instance B, but not back again from B to A.

Solution

OC4J does not require stateful replication to be set up globally for all applications, but instead allows each Web module to configure replication through the `<cluster-config>` element in its `orion-web.xml` descriptor file. Ensure that this element is populated correctly in each Web module's descriptor.

Note that OPMN-managed OC4J only supports clustering at the global level, and not at the application/module level.

Using A Non-Certified Version of the JDK for OC4J Only

Problem

In this scenario, you wish to use a later version of the JDK with OC4J than the version certified for use with all Oracle Application Server components. However, using a later version of the JDK globally for all components increases the risk of breaking certification.

Solution

To use the later JDK version with OC4J only, specify its location in the `<java-bin>` element in the `opmn.xml` configuration file. For example:

```
<module-data>
  <category id="start-parameters">
    <data id="java-bin" value="/myjavalocation/jdk/bin/java"/>
  </category>
</module-data>
```

java.lang.OutOfMemory Errors Thrown When Running OC4J

Problem

This error indicates that the heap size of the Java instance is lower than the memory required by applications running within OC4J.

Solution

Increase the heap size by setting `-Xmx` to the desired amount of memory in the `<java-option>` element in `opmn.xml`:

```
<module-data>
  <category id="start-parameters">
    <data id="java-options" value="-Xmx256M" />
  </category>
</module-data>
```

Alternatively, you can set a system property at OC4J startup:

```
java -Xmx256M -jar oc4j.jar
```

If running under Unix/Linux, verify that `ulimit` settings allow the JVM process to allocate this much memory.

Connection Timeouts Through a Stateful Firewall Affect System Performance

Problem

To improve performance the `mod_oc4j` component in each Oracle HTTP Server process maintains open TCP connections to the AJP port within each OC4J instance it sends requests to.

In situations where a firewall exists between OHS and OC4J, packages sent via AJP are rejected if the connections can be idle for periods in excess of the inactivity timeout of stateful firewalls.

However, the AJP socket is not closed; as long as the socket remains open, the worker thread is tied to it and is never returned to the thread pool. OC4J will continue to create more threads, and will eventually exhaust system resources.

Solution

The OHS TCP connection must be kept "alive" to avoid firewall timeout issues. This can be accomplished using a combination of OC4J configuration parameters and Apache runtime properties.

Set the following parameters in the `httpd.conf` or `mod_oc4j.conf` configuration files. Note that the value of `Oc4jConnTimeout` sets the length of inactivity, in seconds, before the session is considered inactive.

`Oc4jUserKeepalive on`

`Oc4jConnTimeout 12000` (or a similar value)

Also set the following AJP property at OC4J startup to enable OC4J to close AJP sockets in the event that a connection between OHS and OC4J is dropped due to a firewall timeout:

```
ajp.keepalive=true
```

For example:

```
java -Dajp.keepalive=true -jar oc4j.jar
```

OPMN-Managed OC4J Unable to Access EJB Resources Via the Default RMI Port

Problem

OC4J cannot access EJB resources via the default RMI port when running as a component of Oracle Application Server.

Solution

The most common cause is that a user more familiar with Standalone OC4J is reading the RMI port from `rmi.xml`, unaware that the value specified in this file is not used in an OPMN-managed Oracle Application Server environment.

OPMN-managed OC4J instances use dynamic RMI port assignments. The port value ranges are specified in the `<port>` element in `opmn.xml` or are determined using dynamic `opmn:ormi` lookup from the application client.

See the *Oracle Process Manager and Notification Server Administrator's Guide* for more information.

Application Performance Impacted by JVM Garbage Collection Pauses

Problem

An application running on OC4J appears unresponsive, with simple requests experiencing noticeable delays. The cause is that the JVM has crossed the low memory threshold and is running a full garbage collection to free up memory.

Solution

Consider using the *incremental low pause collector*, which avoids long major garbage collection pauses by doing portions of the major collection work at each minor collection. This collector (also known as the *train collector*) collects portions of the tenured generation - a memory pool holding objects that are typically collected in a major collection - at each minor collection. The result is shorter pauses spread over many minor collections.

Note that the incremental collector is even slower than the default tenured generation collector when considering overall throughput.

To use the incremental collector, the `-Xincgc` option must be passed in on the Java command line at application startup. Set the initial and maximum size of the young generation (object pool) to the same value using the `XX:NewSize` and `-XX:MaxNewSize` options. Set the initial and the maximum Java heap sizes to the same value using the `-Xms` and `-Xmx` options.

For example, to use this collector with a server with 1GB of physical memory:

```
java -server -Xincgc -XX:NewSize=64m -XX:MaxNewSize=64m -Xms512m -Xmx512m
```

For more information on garbage collection tuning, read *"Tuning Garbage Collection with the 1.4.2 JavaTM Virtual Machine"* which is available at `http://java.sun.com/docs/hotspot/gc1.4.2/`

Invalid or Unneeded Library Elements Degrade Performance

Problem

If the OC4J process memory is growing consistently during program execution, then you may have references to invalid symbolic links in your global `application.xml` file.

This problem is usually characterized by a growth in the C heap and not a growth in Java object memory, as one would see with a more traditional Java object memory leak. OC4J loads all resources defined in the `application.xml` file. If these links are invalid, then the C heap continues to grow, causing OC4J to run out of memory.

Solution

Ensure that all symbolic links are valid in `application.xml`, and restart OC4J.

In addition, keep the number of JAR files OC4J is configured to load to a minimum. Eliminate all unused JAR files from the configuration and from the directories OC4J is configured to search. OC4J searches all JAR files for classes and resources, thereby causing the file cache to use extra memory and processor time.

You can control the loading more precisely if your `<library>` elements in the `application.xml` file point to the individual JAR and ZIP files that are needed, instead of to the directories where they reside.

JSP Error: Tag Not Registered

Problem

This error occurs when a JSP attempts to call a tag that cannot be found within the OC4J server. The problem typically arises when one or more additional "well-known tag library locations" have been incompletely defined within OC4J.

Solution

Defining a well-known tag library location is a two-step process:

- The directory is defined in the `jsp-taglib-locations` attribute of the of the `<orion-web-app>` element in the `global-web-application.xml` file; and
- The directory is added to the path attribute of the `<library>` element in the `application.xml`.

The error typically indicates that the second step was not completed.

Alternatively, you can copy the JAR file containing the tag library to the default well-known tag library location, which is `ORACLE_HOME/j2ee/home/jsp/lib/taglib/`.

JSP Error: Zero-Length Class File

Problem

This error occurs when OC4J attempts to serve a JSP that failed to compile into a Java class. The cause is that the Java compiler either could not be loaded or ran out of memory, resulting in a 0 byte `.class` file.

Solution

Delete the 0 byte `.class` file. The class will be compiled the next time the JSP is requested.

JSP Error: Illegal use of <when>-style tag without <choose> as its direct parent

Problem

This error occurs when OC4J fails to serve a requested JSP that includes a JSP Standard Tag Library (JSTL) tag—in this case, the `<choose>` tag. The likely cause is that more than one version of the JSTL exists within the OC4J instance.

Solution

Delete the version of the JSTL installed by default with OC4J. This library is packaged as the `standard.jar` file in the `ORACLE_HOME/j2ee/home/jsp/lib/taglib` directory.

Need More Help?

You can search for additional solutions on the following Oracle support-oriented Web sites:

- Oracle Application Server Release Notes, available on the Oracle Technology Network at `http://www.oracle.com/technology/documentation/index.html`
- Oracle MetaLink, available at `http://metalink.oracle.com`

If you still cannot find a solution for the problem you are facing, please log a service request.

B

Additional Information

This appendix contains complete information about the following topics:

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

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 B-14 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 B-16 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 B-21 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 B-24 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, the following elements, which are listed alphabetically and not by DTD ordering, can be configured:

`<application>`

An application is a entity with its own set of users, Web applications, and EJB JAR files.

Attributes:

- `auto-start="true|false"` — Specifies whether the application should be automatically started when the OC4J server starts. The default is `true`. Setting `auto-start` to `false` is useful if you have multiple applications installed and want them to start on demand. This can improve general server startup time and resource usage.
- `deployment-directory="../application-deployments/myapp"` — Specifies a directory to store application deployment information. If none is

specified (the default), OC4J looks in the global `deployment-directory`, and if none exists there, it stores the information inside the EAR file. The path can be relative or absolute. If relative, the path should be relative to the location of the `server.xml` file.

- `name="anApplication"` — Specifies the name used to reference the application.
- `parent="anotherApplication"` — The name of the optional parent application. The default is the global application. Children see the namespace of its parent application. This is used to share services such as EJBs among multiple applications.
- `path=".../applications/myApplication.ear" />` — The path to the EAR file containing the application code. In this example, the EAR file is named `myApplication.ear`.

`<compiler>`

This element is deprecated for version 9.0.4 and forward. See the `<java-compiler>` element for the alternative. For previous releases, it specifies an alternative compiler (such as Jikes) for EJB/JSP compiling.

Attributes:

- `classpath="/my/rt.jar"` — Specifies an alternative/additional classpath when compiling. Some compilers need an additional classpath (such as Jikes, which needs the `rt.jar` file of the Java 2 VM to be included).
- `executable="jikes" />` — The name of the compiler executable to use, such as Jikes or JVC.

`<cluster>`

Cluster settings for this server.

Attribute:

- `id="123" />` — The unique cluster ID of the server.

`<execution-order>`

Defines the ordering of how the startup classes are executed. Value is an integer. OC4J loads from 0 on up. If duplicate numbers, OC4J chooses the ordering for those classes.

`<global-application>`

The default application for this server. This acts as a parent to other applications in terms of object visibility.

Attributes:

- `name="default"` — Specifies the application.
- `path=".../application.xml" />` — Specifies the path to the global `application.xml` file, which contains the settings for the default application. An `application.xml` file exists for each application as the standard J2EE application descriptor file, which is different than this file. This `application.xml` may have the same name, but it exists to provide global settings for all J2EE applications.

`<global-thread-pool>`

You can specify unbounded, one, or two thread pools for an OC4J process through this element. If you do not specify this element, then an infinite number of threads can be created. See "Thread Pool Settings" on page 2-16 for a full description.

Attributes:

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

  Value is a long. The default is 600000 milliseconds.
- `cx-min` —The minimum number of connection threads that OC4J can simultaneously execute. Value is an integer. The default is 20. The minimum value you can set this to is 10.
- `cx-max` —The maximum number of connection threads that OC4J can simultaneously execute. Value is an integer. The default is 40.
- `cx-queue` —The maximum number of connection requests that can be kept in the queue. Value is an integer. The default is 80.
- `cx-keepAlive` —The number of milliseconds to keep a connection thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains alive. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one.

  Value is a long. The default is 600000 milliseconds.
- `debug` —If true, print the application server thread pool information at startup. The default is false.

`<global-web-app-config>`

Attributes:

- `path`— The path where the `web-application.xml` file is located.

```
path=".../web-application.xml" />
```

`<init-library>`

Attributes:

- `path`— The path in which the startup and shutdown classes are located. The path indicates the directory in which the class resides or the directory and JAR filename of the JAR where the class is archived. If more than one directory or JAR file exists, then supply an `<init-library>` element for each directory and JAR filename.

```
<init-library path="../xxx">
```

`<init-param>`

Attributes:

- Defines the key-value pairs of the parameters to pass into the startup class.

<javacache-config>

Attributes:

- path—Specifies the path to the javacache.xml file.

```
<javacache-config path="../../../javacache/admin/javacache.xml" />
```

<java-compiler>

You can specify an alternative compiler—either in or out of process—for your JSP and EJB compilation. The default compiler is an out of process javac compiler found in the JDK bin directory.

Attributes:

- name—Specify the name of the compiler to use. Valid compiler names are as follows:
  - for in-process compilers—modern, classic, javac or ojc
  - for out-of-process compilers (forked)—modern, javac, ojc, or jikes

    These names are defined as follows:

    * javac—the standard compiler name for all JDKs.
    * classic—the standard compiler of JDK 1.1/1.2.
    * modern—the standard compiler of JDK 1.3/1.4.
    * jikes—the Jikes compiler.
    * ojc—The Oracle Java compiler.
- in-process—If true, the compiler is to run in process. If false, the compiler runs out of the process. Most compilers can execute both in and out of the process. The exceptions are as follows:
  - The classic compiler cannot run out of the process; thus, the in-process attribute is always true.
  - The jikes compiler cannot run in process; thus, the in-process attribute is always false.
- encoding—Specify the type of character encoding for the source file, such as UTF-8, EUCJIS, or SJIS. Encoding is only supported by the javac compiler. The default is determined by the language version of the JVM that is installed.
- bindir—Provide the absolute path to the compiler directory. You do not need to specify this attribute for javac, modern, or classic as the JDK bin directory is searched for this compiler.

  The syntax is specific to the operating system platform:
  - Sun Microsystems Solaris example—If you are using jikes, which is in /usr/local/bin/jikes, then specify the following:

    ```
    name="jikes"
    bindir="/usr/local/bin"
    ```

  - Windows example—To specify jikes, which is located in c:\jdk1.3.1\bin\jikes.exe, specify the following:

```
name="jikes"
bindir="c:\\jdk1.3.1\\bin"
```

- `extdirs`—Specifies extension directories that the compilation uses to compile against. The default is your JDK extension directories. Multiple directories can be specified, each separated by a colon. Each `JAR` archive in the specified directories are searched for class files. You can specify certain directories to be searched by modifying the `-Djava.ext.dirs` system property. The `jikes` compiler requires that extension directories are specified in either this attribute or in the `-Djava.ext.dirs` system property.

The following are four examples of how to define alternate compilers in this element:

```
<java-compiler name="jikes" bindir="C:\java\jikes\bin"
in-process="false" />
<java-compiler name="ojc" bindir="C:\java\jdev\jdev\bin"
in-process="false"/>
<java-compiler name="classic" in-process="true" />
<java-compiler name="modern" in-process="true" />
```

<jms-config>

Attribute:

- `path`— Specifies the path to the `jms.xml` file.

  ```
  path=".../jms.xml"
  ```

<log>

<file>

Attribute:

- `path="../log/server.log"` — Specifies a relative or absolute path to a file where log events are stored.

<mail>

An e-mail address where log events are forwarded. You must also specify a valid mail-session if you use this option.

Attribute:

- `address="my@mail.address"` — Specifies the mail address.

<odl>

The ODL log entries are each written out in XML format in its respective log file. The log files have a maximum limit. When the limit is reached, the log files are overwritten.

When you enable ODL logging, each message goes into its respective log file, named `logN.xml`, where N is a number starting at one. The first log message starts the log file, `log1.xml`. When the log file size maximum is reached, the second log file is opened to continue the logging, `log2.xml`. When the last logfile is full, the first log file, `log1.xml` is erased and a new one is opened for the new messages. Thus, your log files are constantly rolling over and do not encroach on your disk space.

Attributes:

- `path`: Path and folder name of the log folder for this area. You can use an absolute path or a path relative to where the configuration XML file exists, which is normally in the `j2ee/home/config` directory. This denotes where the log files will reside for the feature that the XML configuration file is concerned with. For

example, modifying this element in the `server.xml` file denotes where the server log files are written.

- `max-file-size`: The maximum size in KB of each individual log file.
- `max-directory-size`: The maximum size of the directory in KB. The default directory size is 10 MB.

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

`<max-http-connections>`

Used to define the maximum number of concurrent connections any given Web site can accept at a single point in time. If text exists inside the tag, it is used as a redirect-URL when the limit is reached.

Attributes:

- `max-connections-queue-timeout="10"` — When the maximum number of connections are reached, this is the number of seconds that can pass before the connections are dropped and a message is returned to the client stating that the server is either busy or connections will be redirected. The default is 10 seconds.
- `socket-backlog` — The number of connections to queue up before denying connections at the socket level. The default is 30.
- `value` — The maximum number of connections.

`<metric-collector>`

The `<metric-collector>` element specifies that OC4J sends a metric between 0 and 100, inclusive, to `mod_oc4j` so that `mod_oc4j` can make routing decisions to load-balance incoming requests to a list of available OC4J instances. The metric sent has a relative value only, where 0 means that the OC4J instance is very busy and 100 means that the OC4J instance is available (not busy). When configured for metric load balancing, `mod_oc4j` routes first to the OC4J instance with the greater value.

The metric sent from OC4J to `mod_oc4j` is used only when metric-based load balancing is specified for `mod_oc4j` and when OC4J runs in an Oracle Application Server environment.

If you specify metric-based load balancing in `mod_oc4j` and do not specify the `<metric-collector>` element in `server.xml`, then `mod_oc4j` expects OC4J to send metrics, but OC4J does not send metrics. In this case, `mod_oc4j` reports the following warning message:

```
No run time metrics for oc4j(opmnid=%s) in notification Oc4jSelectMethod is
configured to use run time metrics, please make sure OC4J side is configured
accordingly. Default to 50.
```

In this case, `mod_oc4j` uses the value 50 for each of the OC4J processes and continues.

Likewise, if you specify the `<metric-collector>` element in `server.xml`, but do not specify metric-based load balancing in `mod_oc4j`, then OC4J sends metrics but `mod_oc4j` is not configured to receive metrics. In this case, `mod_oc4j` ignores the metrics and uses whatever the configured method is for load balancing. You specify the load balancing method with `Oc4jSelectMethod`. If no `Oc4jSelectMethod` is specified, then `mod_oc4j` uses the default, which is roundrobin.

The `<metric-collector>` element takes the following attribute: `classname`.

The `classname` attribute defines an interface for gathering and calculating a server-wide metric. Use `oracle.oc4j.server.DMSMetricCollector` for the

classname attribute when using a DMS-noun-based metric collector. A DMSMetricCollector instance takes several parameters. Details for the values for these parameters are available in the *Oracle Application Server 10g Performance Guide*.

For example:

```
<metric-collector classname="oracle.oc4j.server.DMSMetricCollector">
   <init-param>
     <param-name>
       dms-noun
     </param-name>
     <param-value>
       /oc4j/default/WEBs/processRequest.time
     </param-value>
   </init-param>
   <init-param>
     <param-name>
       history-proportion
     </param-name>
     <param-value>
       0.2
     </param-value>
   </init-param>
   <init-param>
     <param-name>
       debug
     </param-name>
     <param-value>
       false
     </param-value>
   </init-param>
</metric-collector>
```

For details on using the <metric-collector> element and using metric-based load balancing with mod_oc4j, see the *Oracle Application Server 10g Performance Guide*.

<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

This section provides an overview of the J2EE application deployment descriptor 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

This section describes the OC4J-specific application deployment descriptor 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>/j2ee/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:

- 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 B–1. See the *Oracle Application Server Containers for J2EE Security Guide* for more information.

Table B–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

This section describes the J2EE application client deployment descriptor 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

This section provides an overview of the OC4J-specific application client deployment descriptor 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 B–2 lists all the `oc4j.jar` command-line options:

Table B–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. See Table B–9, " stdout/stderr Archive Management Properties" on page B-36 for additional system properties that can be set to manage `stdout` files.
`-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. See Table B–9, " stdout/stderr Archive Management Properties" on page B-36 for additional system properties that can be set to manage `stderr` files.
-verbosity	Define an integer between 1 and 10 to set the verbosity level of the message output. Example: `-verbosity 10`.
`-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 B–3.
- Application deployment described in Table B–4.

- Web site administration described in Table B–5.
- Data source administration described in Table B–6.

General OC4J Administration

Table B–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 B–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 B–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 B–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.
`-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.

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

Option	Description
`-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.
`-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 B–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 B–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 B–5 (Cont.) Options for Web Site Administration

-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 B–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 B–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.

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

-application Option	Description
`-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.
`-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 B–6 (Cont.) Options For Application And Data Source Management

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

Table B–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
`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 that 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 checks the database. If you do not set this property and you try to shutdown OC4J using CTRL-C, then the OC4J process will hang for at least 2.5 hours.

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

-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.
`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 B–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.
`http.cluster.debug`	Default: false. HTTP clustering debug messages.
`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

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

-D Debug System Properties	Description
`http.error.debug`	Default: false. If true, prints all HTTP errors
`http.virtualdirectory.debug`	Default:false. If true, print the enforced virtual directory mappings upon startup.
`debug.http.contentLength`	Default: false. If true, print explicit content-length calls as well as extra sendError information.
`ejb.cluster.debug`	Default: false. EJB clustering debug messages.
`cluster.debug`	Default: false. Clustering debug messages.
`jms.debug`	Default: false. JMS debug messages.
`multicast.debug`	Default: false. Multicast debug messages.
`rmi.debug`	Default: false. RMI debug messages.
`transaction.debug`	Default: false. If true, prints debug messages for JTA events.
`rmi.verbose`	Default: false. RMI verbose information.
`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 2-23.

Table B–9 stdout/stderr Archive Management Properties

Property	Description
`stdstream.filesize=<max_file_size>`	The maximum size any file in the archive will be allowed to grow to, in megabytes. Files are rotated when this maximum is reached.
`stdstream.filenumber=<max_files>`	The maximum number of files to keep as archives. The oldest file will be automatically deleted when the limit is exceeded.
`stdstream.rotatetime=<HH:mm>`	The time at which the log file will be rotated each day.

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 1-7 to learn how to modify the XML configuration files for the FAQ application demo.

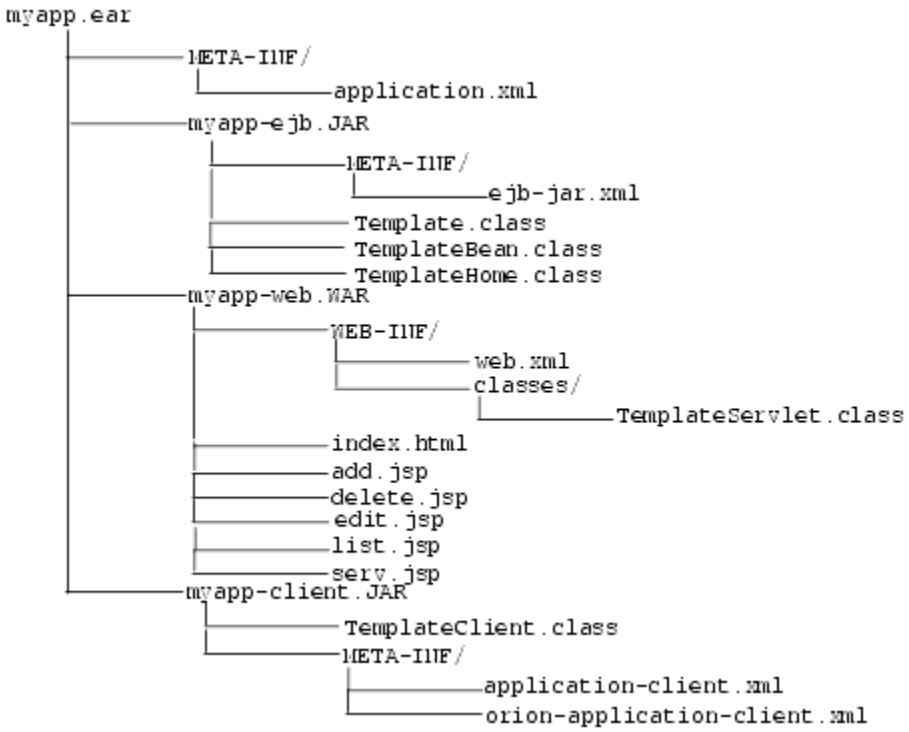
- J2EE Application XML Configuration Example
- Deploying Example

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 Application EAR Structure. Notice that you can separate all the configuration files into logical directories within the application directory.

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.ApplicationClientInitialContext
Factory
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/MYSERVICE
-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
```

C

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

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.
*/
```

Index

Symbols

A

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

X