Oracle9*i*

Servlet Engine Developer's Guide

Release 1 (9.0.1)

June 2001

Part No. A90213-01

ORACLE®

Oracle9*i* Servlet Engine Developer's Guide, Release 1 (9.0.1)

Part No. A90213-01

Copyright © 2000, 2001, Oracle Corporation. All rights reserved.

Primary Authors: Tim Smith and Brian Wright

Secondary Author: John Russell

Contributors: Ellen Barnes, Hal Hildebrand, Sunil Kunisetty, Wendy Liau, Angela Long, Kuassi Mensah, Jasen Minton, Kannan Muthukkaruppan, Joyce Yang, and Susan Yorke-Kraft

Contents

3   OSE Configuration and Examples

4   An Apache Module for OSE

5   Configuring mod_ose

8 Oracle WAR Deployment

Send Us Your Comments

Oracle9*i* Servlet Engine Developer's Guide, Release 1 (9.0.1)

Part No. A90213-01

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

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

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

- Electronic mail: jpgcomment_us@oracle.com
- FAX: (650) 506-7225   Attn: Java Platform Group, Information Development Manager
- Postal service:
  Oracle Corporation
  Java Platform Group, Information Development Manager
  500 Oracle Parkway, Mailstop 4op9
  Redwood Shores, CA   94065
  USA

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

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

x

Preface

Who Should Read This Guide

This Guide has been written for several audiences. They include

- Java developers

- Oracle developers not familiar with Java

- managers

Java developers who are used to a Java J2EE environment should have very little trouble developing applications for the Oracle Servlet Engine. However, since the OSE runs in a virtual JVM in an Oracle database session, there are concepts and procedures discussed in this Guide that you should understand to facilitate OSE application development.

Oracle database developers who are familiar with PL/SQL and other non-Java programming environments should read the overview of Java and object-oriented concepts discussed in the first part of this Guide. For more information about Java, see "Information Resources" on page -xiv.

Chapters 1 and 2 of this Guide should give managers a good overview of the capabilities of the Oracle Servlet Engine (OSE). As a manager, you might have purchased Oracle9*i* for reasons other than Java development within the database. But if you do want to know more about Oracle9*i* Java features, see the *Oracle9i Java Developer's Guide* for a management perspective on Java development.

The OSE is different from a Web server such as Apache with a servlet engine, e.g. Tomcat. In these cases, the servlet server runs in a JVM in an operating system process, not in a JVM in a database session. There are some special concepts that apply to the OSE in the Oracle JVM (OJVM), such as the method of database access, how Java servlet classes and JavaServer Pages are loaded, and the way that sessions

are handled. Becoming familiar with the OSE/OJVM way of doing things is a prerequisite to developing successful applications of the OSE.

If you are developing applications that primarily use JavaServer Pages, read the *Oracle Support for JavaServer Pages Developer's Guide and Reference.*

How to Read This Guide

This Guide describes the Oracle Servlet Engine when it runs in an Oracle server. This Guide does not attempt to teach servlet programming, nor does it document the Java Servlet API. To learn about these topics, see the documentation available from Sun Microsystems, or look at one of the trade books on servlet programming.

This Guide contains the following chapters and appendices:

Chapter 1, "Oracle Servlet Engine Overview"	Introduces the product and describes some of its advantages over alternative servlet engines.
Chapter 2, "Oracle Servlet Engine Concepts"	Describes the way the OSE operates, and how it differs from other servlet engines.
Chapter 3, "OSE Configuration and Examples"	Provides specific instructions on configuring OSE services, domains, and servlet contexts. Includes many examples.
Chapter 4, "An Apache Module for OSE"	Describes the Oracle HTTP Server module that directs requests for servlets from Apache to the OSE.
Chapter 5, "Configuring mod_ose"	Tells you how to configure `mod_ose`.
Chapter 6, "Calling EJBs"	Describes a demonstration application that calls an EJB from a servlet.
Chapter 7, "Oracle Servlet Engine Security"	Covers all aspects of security with the OSE.
Chapter 8, "Oracle WAR Deployment"	Describes how to create servlet contexts, and deploy servlets within them, using a Web Archive file.
Chapter 9, "Writing PL/SQL Servlets"	Describes how to write servlets using PL/SQL and the PL/SQL gateway.
Appendix A, "Abbreviations and Acronyms"	A comprehensive list of network- and Java-related acronyms.

Oracle8*i* Release 3 Users

This Guide has been rewritten extensively from the Oracle8*i* Release 3 version of the Guide.

Nevertheless, if you are using the Oracle8*i* Release 3 version of the OSE, almost all of the descriptions and instructions in this Guide also apply to your release. There have been a few additions in functionality for the Oracle9*i* OSE in OJVM release, mostly in the area of improvements for the Apache module `mod_ose`. New Oracle9*i* functionality is indicated as such.

Conventions

This book generally uses UNIX syntax for file paths and shell variables. In most cases file names and directory names are the same for Windows NT, unless otherwise noted. The notation `$ORACLE_HOME` indicates the full path of the Oracle home directory. It is equivalent functionally to the Windows NT environment variable `%ORACLE_HOME%`, though of course the Oracle installation paths are different between NT and Solaris or other UNIX flavors.

This Guide uses the following additional conventions.

Convention	Meaning
italicized regular text	Italicized regular text is used for emphasis or to indicate a term that is being defined or will be defined shortly.
...	Horizontal ellipsis points in sample code indicate the omission of a statement or statements or part of a statement. This is done when you would normally expect additional statements or code to appear, but such statements or code would not be related to the example.
code text	Code text (Courier font) within regular text indicates class names, object names, method names, variable names, Java types, Oracle datatypes, file names, URL or URI fragments, and directory names.
%	At the beginning of a command, indicates an operating system shell prompt.
$	At the beginning of a command, indicates an Oracle JVM session shell prompt.
SQL>	At the beginning of a command, indicates a SQL*Plus prompt.

Information Resources

To understand the Oracle JVM programming environment, see the *Oracle9i Java Developer's Guide.*

Managing the OSE requires that you issue a number of commands. Some of these are issued at the operating system level, at the shell (command processor) level. For example, to load a Java servlet class into Oracle9*i* you use the `loadjava` command from the shell prompt. To publish a servlet class to the OSE, you use the OJVM *session shell*, and in the session shell you issue the `publishservlet` command.

Each of these commands is described briefly in this Guide. But for a complete description of the commands and all their parameters, see the *Oracle9i Java Tools Reference.*

The following table lists some sources of information about Java, Java Web servers, and other related topics that are available on the World Wide Web.

Location	Description
`http://www.oracle.com/java`	The latest offerings, updates, and news for Java within the Oracle9*i* database. This site contains FAQs, updated JDBC drivers, SQLJ reference implementations, and white papers that detail Java application development. In addition, you can download Java tools (on a try-and-buy basis) from this site.
`http://java.sun.com/`	The Sun Microsystems Web site that is the central source for Java. This site contains Java products and information, such as Javadoc, tutorials, book recommendations, and the Java Developer's Kit (JDK).
`http://java.sun.com/docs/books/jls` `http://java.sun.com/docs/books/vmspec`	The Oracle9*i* JVM implements the Java Language specification (JLS) and the Java virtual machine (JVM) specification, as described here.
`http://java.sun.com/products/...`	The Oracle Servlet Engine is compliant with the Servlet 2.2 specification, part of the J2EE specification. These specifications are available here.
`http://www.apache.org`	Information and documentation on the Apache Web server (a component of the Oracle HTTP Server). The Oracle HTTP Server is used together with mod_ose (see Chapter 4, "An Apache Module for OSE") to access the OJVM OSE.

Location	Description
`comp.lang.java.programmer` `comp.lang.java.databases`	Internet newsgroups can be a valuable source of information on Java from other Java developers. We recommend that you monitor these two newsgroups. Note: Oracle monitors activity on some of these newsgroups and posts responses to Oracle-specific issues.

Your local or on-line bookseller has many useful Java references. You can find another listing of materials that are helpful to beginners, and that you can use as general references, in the *Oracle Support for JavaServer Pages Developer's Guide and Reference*.

Documentation Accessibility

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

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

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

1

Oracle Servlet Engine Overview

This chapter introduces Java servlets and the Oracle Servlet Engine (OSE). It describes the basics of the OSE design and operation, and shows how to combine the OSE with other products to get speed and scalability for your Web application. Later chapters of this guide describe in detail how to develop an application for the OSE and how to administer the OSE.

This chapter covers the following topics:

- Web Servers and Servlet Engines
- A Brief Introduction to Servlets
- About the Oracle Servlet Engine
- The Oracle Servlet Engine Namespace
- Hosting a Web Application
- Steps in Developing a Web Application

Web Servers and Servlet Engines

A *Web server* is simply a program that runs on a server platform, and sends files back to clients that request them. Client requests use the HTTP protocol, and the files are typically pages written using HTML. The client is usually a Web browser such as Netscape Communicator or Internet Explorer, but any client or even a server-side program can send HTTP requests to a Web server.

Apache is a well-known Web server. The *Oracle HTTP Server* is the Oracle extended version of the Apache Web server.

A Web server takes the URL sent by the client, and finds a file on a server that it can send back to the client, as an HTTP response. But some clients require information that has dynamic content, that can change over time, or that varies depending on information that the client sends to the server as part of the request.

To serve dynamic content, a Web server can include or call on a *servlet engine*, which is a program that takes a client request for a dynamic page and activates a Java class—the servlet—that provides the dynamic content. The content is typically derived from queries to the database, normally using a JDBC connection.

A servlet engine is almost always separate architecturally from the Web server. For example, the Apache Web server can use the JServ servlet engine, which supports the Servlet 2.0 specification, or the newer Tomcat servlet engine, which supports the Java Servlet specification version 2.2.

The Oracle Servlet Engine running in the Oracle Java Virtual Machine (OJVM) is a servlet engine that works in conjunction with the Oracle HTTP Server, using the `mod_ose` module.

A Brief Introduction to Servlets

Traditional Web applications have consisted of static pages written in HTML, combined with a few forms to provide for user input and to supply dynamic content. The dynamic content was usually enabled using Common Gateway Interface (CGI) scripts, often written in a scripting language such as Perl or TCL.

Although static HTML pages are a central part of most Web applications, using CGI scripts to supply dynamic content turned out to be very inefficient. For example, in the traditional model each request to a script requires spawning a separate process in the Web server, thus limiting the scalability of the Web application.

Servlets were designed to overcome these limitations, and to provide efficient access to data stored in a database server.

What Is a Servlet?

A servlet is a Java class that executes on a server. It is the server-side version of a Java applet. You write a servlet using standard, platform-independent Java, and compile it using a Java byte code compiler. Compile your servlets using a Sun JDK 1.2-compliant Java compiler. After compilation, the servlet is loaded into an Oracle9*i* server, either on the data tier or perhaps into a read-only database on the middle tier, and is published in the server. The servlet runs inside the servlet engine.

After a few configuration steps, the servlet is accessible to a Web browser or other client that uses the HTTP protocol. The client can query the servlet and receive the servlet response, for display or other processing.

As with Enterprise JavaBeans (EJBs), servlets are invoked indirectly. However, to call an EJB method directly from a client, you must activate an ORB and use the session IIOP or RMI/IIOP protocol to call methods on the bean. The client activates a servlet using an HTTP request, from a Web browser or other client application. Unlike EJBs, you do not call methods on a servlet, because a servlet has a single entry point. Instead, the client activates the servlet, which then processes the HTTP request, and sends a response back to the client.

Data can be passed to the servlet in the request URL. A common scenario is for the client to activate a Web page, written in HTML. The Web page collects some information from the user, then calls a servlet in the middle tier or in the data tier to obtain information from the database.

Servlets are close to the data. This is especially true for the Oracle9*i* platform. In that platform, servlets reside in the server JVM, and have very fast in-memory access to SQL data through a special internal JDBC driver.

Kinds of Servlets

A servlet can be either *stateless* or it can have state.

A stateless servlet simply processes requests, sending the response back to the client. It has no knowledge of other requests, or any data that they might contain. This follows the HTTP protocol, which is a stateless protocol. The only certain way that two stateless requests can share data is for the servlet to store the data of one request in the database, and retrieve it when needed for other requests. There is no common Java memory that stateless servlets can share.

However, for many applications it is very convenient to be able to handle separate but related HTTP requests in a way that preserves their relatedness. For example, when you connect to a Web site to purchase some items, the server maintains a shopping cart of your selections. But to keep the shopping cart available, the server has to know who the client is. The client might make many requests to place items in the cart, and for the server and the client to agree on which client goes with which cart requires that some state be maintained.

Servlet engines have a mechanism for handling related requests: the *session.* A session encompasses a series of related requests that come from the same client, within a specified time period. Session state mechanisms relate the separate requests from the same client. The most successful session state mechanism has been *cookies*, developed by Netscape.

The server sends a cookie to the client that identifies the session, and the client (a Web browser, for example) returns the cookie with each new request to the same domain. There are also other mechanisms to relate requests that are used, for browsers that do not support cookies, or in cases where the client has turned off cookies. Two of these mechanisms are URL rewriting and Hidden Form Fields. The OSE/OJVM supports URL rewriting whenever cookies are not available.

The J2EE Servlet 2.2 specification defines a session interface. This allows stateful servlets to persist information on the server, by using an `HttpSession` object. Thus any servlet that is written to the 2.2 specification can maintain and process session state in the same way.

Advantages of Servlets

A Java servlet offers the following advantages over scripts and other methods that provide database access for Web applications:

- A servlet is written in platform-independent Java, making the application that uses it portable to a wide variety of hardware and software platforms.

- The full range of Java APIs is available to a servlet, such as the *Java Naming and Directory Interface* (JNDI) and many others.

- A servlet can call other components in the server, such as EJBs, CORBA objects, Java or PL/SQL stored procedures, and other servlets.

Servlets are an effective replacement for CGI scripts. They provide a way to generate dynamic content that is both easier to write and runs faster. In addition, servlets address the problem of doing server-side programming with platform-specific APIs: they are developed with the Java Servlet API, a standard Java extension.

So, use servlets to handle HTTP client requests. For example, have servlets process data POSTed over HTTPS using an HTML form, including purchase order or credit card data. A servlet such as this could be part of an order-entry and processing system, working with product and inventory databases, and perhaps an on-line payment system.

JavaServer Pages

JavaServer Pages (JSPs) are also becoming widely used in Web applications to provide dynamic content for HTTP clients. JavaServer Pages is a technology that is specified by Sun Microsystems as a convenient way of generating dynamic content in HTML pages that are served up by a Web application.

JSPs are closely coupled with Java servlet technology. They allow you to include Java code snippets and calls to external Java components within the HTML code (or other markup code, such as XML) of your Web pages. JSPs work nicely as a front-end for business logic and dynamic functionality in JavaBeans and Enterprise JavaBeans (EJBs).

JSP code is distinct from other Web scripting code, such as JavaScript, in a Web page. Anything that you can include in a normal HTML page can be included in a JSP page as well. In a typical scenario for a database application, a JSP page will call a component such as a JavaBean or Enterprise JavaBean, and the bean will directly or indirectly access the database, generally through JDBC or perhaps SQLJ. The server translates a JSP page into a Java servlet before being executed (typically on demand, but sometimes in advance), and it processes HTTP requests and generates responses similarly to any other servlet.

JSPs provide a standard way to separate HTML code and Java code. Web developers who are familiar only with HTML can maintain JSPs independently of the dynamic content.

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

See the *JavaServer Pages Developer's Guide and Reference* for complete information about the use of JSPs in the Oracle Servlet Engine.

About the Oracle Servlet Engine

The Oracle Servlet Engine is a complete, full-featured servlet container. The purpose of a servlet engine is to support *Web applications*. A Web application can consist of Java servlets, JSP pages, static HTML Web pages, and other Web resources. Other resources might include CGI scripts, image files, compressed archives, and various other data files.

The OSE fully supports the servlet 2.2 specification, which is part of the J2EE specification.

Although functionally identical to servlet activation in a non-server JVM, the servlet activation model for the OSE/OJVM depends upon the properties of the OJVM, and hence requires a different development model for optimum speed and scalability.

The Java process, and hence the Java servlet, are started up only when a request comes into the OSE service. This is the way the OJVM itself works, as described in the *Oracle9i Java Tools Reference*. For this reason, "heavy weight" servlets, that do a lot of initialization work when started up, often do not confer any performance advantage in the OSE/OJVM, and they always incur a performance penalty on session start-up.

The OSE/OJVM supports stateful servlets very well, due to the nature of the OJVM session model. However, if your application makes heavy use of stateless servlets, be aware that each request to a stateless servlet requires a new instantiation of the OJVM, which is costly.

Therefore, if your application uses stateless servlets you should always connect to the OSE by way of Apache and `mod_ose`, because `mod_ose` keeps a stateless connection to the OSE open for the duration of each Apache (`httpd`) process. If the application servlets do a lot of initialization (such as caching connection pools, precomputed data, or data obtained from the database), then it is likely that other client requests will get hits on the cached data when you use Apache and `mod_ose`. If you connect directly to the OSE/OJVM to run stateless servlets, each new HTTP request starts a new server session, and any caching that the servlet does is completely wasted.

The most important thing to keep in mind about the OSE is that it runs within an Oracle9*i* server. The OSE runs under the Java Virtual Machine, in a server session. Each new request to the OSE gets a separate session, each with its own JVM.

Because the servlet engine is running as part of an Oracle server session, it offers very fast access to data. A direct in-memory connection is available, using the Oracle server-side internal JDBC driver, thereby making access to SQL data from

JDBC statements almost immediate. This fast-path access is also indirectly but conveniently available to SQLJ statements in a servlet, and to JSP statements.

The OSE can also serve static HTML pages to a browser, but that is not its primary role. For efficient serving of both dynamic content and static pages it is best to combine the OSE/OJVM in a server (running servlets for fast database access) with the Oracle's HTTP server. The Oracle HTTP Server uses a module (*mod_ose*) that is provided for access to the OSE/OJVM. Chapter 4, "An Apache Module for OSE" describes `mod_ose`.

A Web application that is implemented using servlets is by definition *distributable*. When you publish an application in the Oracle9*i* server, it is simultaneously accessible by many clients. Each client that connects to the instance in which the application runs gets its own virtual JVM. It is important to realize that these are not separate threads of execution, but are completely separate JVMs.

The following statement from the *Oracle9i Java Developer's Guide* (page 1-13) summarizes the special way that the OJVM operates:

"As a database server, Oracle9*i* efficiently schedules work for thousands of users. The Oracle9*i* Aurora JVM uses the facilities of the RDBMS server to concurrently schedule Java execution for thousands of users. Although Oracle9*i* supports Java language level threads required by the Java language specification (JLS) and Java Compatibility Kit (JCK), using threads within the scope of the database will not increase your scalability. Using the embedded scalability of the database eliminates the need for writing multi threaded Java servers. You should use the database's facilities for scheduling users by writing single-threaded Java applications. The database will take care of the scheduling between each application; thus, you achieve scalability without having to manage threads. You can still write multi threaded Java applications, but multiple Java threads will not increase your server's performance."

The Oracle Servlet Engine Namespace

When the Oracle Servlet Engine is running in an Oracle server instance, it looks for pages, as well as other objects such as servlets, in its "filesystem". The OSE filesystem is a namespace that represents objects that are stored in the database, as entries in SQL tables. This namespace, which is configured and maintained using JNDI, looks to the OSE application developer just like a UNIX file system. A UNIX or Windows NT filesystem has a root directory. For example, '/' in UNIX is the root directory of a file system, and "C:\" in Windows NT is the root of a disk drive. In JNDI, a directory is called a *context*, and there is a root context, also named '/'. An object name in JNDI is called a *reference*, because it is a name bound to a specific object.

You usually access the OSE/OJVM namespace using command tools that were written using JNDI, and that execute in the Oracle9*i* server. The principal tool is the *session shell*, which is a utility that mimics a simple version of a UNIX command-line shell. The session shell provides built-in commands that let you explore the namespace, such as the directory lister *ls* and the *cd* utility, and tools that let you modify the namespace, such as *mkdir* (create a new context), *rm* (remove a context or a reference), *chmod* (change the access permissions on a context or reference), and many other UNIX-like commands. See Chapters 2 and 3 for more information about the session shell and the other tools. You can find the complete documentation for these tools in the *Oracle9i Java Tools Reference*.

Hosting a Web Application

Hosting a Web application on an Oracle server requires the following:

- You must have an Oracle9*i* (or Oracle8*i* Release 3) server with the Oracle Java Virtual Machine (OJVM) installed.

- The Oracle server must be running in an Oracle shared server configuration. Note that the Oracle shared server was called the multi-threaded server (MTS) in Oracle8*i*.

- For complex applications that serve many static pages as well as servlets and JSPs, and for client applications that use stateless connections, Oracle recommends that you also configure one or more Oracle HTTP Servers on a middle-tier system for maximum scalability and performance.

When using the Oracle Servlet Engine, you can configure your Web site to use *virtual hosts*. Virtual hosting allows you to host more than one Web domain with a single servlet engine, running in a single Oracle server session.

You can also configure the Oracle Servlet Engine to support multiple IP addresses. If your server system has multiple network interface cards (NICs), a domain for each card's address can be configured into the OSE. For example, you might do this if you have a single system that supports entry points for both intranet services inside your company's firewall and internet services external to the firewall.

Multiple IP address and name-based virtual hosting can be combined. You can configure a system that combines one or more network interfaces that host a single Web domain and other network interfaces that support multiple virtual hostnames.

Steps in Developing a Web Application

To develop a servlet-based Web application for the OSE/OJVM, you follow these basic steps. Later parts of this guide describe these steps in detail.

1. If you have not yet done so, create a Web service to host the application. This step must be done by a system or database administrator who has Oracle SYS privileges.

2. If you have not yet done so, configure a Web domain in the OSE to host the Web application.

3. Write the Java code for the servlet(s).

4. Compile the servlet(s) using a Sun Microsystems JDK 1.2-compatible Java compiler.

5. Load the servlet(s) into the Oracle server, using the `loadjava` tool. See the *Oracle9i Java Tools Reference* for information about `loadjava`.

For the OSE/OJVM, you perform steps 1 and 2 using the *session shell* tool. The session shell, and the session shell commands that you use, are described in Chapter 2, "Oracle Servlet Engine Concepts" and in Chapter 3, "OSE Configuration and Examples". Steps 4 and 5 are performed from the operating system command prompt. Steps 1, 2, 4, and 5 are best done indirectly using a makefile in UNIX, or a batch script in Windows NT. See the makefiles and batch scripts in the demo directories of your OSE/OJVM installation for examples.

You can perform the following steps using either commands in the session shell, or by using a Web Application Archive (WAR) deployment file. For maximum portability, Oracle recommends that you write and use a WAR file. See Chapter 8, "Oracle WAR Deployment" for complete information on deploying a Web application using a WAR file.

6. Create a service context to hold the Web application.

7. Publish each of the servlets and/or JSP pages that make up the application. Chapter 2, "Oracle Servlet Engine Concepts" and in Chapter 3, "OSE Configuration and Examples" describe this step.

8. Make sure that external objects called by any servlets in the application, such as EJBs, are loaded, published, and accessible.

9. Test the application using a Web browser or other HTTP client.

2

Oracle Servlet Engine Concepts

This chapter covers the following topics:

- Getting Started
- OSE Building Blocks
- The OSE Session Model
- The OSE Namespace
- Connecting to an OSE Web Application
- Web Services
- Web Domains
- Servlet Contexts
- Accessing the Oracle Database

Getting Started

This chapter discusses the basic functionality and mode of operation of the Oracle Servlet Engine (OSE). You can find specific information about configuring the servlet engine and Web applications in Chapter 3, "OSE Configuration and Examples", in Chapter 5, "Configuring mod_ose", and in Chapter 8, "Oracle WAR Deployment".

If you need help in creating, deploying, and accessing your first servlet under the OSE/OJVM, and are not interested right now in how the OSE/OJVM operates, you can get started very quickly by having a look at the Summary on page 3-23.

> **Oracle8*i* Users Note:**   Wherever this chapter refers to Oracle9*i*, the same information also applies to the Oracle Servlet Engine of Oracle8*i* Release 3, unless specifically indicated otherwise.

OSE Building Blocks

To understand how the OSE works, it is useful to have an overall grasp of its basic building blocks and tools. These are described very briefly in this section. More detailed explanations are given in later sections of this chapter.

OSE Session Model

The OSE/OJVM runs in a virtual JVM inside an Oracle database server session. It is important to understand the OSE/OJVM session model in order to develop and tune your servlet- or JSP-based application for maximum scalability and performance.

OSE Namespace

Objects that support both Web applications and the OSE system itself are organized in a hierarchical *namespace*, similar conceptually to a UNIX file system, or the directories on a Windows NT disk drive. The OSE namespace is implemented using the *Java Naming and Directory Interface* (JNDI), and OSE/OJVM objects in the namespace are persisted in Oracle server tables.

OSE Administration

The tool used to administer the OSE/OJVM is the *session shell*. The session shell is a client-side tool that connects to an Oracle server on which the Oracle JVM has been installed, and the session shell is then used to run administrative commands on the server. The session shell behaves like a UNIX shell, with built-in commands to delete objects, bind new objects, navigate the JNDI namespace, and perform administrative tasks such as creating new Web services, Web domains, and servlet contexts.

Web Service

In order for a Web browser (or any other HTTP client) to connect to the OSE and access servlets or JSPs, a *Web service* must be established in the OJVM. The Web service uses the Oracle server HTTP presentation capability. *Endpoints* for client connection to the service are also established. (An endpoint is simply a port number—programmatically a TCP socket.)

A Web browser can connect to a service by sending a request URL that specifies a hostname and an endpoint. For example the URL

```
http://Oratest:8080/
```

specifies a connection to the host system `Oratest`, at endpoint `8080`. The network DNS servers look up the host name, and substitute the correct 32-bit IP address in its place. If this endpoint is an OSE service, this URL would cause the default servlet for the domain to be activated. If the endpoint is omitted, then the default endpoint for the HTTP presentation on that host is used.

Web Domain

An OSE Web service can support one or more Web domains. While a single Web domain per service is the most common case, the OSE can also support multiple IP-based and/or name-based virtual-hosted Web domains for a single service.

Servlet Context

A servlet context is the basic unit of organization of a Web application. Each Web domain can support one or more servlet contexts. You publish a single application's servlets, JavaServer Pages, and other objects into the servlet context.

For the OSE you can use a Web application archive (WAR) file to configure a servlet context, or you can do the configuration using session shell commands.

The OSE Session Model

When a request from an HTTP client arrives at an HTTP service endpoint, an Oracle server session is started, the JVM is activated, and then OSE/OJVM is activated to handle the request.

> **Important Note:** The most important difference between the OSE/OJVM and other servlet engines, such as Jakarta/Tomcat, is that the OSE runs in an Oracle server session.

By default, the database session is authenticated as the owner of the Web domain used to connect to the server.

HTTP clients can combine several HTTP session objects in a single database session. An HTTP session can exist for each servlet context. A servlet inside a particular context can integrate the corresponding HTTP session when calling the `getSession(true)` method. If the client activates servlets from two different servlet contexts and each servlet calls `getSession(true)`, then two different HTTP session instances exist in the database session. The first time an HTTP client connects to an instance of Oracle9*i* running OSE, a session is created inside the database. *The session is a regular database session* that runs its own virtual JVM.

A stateful session is useful in applications where a dialogue should exist between the client and the servlet. A cookie is sent to the client to maintain the session, that is to assure that the client can return to the same session on subsequent HTTP requests. If the client does not support cookies, the OSE uses URL rewriting to maintain session information.

> **Note:** Any object bound into a HTTP session object is available to any other servlet that belongs to the same servlet context and that handles a request identified as being part of the same session.

The OSE supports the complete servlet interface, as required by the Servlet 2.2 specification. Any servlet that is written to operate under the OSE should be portable to another J2EE-compliant servlet container, and servlets written for other platforms will run in the OSE with no modification.

Servlet Activation

Although functionally the same as servlet activation in a non-server JVM, the servlet activation model for the OSE/OJVM depends upon the properties of the OJVM, and hence requires a different development model for optimum speed and scalability.

The Java process, and hence the Java servlet, are started up only when a request comes into the OSE service. This is the way the OJVM itself works, as described in the *Oracle9i Java Developer's Guide*. For this reason, heavy weight" servlets, that do a lot of initialization work when started up usually do not offer any performance advantage in the OSE/OJVM, and always incur a performance penalty on session start-up.

The OSE/OJVM supports both stateful and stateless servlets. However, if your application makes heavy use of stateless servlets, you need to be aware that each request to a stateless servlet could require a new instantiation of the OJVM, which would costly.

So if your application uses stateless servlets you should always connect to the OSE by way of Apache and `mod_ose`, as `mod_ose` keeps a stateless connection to the OSE open for the duration of each Apache (`httpd`) process. If the application servlets do a lot of initialization (such as caching connection pools, precomputed data, or data obtained from the database), it is likely that other client requests will get hits on the cached data when Apache/`mod_ose` are used. If you connect directly to the OSE/OJVM to run stateless servlets, each new HTTP request starts a new server session, and any caching that the servlet does is completely wasted.

Multithreading

Oracle does not recommend that you write or deploy servlets that use the Java multi threaded model in the OSE/OJVM. The OSE/OJVM works using the Oracle shared server configuration (called the Multi-Threaded Server, or MTS, configuration in Oracle8*i*). The shared server configuration is in itself a distributed system, and scales by adding new servers as the incoming requests multiply. So in effect the threading is being done at the server level, and each servlet executes in its own virtual JVM.

So, each thread of a multithreaded servlet cannot cache static information that is available to other threads. One JVM, one thread of control.

The following statement from the *Oracle9i Java Developer's Guide* (page 1-13) summarizes the special way that the OJVM operates:

"As a database server, Oracle9*i* efficiently schedules work for thousands of users. The Oracle9*i* Aurora JVM uses the facilities of the RDBMS server to concurrently schedule Java execution for thousands of users. Although Oracle9*i* supports Java language level threads required by the Java language specification (JLS) and Java Compatibility Kit (JCK), using threads within the scope of the database will not increase your scalability. Using the embedded scalability of the database eliminates the need for writing multi threaded Java servers. You should use the database's facilities for scheduling users by writing single-threaded Java applications. The database will take care of the scheduling between each application; thus, you achieve scalability without having to manage threads. You can still write multi threaded Java applications, but multiple Java threads will not increase your server's performance."

The OSE Namespace

When a Web application is deployed in an Oracle server, the Java classes that implement the servlets and JSPs are stored in the database as SQL objects. You use the `loadjava` command to load a Java class to an Oracle9*i* database—it is stored in a SQL table in the database. The OJVM is able to resolve and execute the servlet within the OJVM context, in a way analogous to the activation of a Java Stored Procedure or an Enterprise JavaBean.

The OSE/OJVM uses JNDI to implement the namespace. The JNDI layer insulates the OSE from direct dependence on the database, but enables the database to be used for persistence of the container objects. The JNDI interface leverages the features of the Oracle9*i* server, including concurrency and transactional capability, as well as scalability, replication, and other advanced capabilities.

A Short Introduction to JNDI

For those unfamiliar with JNDI, this section provides a very brief overview. For more information, point your Web browser to

`http://java.sun.com/products/jndi/index.html`

The JNDI is a set of APIs developed by Sun Microsystems as a standardized way of naming objects, and navigating to them. The OSE uses JNDI as the method of implementation for its namespace.

A namespace is simply a collection of names in a naming system. For example, UNIX provides a set of names for files stored on a file system that is organized in as a hierarchical tree structure. The OSE has a namespace that consists of objects, such as servlets, configuration objects, JSPs, as well as other material, such as graphic or sound images, that make up Web applications.

The JNDI has two fundamental concepts: *contexts* and *references*, and one fundamental action: *binding*. There are of course other actions, implemented by classes in the API, that allow the developer using JNDI to navigate the namespace, create new contexts, and do other operations.

When you develop a Web application for the OSE/OJVM, you use JNDI-based tools, implemented as session shell commands, to perform most administrative operations. A servlet developed for the OSE/OJVM can use the JNDI API to perform application tasks, such as calling EJBs or other J2EE components.

Since the JNDI namespace for the OSE/OJVM is persisted in SQL tables, you could use the JNDI API to store information in the OSE JNDI namespace that needs to be shared among Oracle sessions.

JNDI References and Contexts

A *reference* is a name-to-object binding that points to (or names) an object that implements some functionality, for example a servlet class. A *context* is an object that can contain references or other contexts. That is, a context is a set of name-to-object bindings. A context is like a directory in a file system. For example, in the UNIX naming system a file path such as `/usr/bin/wc` names three directories: '`/`' (root), '`usr`', and '`bin`', and a leaf node '`wc`', which is an ordinary file. In a JNDI namespace, these would be represented by three contexts and a reference.

Other naming systems use different notational conventions. For example, the DNS naming system uses dots to separate contexts: `java.sun.com`. The JNDI can be used to represent, or implement, any hierarchical naming system.

References and contexts are Java objects, and so they can have attributes, or properties, associated with them. For example, a file in a file system typically has a creation date, a last-modified date, and a set of attributes that tell who can read, who can modify, and, if possible, who can execute the file. In an analogous way, references and contexts in JNDI can have attributes. The OSE uses these attributes to encode permissions (read, write, execute), and to store configuration parameters for services, domains, and servlet contexts.

In this guide the JNDI-specific term "context" is frequently used. However, since a context is quite analogous to a directory in a file system, the more imprecise term "directory" is sometimes used. For example, it seems clearer to talk about a "servlet context directory" in the JNDI namespace, rather than the redundant "servlet context context", when mention must be made of the specific object that contains a servlet context.

Binding

The JNDI provides API functionality that can bind objects to named contexts or references, and unbind the name from the context or reference.

In the OSE, these names make up the namespace. Binding is a JNDI operation that associates a name with the information required to construct an object. When you give a name to a file in a file system, you are in effect binding the name to the object: the file.

As you will see in following sections, the OSE/OJVM session shell provides tools that use the JNDI bind to associate names in the OSE namespace with servlet classes loaded into the database.

Interfaces

There are two components that make up JNDI. The application programming interface, or API, implements the classes that allow the user (for example, a tool implementer) to access and navigate the namespace. The interfaces and classes that make up this API are provided by Sun.

A servlet container developer who wants to make JNDI available for use in an application must also implement the Service Provider Interface (SPI), to allow the API to access a specific set of objects that are stored somewhere. The objects might be stored in a file system, or in a database table, or perhaps even in a flat file on a disk. It does not matter how or where the objects are ultimately stored, as long as they can be accessed through the SPI.

Note that the storage does not need to be hierarchically organized. For example, a (non-object) Oracle table contains rows, but does not offer a tree-like structure. Nonetheless, it is quite feasible to interface a tree-like hierarchical namespace to a database table by writing the appropriate JNDI SPI. Figure 2–1 shows the relationship of the various components that participate in a JNDI-based application and its container.

Figure 2–1   Application Architecture Using JNDI
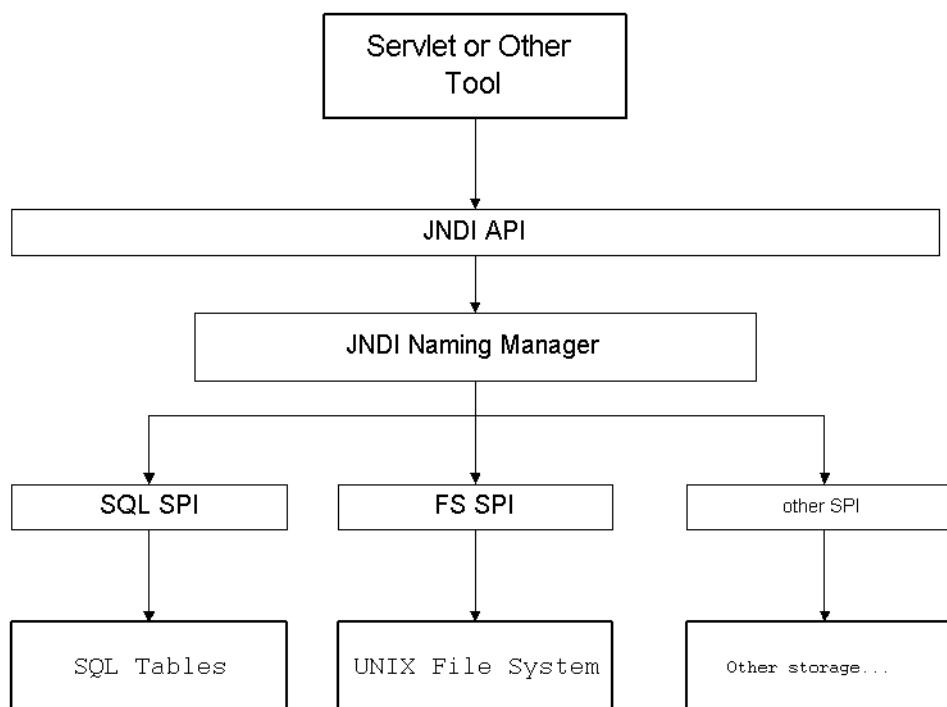
Federated Namespaces

JNDI also provides for a namespace whose root references one storage system (that is, uses one SPI) to contain one or more contexts that refer to different storage systems, that use one or more different SPIs. The second and other namespaces are federated with the root namespace in this case. The transition from one namespace to a federated one is usually transparent to the application user.

However, the OSE developer or administrator needs to understand the concept of federated namespaces, as they are used in most Web applications. For example, the document root (`doc_root` context) in an OSE/OJVM application is typically an object that is a link to a federated OS file system.

> **Note:** Files in the OS filesystem have access permissions that are associated with the operating system users, for example UNIX or Windows NT logins. But when the OSE accesses files in the OS file system, you must assign the files Java access permissions through SQL commands. The permissions are assigned to database users, not to OS logins.
>
> You use the grant_permission procedure in the dbms_java package to grant (or restrict) Java permissions (java.io.FilePermission) on files in the operating system.
>
> See the Security chapter in the *Oracle9i Java Developer's Guide* for more information.

The OJVM Root Namespace

Figure 2–2 shows an example of the contexts that might be found in the root of the JNDI namespace in an OJVM.

Figure 2–2   The Root of the OJVM JNDI Namespace

```
/ ─┬─
   ├── /bin
   ├── /etc
   ├── /system/admin
   ├── /service
   ├── /test
   ├── /HRRoot
   │
   └──
```
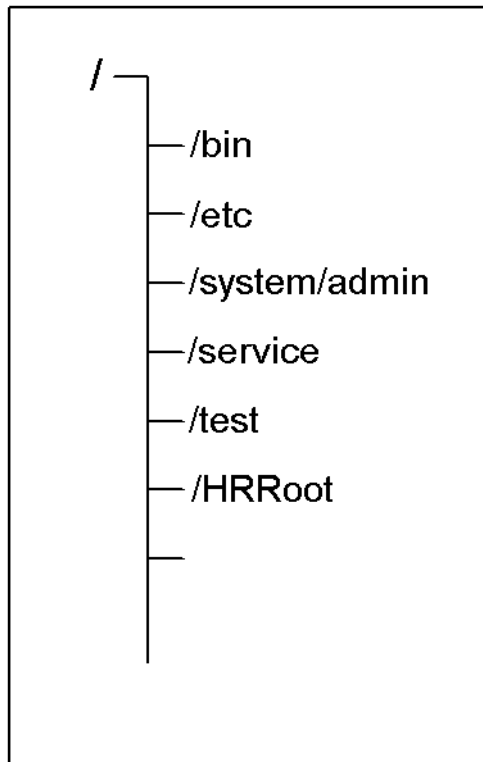
The following table describes briefly the function of each of these contexts.

Context	Purpose
/bin	Contains objects that implement system functions. For example, the session shell commands such as `createwebservice`, `createwebdomain`, `addendpoint`, `createservice` and others are found here. Do not delete these objects (Oracle SYS privileges would be required), or the OSE might not continue to function properly.
/etc	Contains published objects that are usually called from EJB or CORBA clients. They are not used by the servlet engine.
/system/admin	Holds an admin service, that supports certain servlets used by the OSE, for example the servlet that implements the session shell on the server side.
/service	Contains one object for each HTTP service that has been created for this Oracle instance. In the default installation, there will be two services: `admin` (for administrative uses), and `HRRoot` (a demo service).
/test	If present, used only for Oracle internal testing purposes.
/HRRoot	A demonstration service. Contains a single Web domain (also called `HRRoot`), with some published demo servlets.

When you create a new Web service, you normally place the service root in the root directory of the JNDI namespace.

Connecting to an OSE Web Application

Before developing and configuring a Web application that runs under OSE/OJVM, you must have available an HTTP service, with a specified service endpoint (TCP/IP socket). For example, a URL such as

```
http://Oratest:8088/myPage.html
```

specifies a connection to a TCP/IP port (or socket), number 8088. If the URL does not specify a port, the default port for the HTTP presentation, usually port 80, is assumed.

The endpoint is used by the dispatcher to connect to the HTTP service. See "Creating a Web Service" on page 2-21 for information about services and setting up a service endpoint.

There are two routes that an incoming HTTP request from an HTTP client, such as a Web browser, can take to connect to the OSE, and invoke a servlet or JSP:

1. connecting to OSE using the Oracle HTTP Server's listener with mod_ose

2. connecting directly from the browser to OSE, through the Oracle server's listener

Oracle strongly recommends that you connect to servlets and JSPs by going through the Oracle HTTP Server (Apache), and using the mod_ose Apache module to connect to an Oracle listener and then the OSE. See "Connection Using the Oracle HTTP Server as Listener" on page 2-16 for the benefits of this approach. For information about Apache and mod_ose see Chapter 4, "An Apache Module for OSE".

However, for occasional debugging purposes, it is possible connect to the OSE directly, by registering a service endpoint with the Oracle listener (or even directly with a dispatcher), and connecting to that port.

Figure 2–3 shows some of the various ways that an HTTP client can connect to an Oracle9*i* server.

Figure 2–3   Oracle 9i Connection Architecture
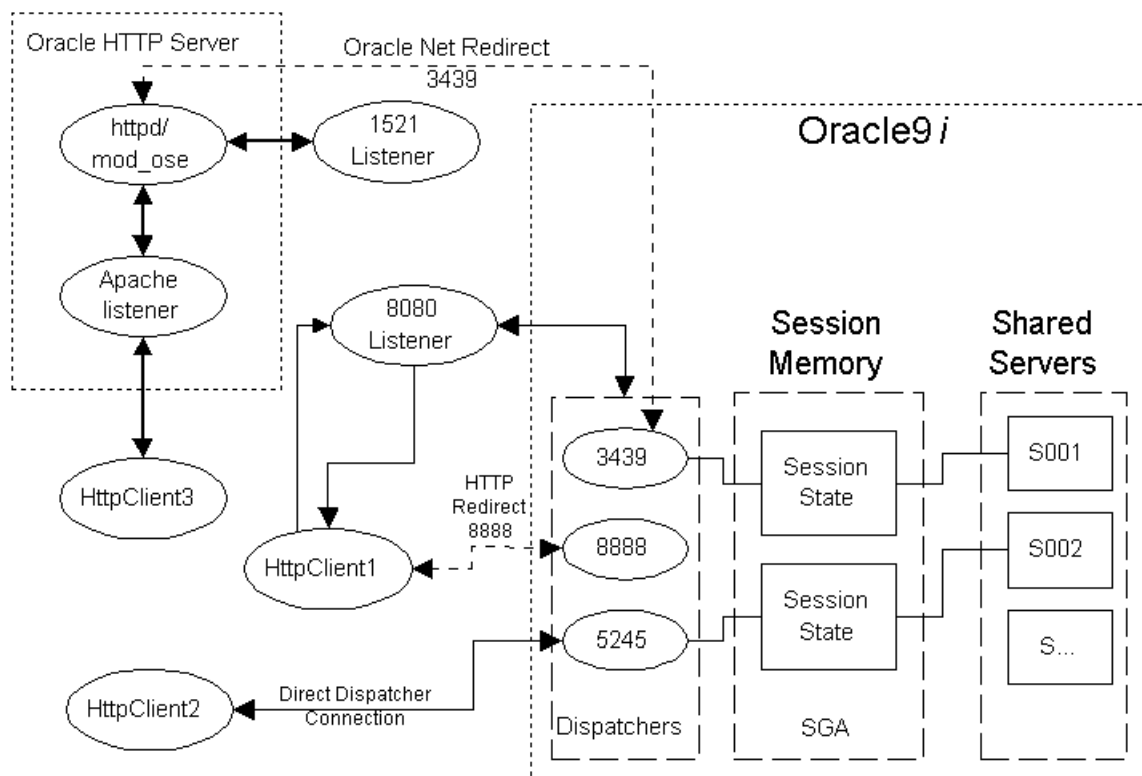
Connection Using the Oracle HTTP Server as Listener

A large Web application might have many static pages, in combination with servlets and JSP pages to serve the dynamic content. This kind of application will run more efficiently if the static pages are served directly from a file system, using a server running either on the middle tier or the data tier, and the dynamic pages are served by the OSE running in the Oracle server.

In this case, configure the application so that connections to servlets and JSP pages are routed to the OSE running in the database using the Apache module mod_ose. See Chapter 4, "An Apache Module for OSE" for information about mod_ose, and instructions on how to configure it.

Another reason to use Apache and `mod_ose` is because applications that make heavy use of stateless requests can place a heavy load on an Oracle server, if they connect directly to an Oracle listener. Each stateless request that comes *directly* to the Oracle server's OSE causes a new server session to be started, with the attendant overhead. But when using Apache with `mod_ose`, stateless requests are in effect pooled, each Apache with `mod_ose` process keeping a stateless connection to the server and OSE open. This allows much better throughput and scalability.

Direct Connection To an Oracle Listener

With a direct connection, the client request is first routed to the Oracle listener running on the host named in the URL. For example, a URL from a browser starts with the string `http://Oratest:8088/....` The domain name servers that the client system is using will look up `Oratest`, and substitute the appropriate 32-bit Internet address, for example 10.10.10.5. The request is then routed to the listener on that host system. If no listener is running on that host, an error is returned. The error is returned to the client is an HTTP 500 error. Different clients, such as different Web browsers, might handle the error in different ways, for example by displaying a "failed to connect" message.

An existing listener must then route to a service registered on the endpoint at port 8088. If no service is registered on that port, the listener returns to the client an error indicating a failure to connect. But if a service is registered, the service is started. For OSE, the Oracle JVM is started, and then the OSE is started to route the request to the appropriate servlet named in the remaining part of the URL (the URI). Routing to a servlet is explained in detail in "Finding the Servlet" on page 2-35.

Direct Connection to an Oracle Dispatcher

If you want HTTP client requests to go to an Oracle dispatcher directly, bypassing the listener, you direct your client to the dispatcher's port number. Do one of the following to discover the dispatcher's port number:

- Configure a port number for the dispatcher by adding the ADDRESS parameter that includes a port number.

- Discover the port assigned to the dispatcher by invoking `lsnrctl service`.

- If you choose to configure the port number, the following shows the syntax:

```
mts_dispatchers="(address=(protocol=tcp | tcps)
(host=< server_host>)(port=<port>))
(presentation=oracle.aurora.server.<http_service_name>)"
```

The attributes are described below:

ADDRESS (ADD or ADDR) Specifies the network address that the dispatchers will listen on. The network address may include either the TCP/IP (TCP) or the TCP/IP with SSL (TCPS) protocol, the host name of the server, and an HTTP listening port, which may be any port you choose that is not already in use.

The client supplies the port number on its URL, as follows:

```
http://<hostname>:<portnumber>
```

Notice that the URL excludes a SID or service name. The dispatcher does not need the SID instance or service name because it is a directed request.

PRESENTATION (PRE or PRES) The PRESENTATION enables support for HTTP. For an HTTP presentation, use the HTTP service name that you established with the `createwebservice` command, for example

```
http://testService
```

Notice that the URL excludes a SID or service name. The dispatcher does not need the SID instance or service name because it is a directed request.

> **Note:**   While it is possible to connect directly to an Oracle dispatcher from an HTTP client, and this capability might be useful in some circumstances for debugging, Oracle does not recommend that you do this when testing and deploying production applications. You lose the load balancing and other advantages of the listener.

Web Services

An Oracle9*i* server can present many different services to a client. The most common one are:

TTC services Clients that follow the traditional client-server model, perhaps with a client running an OCI or a Forms application, connect to the TTC Oracle service. (TTC is a traditional Oracle abbreviation that stands for "two-task common".)

IIOP services Remote clients that call Enterprise JavaBeans or CORBA objects in the server connect to a service that supports session IIOP connections.

HTTP services Web-based client applications connect to an HTTP service on the Oracle server.

Some developers are able to write customized special services, that can support an arbitrary communications protocol, and provide database services beyond the scope of those offered by standard Oracle9*i* server. Writing custom services is not covered in this document.

> **Note:** The session IIOP and HTTP services are required for many applications that use the OJVM. The OJVM requires the Oracle shared server. (The shared server configuration was called the multi-threaded server (MTS) in Oracle8*i*). IIOP and HTTP services are not normally supported for a dedicated, non-shared, Oracle server.

Oracle9*i* provides installed support for HTTP services. A single system can support one or more Web services, with different endpoints. For example, an e-business that sells a range of products might want to install several Web services, each supporting a different product line.

A Web service is associated with an *endpoint*. An endpoint is a port to which the HTTP client can send incoming requests. When you register a Web service you must specify the endpoint for the listener. For the TCP/IP connections that all HTTP requests use, the endpoint is a socket number. 80 is the usual default socket number for HTTP requests. Secure Socket Layer (SSL) connections would use a different port, perhaps 90.

Only a user with Oracle SYS privileges can establish a Web service and a service endpoint for an Oracle server. Severe security problems could arise if non-privileged users were able to establish new Web services.

Single-Domain and Multi-Domain Services

The Oracle Servlet Engine supports two kinds of Web service: *single-domain* and *multi-domain*. A single domain Web service supports a single Web domain. In a single-domain Web service, the port (endpoint) is the sole determiner of the routing of servlet request to service and then the domain.

There are several kinds of multi-domain services. Multi-domain services are also called *virtual hosting*.

Virtual Hosting

For many purposes, a single-domain service is all that is required. However, in many cases a single server machine must host multiple Web domains. For example, a single system that is required to host an administrative domain and an application domain, each with entirely different sets of privileges and applications. Or, a single server system might be required to host users on the intranet as well as users connecting from the internet, outside the firewall. In the latter case, the server system will typically have two or more physical network connections, and each network interface card can support a separate Web domain.

Support of multiple Web domains is called *virtual hosting*. There are two types of virtual-hosted service:

1. hosting multiple domains using different IP addresses, called *IP-based* virtual hosting

2. multiple domain names, called *name-based* virtual hosting

Because it is possible for a Web domain that is served by an IP address to have multiple domain names (multiple entries in the DNS tables), it is possible to combine case 1 above with case 2. For example, a server system could have three network interface connections, two of which support single Web domains, and a third that supports multiple (name-based virtually-hosted) domains.

The OSE/OJVM supports virtual hosting differently from a purely JDK-based implementation of servlet containers, such as Tomcat or JServ. See "Virtual-Hosted Services" on page 2-24 for more information.

You can also find more information on setting up single and multiple Web domains in Chapter 3, "OSE Configuration and Examples".

Creating a Web Service

The system administrator or DBA uses the `createservice` or the `createwebservice` session shell commands to establish a service for the OSE, and identify its

- name
- root in the OSE JNDI namespace

The administrator must be connected to the session shell as SYS to use either of these service creation commands.

The `createwebservice` command is a more specific form of `createservice`, and can only be used to create HTTP-based services. For information about the more general `createservice` command, see the *Oracle9i Java Tools Reference*.

Once a service is created, the administrator must establish service endpoints on which the service can listen, using the `addendpoint` session shell command.

See "Creating a Web Service" on page 3-5 for specific instructions on how to create a new Web service, and add listening endpoints for it.

The Service Context

The OSE JNDI namespace contains a `service` "directory" at its root. Objects of type `SERVICE` exist in `/service`, one for each Web service that has been created for the Oracle instance. For example, a newly installed Oracle9*i* server with the JVM has a `/service` context that includes the following services:

- `admin`
- `HRService`

The `admin` service is required for proper operation of the OSE, and must not be removed. The `HRService` is installed as part of the OSE demo set.

Each `SERVICE` object in the `/service` context contains a set of property groups that describe the service. These include

- the `service` group:
  - the service name: `service.name`
  - the overall default timeout for the service: `service.globalTimeout` (in seconds)
  - the `service.root`

- the `endpoint` group:

  - the endpoint class: `endpoint.class`

  - the endpoint name: `endpoint.name`

  - specifics for the named endpoint, such as ports, minimum and maximum number of threads that the endpoint can handle, endpoint timeout value (in milliseconds)

- the `environment` group:

- the `contexts` group:

- the `mime` group:

  - properties for all MIME types that the service handles

Most of the property values in the `SERVICE` object are set by the `createservice` or `createwebservice` command, and by the `addendpoint` command, either as defaults or from the parameters of the commands.

You can use the session shell `getproperties` command to list the properties for a `SERVICE` object.

Web Domains

An HTTP Web service can support one or more than one Web domain. The Web domain is the basic organization unit of the Web service, and contains the servlet contexts into which applications are loaded. Web domains correspond either to service roots, to DNS domain names, or to IP addresses of that host, depending upon how the Web service is configured (single- or multi-domain).

Each Web domain that a service supports has a name: the *domain name*. The domain name is the same as the name of the Web service root in the most common case—a single-domain Web service.

In the OSE/OJVM, a Web domain is owned by a database schema. For example, user HR might own the /HRRoot Web domain, where /HRRoot is also the root of a single domain service.

JNDI Contents of a Web Domain

Every Web domain context contains a config object in the JNDI namespace. The domain also contains a contexts directory and a logs directory. A single-domain service Web root also has a servicelogs directory, but that is for the Web service, and is created by the createwebservice command.

Figure 2–4 shows the JNDI namespace after the creation (as shown above) of a single-domain Web service and the corresponding Web domain.

Figure 2–4   JNDI Namespace—Single-Domain Service

```
/
    /bin  <----------------------------- OSE system servlets (session shell commands)

    /etc

    /system

    /service <---------------------- where service configuration is kept

        admin

        testService  <----------- the new service configuration that was created

        HRService  <----------- the demo service configuration

    /HRRoot   <-----------        where the demo servlets live

    /testRoot  <-------------------- the service and domain root

        /servicelogs <----------- the service logging utilities

        /logs  <--------------------- the domain logging utilities

        config <-------------------- the domain configuration object

        /contexts  <--------------- where the domain servlet contexts are kept
```

The remainder of the Web Domains section of this Guide explains multi-domain
Web services. If your application uses only a single-domain service, you can skip
ahead to "Servlet Contexts" on page 2-32.

Virtual-Hosted Services

A multi-domain (also called virtual-hosted) Web service can support more than a
single Web domain under a single service. There are three kinds of virtual-hosted
service:

1. A service that uses separate host names (for example `abc.com` and `abc.us.com`) for different Web domains. These are *name-based virtual-hosted* services.

2. A service that uses different network interfaces to serve different Web domains. In this case, the system supports different IP addresses, one for each discrete network interface (network interface card, in the system). These are called *IP-based virtual-hosted* services.

3. A combination of 1 and 2. That is, a service that supports several IP addresses, where each IP address can support name-based virtual hosting. In this case, each IP-addressed domain does not have to support virtual hosting. Some might do so, and some might support just a single Web domain.

A Web server/servlet engine combination such as Apache/Tomcat supports multi-domain Web services in two ways. There can be a separate servlet engine process for each Web domain, each perhaps configured differently. Or, for some cases, a single process can support virtual hosting. (This can run into OS limitations if there are a large number of virtual hosts.)

With the OSE/OJVM, however, there is no reason or need to have multiple Oracle instances running to serve different Web domains. The Oracle JVM is capable of supporting multiple Web domains, and has proven to scale well when doing so.

So the OSE/OJVM allows you to configure an Oracle servlet engine to support multiple Web domains and/or multiple IP addresses, all running in the same server instance.

Name-Based Virtual Hosting

Name-based virtual hosting is used to handle the case in which multiple DNS domain names resolve to the same IP address, and multiple Web domains are needed to service each domain name.

For example, you might have the following domain names

■ `xyz.com`

■ `xyz.us.com`

all resolving to the IP address 10.5.5.10.

In the OSE/OJVM you handle name-based virtual hosting by establishing a separate Web domain under a single service root. **The name of the Web domain must be the same as the DNS-recognized hostname that is used in the request URL.**

See "Determining the Web Domain" on page 2-27 for information about how the OSE processes these URLs to route requests to the correct Web domain.

IP-Based Virtual Hosting

In some situations it is a requirement that a single system support multiple IP addresses. The usual case is that the machine has multiple network interface cards (NICs), but virtual interfaces (also called "ip aliases") could also be used.

In the OSE/OJVM, you create an IP-based virtual-hosted service by establishing separate Web domains under the service root. **The names of the Web domains are exactly the same as their corresponding IP addresses, and are directly under the service root.**

How are these Web domains accessed in the URL? They are accessed by using the IP addresses as the domain names in the URL. For example, the request URL

```
http://10.5.5.10:8080/
```

could reach the Web domain at `/MHHost/10.5.5.10`.

Note the difference between the URL `http://10.5.5.10:8080/` in a single-domain case, and the URL used to access the IP-based domain `10.5.5.10` above. In the single domain case, the IP address in the URL is used to reach the server, via the DNS mappings. In the IP-based virtual host case, the IP address of the network interface is used to determine the correct Web domain.

IP- and Name-Based Virtual Hosting

When you set up a service that supports both IP-based and name-based virtual hosts, the name-based Web domains are in effect embedded in the IP-based domains. See Figure 2–8, where for example the Web domains `def.com` and `ghi.com` are contained within the IP-based domain `10.5.5.11`.

In the case of embedded Web domain, it is only necessary to use the `createwebdomain` command to create the "leaf" domains, that is the domains that are used, and that must contain a valid `config` object, a valid `/contexts` directory, and so on. So for example, to set up the Web domains for the virtual-hosted service shown in Figure 2–8, you use the session shell `mkdir` command to create the non-leaf domains, and use `createwebdomain` only for the terminal domains. Using `createwebdomain` to create the IP domains would not result in errors at run time, but it would create unnecessary objects in the JNDI namespace that are never accessed.

Here are example commands that you might use to set up the four Web domains for the service shown in Figure 2–8 on page 2-31:

```
$ cd
$ createwebservice -root /ServiceRoot -ip -virtual IPservice
$ # add endpoints and chown, not shown
$ mkdir /10.5.5.10
$ mkdir /10.5.5.11
$ createwebdomain -docroot /tmp/testService1 /10.5.5.10/abc.com
$ createwebdomain -docroot /tmp/testService2 /10.5.5.10/10.5.5.10
$ createwebdomain -docroot /tmp/testService3 /10.5.5.11/def.com
$ createwebdomain -docroot /tmp/testService4 /10.5.5.11/ghi.com
```

Determining the Web Domain

In the case of a multi-domain Web service, the OSE must determine which Web domain in the Web service the request URL should be directed to. There are four pieces of information that the OSE can use to determine the Web domain:

1. The JNDI context that is the service root for that service.

2. The host part of the URL, that is all the text after the `http://` protocol indicator and before the next forward slash ('/'). The host part of the complete URL

   `http://myHost.co.uk:8080/ose/testServlet`

   is

   `myHost.co.uk:8080`

   where `:8080` specifies the port number.

3. The IP address that the request arrived on. This information is passed to the OSE independently of the URL.

4. Whether the Web service is virtual hosted, either IP-based, name-based, or both. This information is in the properties of the object having the Web service name in the `/service` context of the JNDI namespace.

Note that if the service is not virtual hosted, it must be single domain, and the Web domain is simply the service root.

The algorithm used to determine the Web domain for a request to a multiple domain Web service is the following:

1. Determine the service root (from the Web service properties).

2. If the service is IP-based virtual hosted, use a text form of the actual IP address to lookup a context of that name. If one is found, that is the Web domain.

3. If (2) succeeds, and the service is also name-based virtual hosted, look for a context in the JNDI namespace that matches the host name in the URL.

Examples

This section demonstrates how the OSE processes a URL to find the Web domain. Four cases are shown, one for each of the four Web service types: single-domain, name-based virtual-hosted, IP-based virtual-hosted, and both IP-based and name-based virtual-hosted.

In each case, two URLs are processed:

- http://abc.com/
- http://10.5.5.10/

The IP address passed to the OSE for each of these examples is always 10.5.5.10.

Figure 2–5 shows request processing for a single-domain Web service. The IP address available to the OSE outside of the URL is ignored, because it is a single-domain service, but either form of the host name in the URL—using the DNS host name or using the actual IP address, direct the requests correctly to the single Web domain.

Figure 2–5   Single-Domain Request Processing
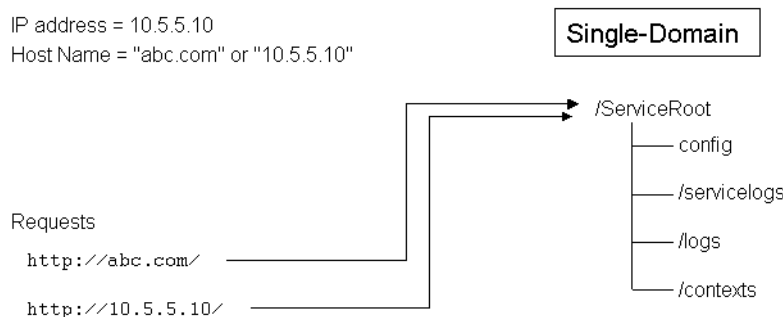
Figure 2–6 shows an example of a name-based virtual hosted service, with two Web domains. The Web domains are named abc.com and 10.5.5.10, as shown in the JNDI namespace. The interesting thing about this example is that the domain named 10.5.5.10 is reached *by name*, and not by the entirely incidental fact that the service IP address also happens to be 10.5.5.10. This is because the service type does not include IP-based virtual hosting, so the OSE ignores the actual NIC IP address.

Figure 2–6   Name-Based Virtual-Hosting Request Processing
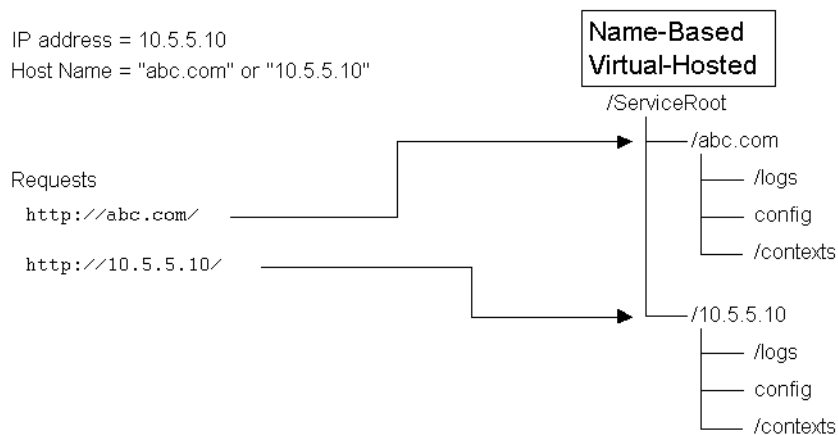
Figure 2–7 shows an example of an IP-based virtual-hosted service. There are two separate Web domains, named 10.5.5.10 and 10.5.5.11. Since the requests are incoming on the IP address 10.5.5.10, both requests are serviced by the Web domain named 10.5.5.10.

Figure 2–7   IP-Based Virtual-Hosted Request Processing
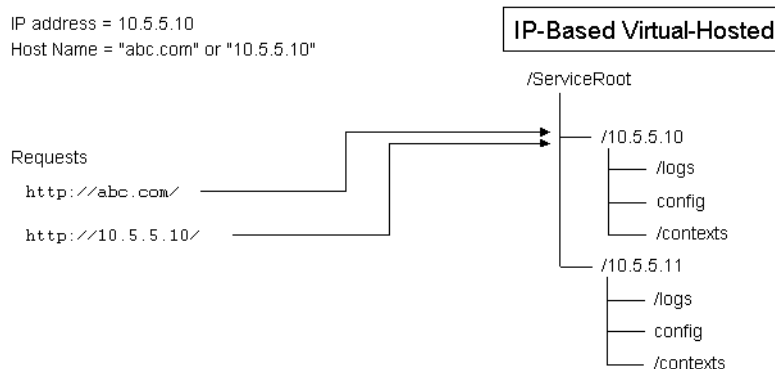
Figure 2–8 shows the request processing when the service is configured for both IP-based and name-based virtual hosting. This example is a somewhat contrived case, as there is a name-based virtual-hosted Web domain called 10.5.5.10 that is in fact nested under the IP-based Web domain also called 10.5.5.10. Although it would be unusual to set up a Web service with this configuration, it is perfectly legal to do so.

Figure 2–8   Both IP-Based and Name-Based Virtual-Hosted
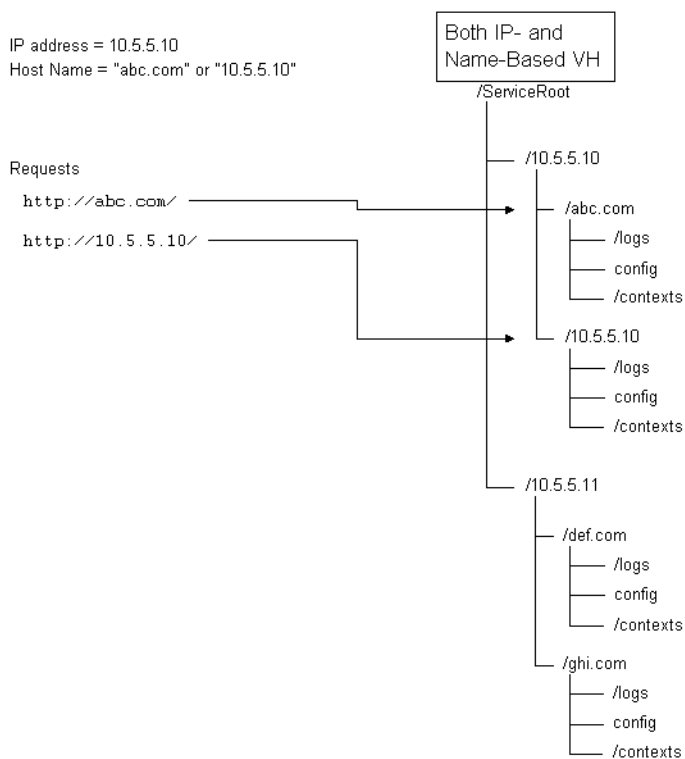
Servlet Contexts

This section describes the OSE servlet context. It includes the information you need to understand how a servlet context governs a Web application. For specific information about creating and configuring a working servlet context, and for examples of session shell commands to do this, see "Creating Servlet Contexts" on page 3-13.

Overview

A *servlet context* holds a Web application that is loaded into the OSE. The servlet context contains the servlets that make up the application, as well as other objects that the application needs. These might include JSPs, image files, sound files, static HTML pages, and other objects that make up the application.

You should create at least one servlet context for each discrete application that the Web domain supports.

A servlet context belongs to a specific Web domain. When you create a servlet context, you must specify the Web domain for the servlet context. A servlet context is a self-contained namespace in the JNDI namespace. All servlets in the context have a published name, and the name is published in a single context (directory) in the JNDI namespace for the servlet context. So no two servlets in a servlet context can have the same name.

> **Note:** A servlet context is owned by an Oracle server schema. A schema can "own" more than one servlet context, but a schema has a single namespace for class names. A servlet class can be published in different servlet contexts, with the same or different published names.

The servlet context maintains certain static information about the context. Each servlet context has a configuration object, named `config`, that is located in the root of the servlet context. The config object has property groups that provide information about things such as what languages and character sets servlets will accept, what MIME groups are supported, and the servlets that the application contains. See "Configuring a Servlet Context" on page 3-15 for specific information about the `config` object.

When you create and maintain a servlet context, you must keep two concepts distinct: the servlet *class* and the servlet *published name*. The class is

- the `.class` file that results when you compile the Java servlet source on the client system

- this object when loaded into a schema in the Oracle server database (done when you execute the operating system level `loadjava` command)

- this object to which a reference is published in the JNDI namespace, under a published name, when you execute the session shell `publishservlet` command

The published name is the reference to the class that is stored in the `named_servlets` directory of the servlet context when you publish the servlet, using the `publishservlet` session shell command, or when you deploy an application using a WAR file.

Each servlet context has the following:

- a name

- an owner

- a Web domain that it belongs to

When a Web domain is first created, only one servlet context is created with it: the *default context*. Applications should not run under the domain default context. The default context is there to handle cases where a URL specifies a domain, but no path to a servlet context or a specific servlet in the context. In that case, the default context can serve a default page.

JNDI Namespace Contexts and Objects

A newly-created servlet context has the namespace structure shown in Figure 2–9.

Figure 2–9   JNDI Servlet Context Namespace
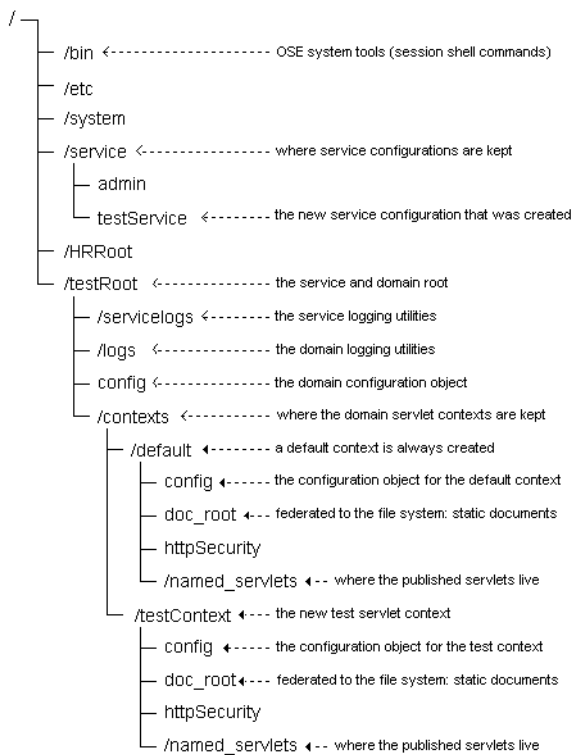
Loading and Publishing Servlets

To make your servlets available to HTTP clients and to other servlets you must first load them into the Oracle server, and then publish them to the JNDI namespace. All lookups of servlet references (such as in URLs) are done through the JNDI namespace.

You use the `loadjava` operating system command to load the servlet to the Oracle server. Use the `publishservlet` session shell command to publish a servlet to the JNDI. See "Publishing Servlets" on page 3-21 for specific instructions about servlet publication, and examples.

Finding the Servlet

When configuring the servlet context, you should understand how the OSE processes a URL sent in by an HTTP client, to find the right servlet context and the servlet in the context. Within a Web domain, a servlet is located by

- the virtual path of the servlet context, which can be null

- the virtual path of the servlet, which must be specified in the URL

The published name of the servlet may or may not be the same as virtual path name. This allows considerable configuration flexibility, as the same servlet can be reached by different paths.

The URL

The Uniform Resource Locator is part of the request that the client sends to the OSE. A URL consists of three or four main parts:

- the protocol

- the host specification (along with an optional port number)

- the URI

- additional optional information passed along by the client, such as one or more query strings

Figure 2–10 shows graphically the format of a typical URL request.

Figure 2–10   URL Format


```
http://host_name:port_no/URI

http://Oratest:8088/ose/simpleServlet?x=12345
```

PROTOCOL
HOST NAME
PORT NUMBER
CONTEXT VirtualPath
SERVLET VirtualPath
Query String
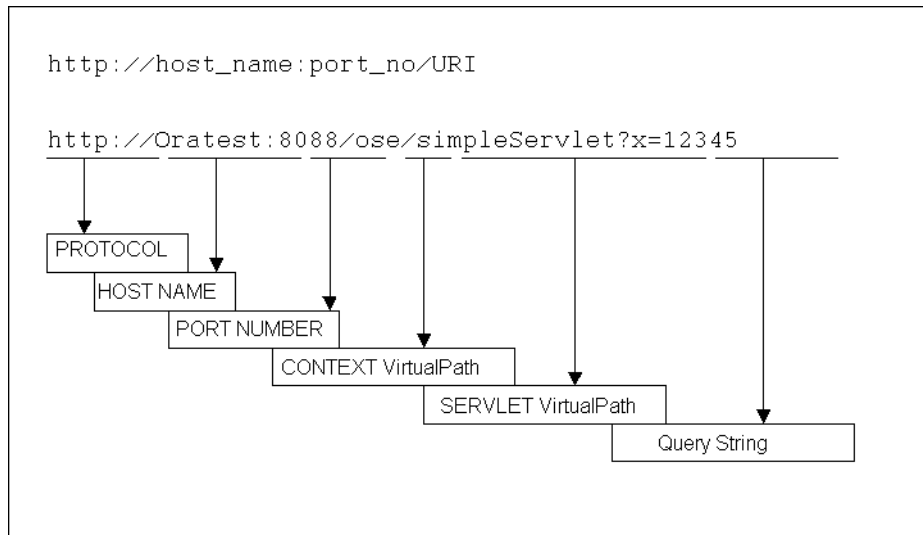
The protocol is the initial part, delimited by a colon and two forward slashes. The protocol for all requests accepted by the OSE must be `http://`. (Other protocols, accepted by different services than those the OSE supports, might be `ftp://`, or `sess_iiop://`.)

The host name and port number determine the Web service and the Web domain to which the request is to be forwarded. Every thing after the port number, or after the hostname if the port number is missing (and thus is defaulted), is the URI. The URI contains

- the servlet context specification, that is the servlet context virtual path, which may be null

- the servlet virtual path

- extra information from the client, that does not map to the servlet context virtual path or the servlet virtual path, and which is passed on to the servlet for further processing as required

These are shown in the lower parts of Figure 2–10. In this example, the remaining material in the URI consists of a query string.

The method that the OSE uses to determine the correct Web domain was described in the section "Web Domains" on page 2-23.

> **Note:** The components of the URI can be retrieved in servlet code using the `HttpServletRequest` methods:
>
> - `getContextPath()`
>
> - `getServletPath()`
>
> - `getPathInfo()`
>
> The complete URI of the request can be obtained using the `getURI()` method of the `HttpServletRequest` interface.

The Servlet Context Virtual Path

The `contexts` property group of the Web domain `config` object contains the virtual path mappings for each servlet context in the domain. For example, if `HRContext` is the name of a servlet context in the `HRRoot` domain, then the `config` object for the `HRRoot` domain might contain an entry such as:

```
--group--=contexts
/ose=HRContext
```

where `/ose` is the virtual path mapping for the `HRContext` servlet context. The virtual path mapping for the servlet context is set by the `-virtualpath` parameter of the `createcontext` command.

When the OSE processes a URI, it looks for the longest possible match to find the servlet context. So for example if a Web domain contains two servlet contexts, one having a virtual path mapping `/ose`, and the other a virtual path mapping `/ose/test`, and the URI is `.../ose/test/servlet1/`, then the OSE looks for a servlet with the virtual path `servlet1` in the servlet context whose virtual path mapping is `/ose/test`. In this case the OSE does not look for a servlet with the mapping `/test/servlet1` in the servlet context whose virtual path mapping is `/ose`.

The Servlet Virtual Path

The virtual path of the servlet itself is specified in the `publishservlet` session shell command. It cannot be null, as the servlet virtual path is the name that the OSE uses to find the servlet in its context.

If the OSE finds no servlet virtual path in the URI, then the default servlet for the context, or for the domain, is activated. The following example should clarify this.

Example 2–1   Servlet Configuration 1

Assume that you have set up the following

1. a Web service called `testService` on host `Oratest`, that listens on endpoint **8088**

2. the root of the service in the JNDI namespace is `/testRoot`

3. a single Web domain within that service: `testRoot`

4. two servlet contexts in the Web domain: `oseContext` and `testContext`

5. `/ose` is assigned as the virtual path for the `oseContext` servlet context

6. `/ose/test` is assigned as the virtual path for `testContext`

7. a servlet with the class name `oseServlet` is loaded into the database, and published to the `oseContext` servlet context as `Servlet1`, with the virtual path name `/test1`

8. a servlet with the class name `simpleServlet` is loaded into the Oracle database, and published to the `testContext` servlet context as `Servlet1`, assigning it the virtual path `/simple`

Figure 2–11 shows a view of the relevant parts of the OSE JNDI namespace as configured for this example.

Figure 2–11   JNDI Namespace for Servlet Configuration 1

```
/
├── /bin
├── /etc
├── /system
│
├── /service
│     ├── /admin
│     ├── /testService
│     └── /...
├── /testRoot
┆     ├── /servicelogs
┆     ├── /logs
┆     ├── config
      └── /contexts
            ├── /default
            │     ├── config
            │     ├── /doc_root
            │     └── /named_servlets
            │
            ├── /oseContext
            │     ├── config
            │     ├── /doc_root
            │     └── /named_servlets
            │           ├── servlet1
            │           ├── (servletn)
            │           └── (...)
            │
            └── /testContext
                  ├── config
                  ├── /doc_root
                  └── /named_servlets
                        ├── servlet1
                        ├── (servletm)
                        └── (...)
```
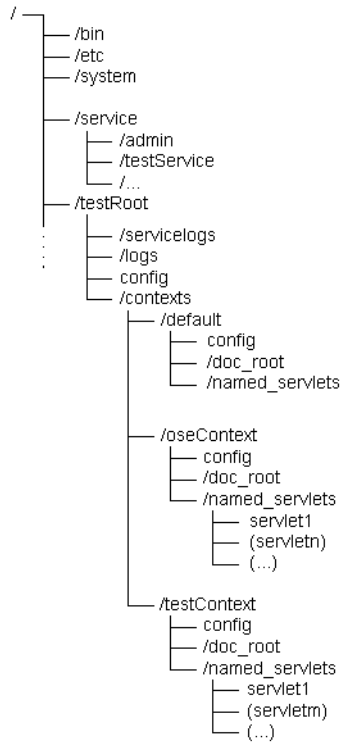
Figure 2–12   Processing a URL: Example1
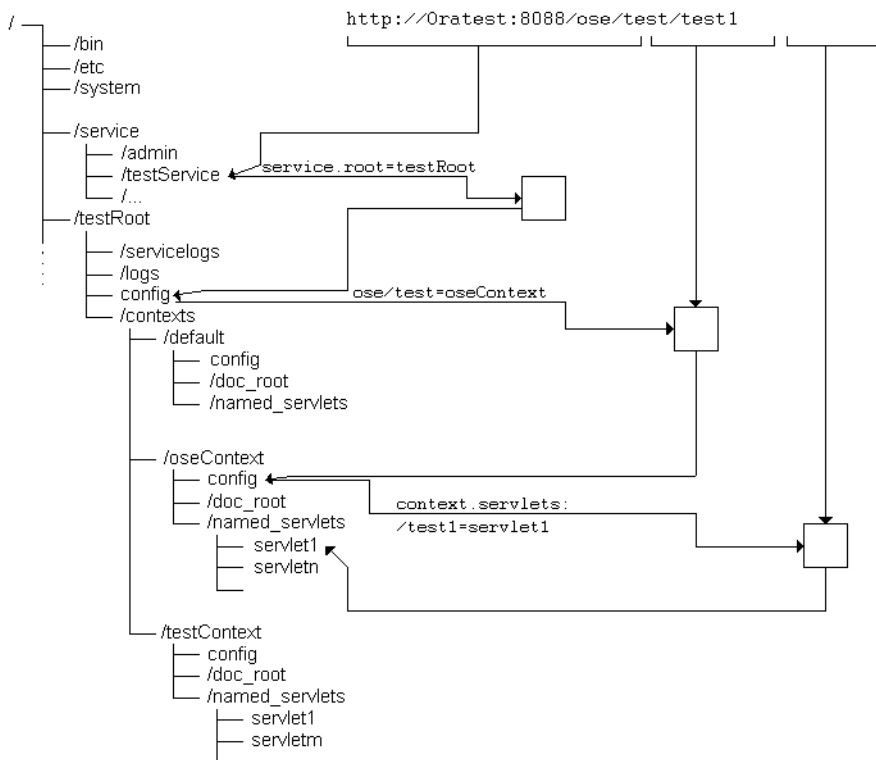
Figure 2–12 shows how the OSE resolves the URL

```
http://Oratest:8088/ose/test/test1
```

to instantiate the servlet `servlet1` in the `oseContext` servlet context, given the configuration listed in Example 2–1.

The OSE processes the URL with the following steps:

1. The service is determined by the host name (`Oratest`) and the port number (8088).

2. The service has a single Web domain: `testRoot`. This is determined by the properties in the `/service/testService` object (the configuration object for the `testService`).

3. In the `config` object for the `testRoot` domain, the OSE finds that the virtual path `/ose/test` maps to the `testContext` servlet context. This mapping is in the `contexts` property group of `config`.

4. In the `testContext` servlet context, the `config` object maps the virtual path `/test1` to the reference `servlet1` in the `named_servlets` directory.

5. The OSE activates the servlet class `simpleServlet`, and passes it the URL, parsed into its components (URI, context info, PathInfo).

6. If a servlet context match is not found, the request is served by the default servlet for the domain: `/testRoot/contexts/default`.

According to the Servlet 2.2 specification, entries in the `context.servlets` property can be paths or wild-card names. Partial paths have priority over wild-card names. Exact matches have priority over both partial paths and wild-card names.

If no match for the requested URL is found, the OSE looks for a servlet named `defaultservlet`, first in the servlet context directory, next in the Web domain directory, and then in the Web service. If the default servlet is found, it processes the request. If no default servlet is found, OSE generates an internal error code.

> **Note:** In every Web domain, a default servlet is automatically generated but is not published in the JNDI namespace. In any cases, OSE should be able to find a default servlet.

Example 2–2   Servlet Context Mappings

This example shows the effects of four different servlet context virtual path mappings. There are three servlet contexts, cleverly named `ServContext1`, `ServContext2`, and `ServContext3`. The fourth case has a null mapping. The virtual path mappings are:

1. `ServContext1: /abc`

2. `ServContext2: /abc/def`

3. `ServContext3: /abc/def/ghi.html`

Assume that the URI in the request URL is `/abc/def/ghi.html`. Here is what will happen in each case:

1. A servlet with the virtual path mapping `/def/ghi.html` is searched for in `ServContext1`. If found, it is activated.

2. A servlet with the virtual path mapping `/ghi.html` is searched for in `ServContext2`. If found, it is activated.

3. Otherwise, the default servlet for `ServContext3` is activated.

Accessing the Oracle Database

When a servlet needs to communicate with the database, it uses JDBC and an Oracle JDBC driver. There are three basic kinds of JDBC drivers available with Oracle9*i*:

- the OCI driver
- the server-side internal driver
- the thin driver

The OCI driver is used by thick clients who are connecting through Oracle Net to a server.

Server-side Internal Driver

The server-side internal driver is used by a servlet when accessing the database in the same instance that the servlet is running in. This driver provides a fast, in-memory path to the SQL data, as no network connectivity is required when using this driver.

To connect using the server-side internal driver, first make sure that your servlet code imports the `java.sql` and `oracle.jdbc` packages, as follows:

```
import java.sql.*;
import oracle.jdbc.*;
```

Use the `defaultConnection()` to connect. No connect string is required, as the connection is always the same:

```
Connection conn = null;
try {
  // connect with the server-side internal driver
  OracleDriver ora = new OracleDriver();
  conn = ora.defaultConnection();
  ...
```

For a complete examples that use the server-side internal driver to access the sample schema HR.EMPLOYEES table, see "Creating a Servlet" on page 3-29, and "EJB" on page 6-5.

Thin Driver

The thin driver is used by Java applets on the client side, and is also used by servlets and JSPs when connecting from one database instance to another. So if you

have a servlet running in one instance, and need to call an EJB, or a CORBA object, or another servlet that is running in a different instance, use the JDBC thin driver.

3

OSE Configuration and Examples

This chapter describes how to set up and configure applications to run in the Oracle Servlet Engine under the Oracle JVM. Also included are many examples that show the configuration steps, and some demo servlet applications.

Accessing servlets in the OSE/OJVM by going through the Oracle HTTP Server (powered by Apache) and the mod_ose module requires additional configuration steps. Chapter 5, "Configuring mod_ose" describes these steps.

This chapter contains the following topics:

- Connecting to the OSE
- Configuration Steps
- Creating a Web Service
- Creating Multi-Domain Web Services
- Creating Web Domains
- Creating Servlet Contexts
- Configuring a Servlet Context
- Publishing Servlets
- Summary

Connecting to the OSE

The OSE runs in a virtual JVM inside an Oracle (database) server session. You must be running an Oracle shared server configuration to run the Oracle JVM, and hence to use the OSE. There are three ways that an HTTP client can access a servlet or JSP in the OSE:

1. by connecting to an OSE Web service directly through a dispatcher

2. by connecting to an Oracle listener that is configured to hand HTTP requests off to an OSE Web service (through a dispatcher)

3. by connecting to the Oracle HTTP Server (powered by Apache), and routing HTTP client requests through mod_ose to the Oracle Servlet Engine in an Oracle (database) server

The first of these possibilities can be useful for debugging, or for quick testing of deployed servlets. But because it does not provide adequate scaling when multiple clients are involved, it should never be used when developing or testing production applications.

The second possibility can serve when your application consists solely of stateful servlets or JSPs. This, however, is true of few applications.

The third possibility is the one that Oracle recommends for all applications. Static HTML pages and other static data are served from the Oracle HTTP Server, and requests for dynamic content are routed through mod_ose to the OSE. To set up your application to run in this mode, there are additional configuration steps that you must perform. For example, you must configure the Apache mod_ose configuration file ose.conf to indicate which requests will be routed to the OSE. See Chapter 4, "An Apache Module for OSE" and Chapter 5, "Configuring mod_ose" for information about configuring mod_ose.

It is also possible to use the Oracle HTTP Server with mod_ose to connect to OSE applications with a server configuration that does not normally use shared servers. See "Non-Shared Server Installations" on page 5-12 for more information.

Configuration Steps

Configuring the Oracle Servlet Engine itself and an application that is to run in the OSE requires some or all of the following steps:

1. possible reconfiguration of the Oracle server, through the server startup file (the "INIT.ORA" file)

2. establishment of Oracle Net parameters, for example by modifying the TNSNAMES.ORA file or an LDAP configuration file

3. creating an OSE Web service that listens to one or more endpoints

4. creating an OSE Web domain to contain the application contexts

5. modifying the properties of a Web domain

6. creating one or more application (servlet) contexts

7. modifying the properties of an application context

8. loading servlets (and JSPs) into the Oracle server (the database)

9. publishing servlets so that the servlets have a virtual path associated with their location in the OSE namespace

Not all of these steps are required in all cases. For example, steps 1 through 7 will not be required to run most of the sample applications that come with this OSE release. You might need to perform steps 1 through 3 or 4 only once for all your application requirements.

Configuring the Oracle Server

The OSE/OJVM runs in a "virtual JVM" in the Oracle server. As with any other application that uses the OJVM, you must be running the Oracle shared server to use the OSE.

> **Note:** The Oracle shared server was called the Oracle multi-threaded server (MTS) prior to Oracle9*i*.

The shared server is configured in the server initialization file (traditionally called INIT.ORA file). Entries such as

```
shared_servers=5
dispatchers="(PROTOCOL=tcp)"
```

are typically found to indicate that a shared server configuration, with associated dispatchers, is in effect for the instance.

Oracle Net Configuration

When you are using the Oracle HTTP Server (Apache) together with `mod_ose` to access the OSE, you might need to configure the TNSNAMES.ORA file. If this file contains an entry called `inst1_http`, you can use that Oracle Net entry for `mod_ose`. If it does not, you must configure a descriptor for `mod_ose`. See "Oracle Net and Oracle Listener Configuration" on page 5-9.

Creating a Web Service

This section describes how to create a single-domain OSE Web service. Later sections describe how to configure a Web domain for the service, and how to create servlet contexts and publish servlets in the domain.

In order to access servlets in the OSE, you must have created a service to handle incoming HTTP requests, as well as a Web domain in which the servlets can run. The service runs in the Oracle server. An Oracle server can support more than one service. For example, you might configure an OSE to serve administrative HTTP requests from one service, and application servlet requests from separate service. You can set up the administrative service so that it has additional privileges not available to the normal application Web service.

Commands

There are two session shell commands that the SYS user must issue to create a Web service: `createwebservice` and `addendpoint`.

To remove an existing Web service, the SYS user issues the `destroyservice` session shell command. To remove an existing endpoint, use the `rmendpoint` command. See to see how the two commands are used.

createwebservice

Issue the `createwebservice` command from the session shell. The syntax for this command is

```
createwebservice -root <location> [optional_parameters] <service_name>
```

The parameters of the command are

<service_name>	Required parameter that specifies the name of the new service. Any JNDI identifier is legal.
-root	Required parameter that specifies the location in the JNDI namespace for the Web service's service root.
-ip	Optional parameter that creates a service that uses IP-based virtual hosting.
-virtual	Optional parameter that creates a service that uses name-based virtual hosting. This parameter can be combined with the -ip parameter.

-properties <prop_ groups>	Optional parameter that specifies a list of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the setgroup command. See the *Oracle9i Java Tools Reference* for information about setgroup.

addendpoint

The addendpoint command is used to specify the endpoints (port numbers) on which a service listens. The syntax for this command is

```
addendpoint -port <port_number> [optional_parameters] <service> <endpoint_name>
```

The parameters of the addendpoint command are

-port	Required parameter that specifies the TPC port number on which the service listens.
<service>	Required parameter that specifies the name of the service, as used in the createwebservice command, that the endpoint will listen on for incoming requests.
<endpoint_name>	Required parameter. Specifies the name of the endpoint. Any JNDI identifier is a legal endpoint name. The endpoint name is used in the rmendpoint command.
-listener	Optional parameter. Specifies the address of the listener to add the endpoint to for this service. If not specified, the endpoint is added to the default listener.
-net8	Optional parameter. This parameter indicates that the endpoint is an Oracle Net endpoint. It is used to establish endpoints for Oracle Net TTC connections, for use by mod_ ose.
-interface <int_spec>	Optional parameter. Specifies the IP address used to connect to this service. The default allows all IP addresses. The IP address specified is mapped to the service domain.
-register	register is an optional parameter that saves the endpoint information in a table in the database. This means that every time the server is started, the specified endpoint is established. This avoids having to reconfigure the INIT.ORA file.
-ssl	Specifies that this is to be an Secure Socket Layer connection.

-threads <min> <max>	The minimum and maximum number of threads for the endpoint. The minimum value is started upon listener initialization; the maximum value is used to deny any more incoming requests.
-timeout	The socket read timeout value in seconds. The amount of time that the Web server will allow you to block on the socket.

Examples

As always in this Guide, '`%`' indicates an operating system prompt, and '`$`' indicates a session shell prompt.

Here are examples of the commands that you can use to create a single-domain service named `testService`, and add named endpoints to it:

```
$ createwebservice -root /testRoot testService
#
$ addendpoint -port 8088 -register testService TestPublic
$ addendpoint -port 9088 -register -ssl testService TestSSL
$ addendpoint -net8 -register testService TestNet8
```

This example adds two dynamically-registered endpoints for connection through an Oracle listener, on ports 8088 and (for SSL) port 9088. The last endpoint is for access to OSE from Apache and `mod_ose`, which uses the normal Oracle Net TTC. If you always connect using `mod_ose`, as recommended by Oracle, you only need to use the last `addendpoint` command.

> **Important Warning:**   When you create a new service, it is usually owned by SYS, because only SYS can add an endpoint to a new Web service.
>
> **For application services, it is very important to remove the SYS privileges from the newly-created service root.**
>
> **Create Web domains in the service when connected to the session shell as the eventual real owner, not as SYS.**
>
> If you do not do these things, servlets that run under the new service and domain(s) inherit SYS privileges, and could browse and modify a database at will, possibly causing irreparable damage.

The following session shell commands (issued as SYS) show how to make the HR schema the owner of his service:

```
$ cd
$ chown -R HR /testRoot
```

rmendpoint

Use this command to remove a previously established endpoint.

The syntax is:

```
rmendpoint [-force] <service> <name>
```

where the `<service>` is the name of the service that the endpoint was originally established for, and `<name>` is the endpoint name that was used in the `addendpoint` command.

If this command results in errors, or does not seem to remove the endpoint (shown by errors when trying to create a new endpoint of the same name on the same service), use the `-force` option. It never hurts to use `-force`.

After creating the service, you can examine the properties of the service using either the `getproperties` command, or more specifically the `getgroup` command, to examine the properties of the `service` group of the newly-created `testService` service. In the `createwebservice` example command above, notice that neither the name-based virtual-hosting (`-virtual`) nor IP-based virtual-hosting (`-ip`) options are specified. This means that neither `service.http.virtual-host` nor `service.http.multi-homed` properties are present in the `service` property group of the `testService` service. You can verify this by issuing the commands

```
$ cd /service
$ getgroup testService service
```

Changing Service Properties

You can change properties of the service using the `addgroupentry` session shell command, but you must be careful if you do this. Changing properties at random, without a real understanding of their function, can cause the service to malfunction or to stop working completely.

Some service properties that you might need to change are:

- `service.globalTimeout` (service timeout value in seconds)
- `endpoint.<endpoint_name>.timeout` (endpoint timeout value in milliseconds)

Creating Multi-Domain Web Services

See "Virtual-Hosted Services" on page 2-24 for general information about multi-domain Web services. To create a multi-domain service, the SYS user issues the session shell command createwebservice, just as for a single-domain service. The difference is that either the -ip or the -virtual options are used. Both options are used to create a Web service that supports both IP-based and name-based virtual hosting.

Examples

Name-Based

Here is an example of session shell commands that create Web domains for a name-based virtual-hosted service:

```
$ createwebservice -root /vhost -virtual vhostService
$ addendpoint -port 8011 -register vhostService ep8011
$ createwebdomain -docroot /private/youruserID/test/docsv1 /vhost/xyz.com
$ createwebdomain -docroot /private/youruserID/test/docsv2 /vhost/xyz.us.com
```

Note that each Web domain under /vhost has the same name as a separate DNS domain name. Now the request URLs

```
http://xyz.com/
```

and

```
http://xyz.us.com/
```

will be served by different Web domains. One domain might be used for one of a company's product lines, the other Web domain to support a different part of the company.

IP-Based

Consider a system that has two network cards. The network interfaces are configured to the IP addresses 10.5.5.10 and 10.5.5.11.

> **Note:** IP addresses must be statically assigned for IP-based virtual hosting support to work. An IP-based virtual hosted Web service cannot work in a subnet where IP addresses are dynamically assigned.

Configure an OSE to support these three network interfaces, each having a separate Web domain, using the following session shell commands:

```
$ createwebservice -ip -root /MHhost MHService
```

Note the properties that are established for the MHService:

```
$ cd /service
$ getproperties MHService
--group--=service
service.name=Service MHService
service.description=Aurora HTTP Servlet Engine
service.version=1.0
service.vendor=Oracle Corp.
service.globalTimeout=60
service.root=/MHhost
service.presentation=http://MHService
service.error.log=servicelogs/error
service.event.log=servicelogs/event
service.http.multi-homed=true
...
```

The last property in this abbreviated list is the one that determines that the Web service supports IP-based virtual hosting.

Now create the domains:

```
# the backslash is a line-continuation character
$ createwebdomain -docroot /private/youruserID/test/docs010.005.005.010 \
        /MHHost/10.5.5.10
$ createwebdomain -docroot /private/youruserID/test/docs010.005.005.011 \
        /MHHost/10.5.5.11
$
```

Creating Web Domains

Before creating one or more Web domains, first create the Web service that hosts the domains. See "Creating a Web Service" on page 3-5 for instructions.

Creating a Single-Domain Web Domain

The single-domain Web domain has a name that is the same as its service root. For example, you created a single-domain service in the example above with the root /testRoot. To create a Web domain under this root, use the createwebdomain session shell command.

When you create a Web domain, you specify a *document root*, using the -docroot parameter. This is a place in the file system on the server's system from which the OSE serves static pages. For example, on a UNIX system create a directory such as /private/youruserID/test/docs. Put an HTML file, perhaps welcome.html, in that directory.

Files in the OS filesystem have access permissions that are associated with the operating system users, for example UNIX or Windows NT logins. But for the OSE to access files in the OS file system, the files must have Java file permissions for the database user who is trying to access the file.

You assign the files Java access permissions through SQL commands. The permissions are assigned to database users, not to OS logins. Use the grant_permission procedure in the dbms_java package to grant (or restrict) Java permissions (java.io.FilePermission) on files in the operating system.

See the Security chapter in the *Oracle9i Java Developer's Guide* for more information about Java file permissions.

Here is an example that creates a Web domain, using the session shell:

```
$ createwebdomain -docroot /private/youruserID/test/docs /testRoot
```

On a Windows NT or Windows 2000 system, create a directory such as C:\test\docs, and issue the create command as:

```
$ createwebdomain -docroot C:\test\docs /testRoot
```

To set the file permissions, use SQL*Plus. For example, to assign read and write permissions to the HR user for the doc_root /private/youruserID/test/docs (and all subdirectories of it) of the createwebdomain example above, enter SQL*Plus as the SYS user and issue the command:

```
SQL> call dbms_java.grant_permission('HR', 'java.io.FilePermission',
```

```
'/private/youruserID/test/docs/*', 'read,write');
```

Once you have established the service, and a service endpoint, the OSE can always access this domain directly. For example, the URL

```
http://Oratest:8080/
```

will access the `/testRoot` domain as long as these conditions are met:

- an Oracle9*i* platform is running on `Oratest`, and the OSE/OJVM is installed

- the DNS servers can find the DNS name `Oratest` and substitute the correct 32-bit IP address for it, so that the request can be routed to the correct machine

In this example, the hostname part of the URL is only used by the DNS servers. The OSE receives the complete URL, as well as the IP address, but does not need to process either to find the Web domain. There is only one Web domain established for that service.

Some machines have multiple entries (multiple host names) in the DNS namespace for the same IP address. For example, the hostnames `Oratest` and `Oratest.us.oracle.com` might map to the same address. (On UNIX-like systems, this can be seen in the `/etc/hosts` file.) Assume that the IP address in this case is 10.5.5.10. So, given the create service and create Web domain conditions above, each of the following URLs gets a request to the same OSE Web domain on the `Oratest` machine:

```
http://Oratest/
http://Oratest.us.oracle.com/
http://10.5.5.10/
```

(In this case, the default servlet for the `testRoot` service is activated by these URLs.)

Creating Servlet Contexts

Before creating a servlet context, you must establish a Web domain in which the context will exist. See "Creating Web Domains" on page 3-11.

There are two ways to create and configure a servlet context for the OSE:

- Use session shell commands.

- Deploy an application to the OSE using a WAR file deployment tool.

This chapter uses the session shell commands in examples, because they show finer-grained control of the servlet context. See Chapter 8, "Oracle WAR Deployment" for information about deploying an application using WAR files.

In the session shell, use the createcontext command to create a servlet context. The syntax of this command is:

```
createcontext -virtualpath <path> [options] <domain_name> <context_name>
```

where the options are

```
[-recreate]
[-properties <prop_groups>]
[-docroot <location>]
[-stateless]
```

Here is an example that uses this command to create a servlet context named HRRoot in the /HRRoot Web domain:

```
$ createcontext -virtualpath /ose/hr -docroot /private/hr/test/html
      /HRRoot HRContext
```

The command parameters and options are:

-virtualpath	This is a required parameter, not an option. Use it to specify a path in the URL that precedes the servlet path. See "Finding the Servlet" on page 2-35 for more information about how the OSE processes the URL, and finds the right servlet context and servlet. The minimum virtualpath is '/'.
<domain_name>	The JNDI name of the Web domain that the context is to be located in. The domain name should be an absolute path. For example, a Web domain in a virtual hosted service (both IP- and name-based) might be /vhost/10.5.5.10/web1.

<context_name>	The name you give the servlet context. This name is arbitrary. It is used when you configure the context, publish servlets to the context, and when you finally destroy the context.
-docroot	Specify the location in the file system of the computer on which the OSE runs where static files (such as HTML files) are kept. The docroot must be specified as an absolute path.
-recreate	If a context with this name already exists, delete it before adding an empty context with this name. Doing this destroys any servlets that were associated with this context before the present `createcontext` command.
-properties	<prop_groups> List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the `setgroup` command.
-stateless	All servlets in this context are stateless. Contexts declared to be stateless can contain only servlets that are stateless. Stateless servlets never try to access the `HTTPSession` object.

Configuring a Servlet Context

When you create a servlet context, the root context has a `config` object. By default, this object has the property groups and property/value pairs shown below, for the `HRContext` that was created in the previous example:

```
$ cd /HRRoot/contexts/HRContext
$ getproperties config
--group--=context.properties
context.browse.dirs=true
context.welcome.names=index.html:index.htm
context.accept.charset=ISO-8859-1
context.accept.language=en
context.default.languages=*
context.default.charsets=*
--group--=context.params
--group--=context.mime
java=text/plain
html=text/html
htm=text/html
body=text/html
--group--=context.servlets
/errors/internal=internalError
--group--=context.error.uris
401=/system/errors/401.html
403=/system/errors/403.html
404=/system/errors/404.html
406=/system/errors/406.html
500=/errors/internal
```

You must configure the new context so that it has the properties that are appropriate for your application. Do this by using the `accesslog` and the `addgroupentry` session shell commands.

accesslog

This command specifies how HTTP access logging is handled for the servlet context. Access logging records information about each incoming HTTP request. The syntax for the `accesslog` command is:

```
accesslog [options] <context_name>
  [-trace]
  [-systable
  [-table <table_spec>]
```

For more information about the required logging tables, see "Logging" on page 3-17.

addgroupentry

Use this command to add or change the values for properties (in property groups) in the `config` file. The syntax for this session shell command is:

```
addgroupentry <object_name> <group_name> <property_name> <property_value>
```

The parameters for this command are:

`<object_name>`	The name of the target object to which properties are to be added, or whose properties are to be modified.
`<group_name>`	The name of the property group containing the property. For example, `context.properties`.
`<property_name>`	The name of the new or changed property.
`<property_value>`	The string or numeric entry for the property value.

You can add the following propertiesto the servlet context `config` object:

context.browse.dirs	If "true", allows the response to list the files in a directory, when the URL ends with a '/'.
context.debug	If "true", sends debug output to the console.
context.default.charsets	The charset(s) that are supported.
context.default.languages	The two-letter standard abbreviation for the languages that are supported. Following the HTTP specification.
context.default.timeout	The timeout for all servlets in the servlet context. The value is in seconds.
context.runAsOwner	Allows servlets in the servlet context to run with the permissions of the JNDI owner of the context, rather than inheriting the permissions of the web domain owner. See "Run As Owner" on page 7-4 for more information on using this property.

Other config Property Groups

context.mime The `context.mime` property group lists the MIME types supported by the context.

context.servlets The `context.servlets` group contains the virtual path mappings for each published servlet in the context.

context.error.uris The `context.error.uris` group shows the mappings for HTTP errors to HTML files, or other files that the OSE responds with when the error occurs.

Logging

The OSE logs events and errors. By default, logs are directed to the `JAVA$HTTP$LOG$` table in the database, which is in the `SYS` schema. If your application requires that you process and have available the logs of events (HTTP requests, for example), and errors (and this is always a good idea), then you should redirect the event and errors to tables in the schema that owns the context.

These are the actions that you must take to enable per-context logging for your application:

- Create SQL tables in the schema to hold the log information, and that match JAVA$HTTP$LOG in structure.

- Create SQL sequences for the logging tables.

- Verify that the system classes that perform event and error logging have been bound into the `/servicelogs` context of the service root, using the `accesslog` tool.

System Classes

Use the session shell `bind` command to bind the event and error handling capability into the `/servicelogs` context of the service. Indicate the names of the tables that you created in the step above to set an attribute for the bound classes. Here are examples based on the `HRService`:

```
bind /HRRoot/servicelogs/event -rebind \
   -c SYS:oracle.aurora.namespace.rdbms.TableStream \
   -f oracle.aurora.namespace.PublishedObjectFactory \
   -string table.name HR.EVENT$LOG

bind /HRRoot/servicelogs/error -rebind \
   -c SYS:oracle.aurora.namespace.rdbms.TableStream \
   -f oracle.aurora.namespace.PublishedObjectFactory \
   -string table.name HR.ERROR$LOG
```

Logging Tables

The names used below are for the tables created for the `HRRoot` demos. You may use any names for your application. There are three tables:

- an HTTP log, here called `<schema_name>.HTTP$LOG$`

- an event log, here called `<schema_name>.EVENT$LOG`

- an error log, that can be named `<schema_name>.ERROR$LOG`

The HTTP log table has the same structure as `SYS.JAVA$HTTP$LOG$`. Here is its description, as produced by SQL*Plus:

```
Column Name                    Type
------------------------------ ----
SERVER_NAME                    VARCHAR2(80)
TIMESTAMP                      DATE
REMOTE_HOST                    RAW(4)
REMOTE_USER                    VARCHAR2(80)
REQUEST_LINE                   VARCHAR2(256)
STATUS                         NUMBER(3)
RESPONSE_SIZE                  NUMBER(38)
REQUEST_METHOD                 RAW(1)
REFERER                        VARCHAR2(80)
AGENT                          VARCHAR2(80)
```

The description of the EVENT$LOG table is:

```
Column Name                    Type
------------------------------ ----
ID                             NUMBER
LINE                           NUMBER
TEXT                           VARCHAR2(4000)
```

And here is the ERROR$LOG table:

```
Column Name                    Type
------------------------------ ----
ID                             NUMBER
LINE                           NUMBER
TEXT                           VARCHAR2(4000)
```

You must also create sequence numbers for use by the event and error logging. The following SQL commands create the required sequences. These are for the HR schema, substitute your own schema name and sequence names, as required:

```
create sequence HR.EVENT$LOG_ID
```

```
create sequence HR.ERROR$LOG_ID
```

Oracle supplies example servlets that you can use or adopt to view the log tables. They are the HttpRdbmsLogServlet and the TableReaderServlet classes. You can use the following session shell commands to publish these servlets into your context. Use the table names that you employed in the CREATE TABLE commands to set the table.name property for the published names.

```
$ publishservlet -virtualpath /http_log HRContext httpLog_viewer \
    SYS:oracle.aurora.mts.http.servlet.HttpRdbmsLogServlet -properties \
    table.name=HR.HTTP$LOG$
$
$ publishservlet -virtualpath /error_log HRContext error_log_viewer \
    SYS:oracle.aurora.mts.http.servlet.TableReaderServlet -properties \
    table.name=HR.ERROR$LOG
$
$ publishservlet -virtualpath /event_log HRContext event_log_viewer \
    SYS:oracle.aurora.mts.http.servlet.TableReaderServlet -properties \
    table.name=HR.EVENT$LOG
```

Timeouts

The timeout values determine how long a stateful session will stay active after the last request. There are two timeout properties for the service. In addition, each servlet context can establish its own timeout value, which takes precedence over the global service and endpoint timeouts. Also, individual servlets can set a timeout value for the session, using the setMaxInactiveInterval() method of the HttpSession interface.

Global Timeout   This is a property of the service group for the service. This value defaults to 60 seconds when a service is created. You can change the default using the createservice command when you create the service, but you cannot change it if using the createwebservice command.

You can see the value of the global timeout by entering these commands in the session shell:

```
$ cd /service
$ getgroup <service_name> service
```

You can also change the global timeout using the addgroupentry command, after the service has been created. Specify the new value in seconds.

Endpoint timeout Each service endpoint can have a separate timeout value, less than or equal to the global session service timeout. You can set the timeout for each endpoint when the endpoint is established, using the `-timeout` option of the `addendpoint` command (value in milliseconds). You can see the timeout values for all endpoints by using these session shell commands:

```
$ cd /service
$ getgroup <service_name> endpoint
```

Servlet Context Timeout Each servlet context can have a default timeout, which applies to all servlets in the context that have not set their own timeout values. The servlet context default timeout is set in the `config` object of the servlet context. (The `config` object for the servlet context is in the root directory of the context.)

This timeout is set in the `context.properties` group of the `config` object, as the property `context.default.timeout`. Set it using the `addgroupentry` command, as follows (using the HR demo context as the example):

```
$ cd /HRRoot/contexts/HRContext
$ addgroupentry config context.properties context.default.timeout NN
```

where NN is the timeout value in seconds.

Publishing Servlets

Use the `publishservlet` session shell command to publish servlets. The syntax of this command is:

```
publishservlet [options] <context_name> <servlet_name> <class_name
```

where the options are:

```
[-virtualpath <path>]
[-stateless]
[-reuse]
[-properties props]
```

The command parameters are:

<context_name>	The servlet context path into which to publish.
<servlet_name>	The name to be given to the servlet in the `named_servlets` directory of the servlet context.
<class_name>	The name of the class in the schema. This is the class that you loaded using the `loadjava` command, or a class that was loaded in a deployment file.
-virtualpath	The path name that will be used in the URL to reach the servlet
-stateless	The servlet is stateless. Stateful is the default property.
-reuse	Add the specified virtual path to an existing servlet without republishing the servlet. This could also be done by adding a property to the `context.servlets` property group in the servlet context `config` object.
-properties	<prop_groups> List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the `setgroup` command. See the *Oracle9i Java Tools Reference* for information about this command.

The `publishservlet` command does two things:

- Binds a reference object to the servlet class into the `named_servlets` directory (context) of the servlet context.
- Adds a mapping between the virtual path you specify in the command and the servlet name in the `named_servlets` directory.

The JNDI object that is bound into the `named_servlets` directory contains the servlet class name and any initialization parameters for the servlet, as specified in the `-properties` option of the `publishservlet` command. You can see the class name and other properties of a published servlet by using the `getproperties` session shell command on the published object.

Summary

This section reviews and summarizes information that has been presented in previous sections. It presents a complete scenario, in which the following steps are performed:

- Creating a Web Service
- Creating a Web Domain
- Creating a Servlet Context
- Creating a Servlet
- Compiling the Servlet
- Loading the Servlet into the Database
- Publishing the Servlet
- Accessing the Servlet
- Adding Logging Tables
- Adding Security

Where it is useful, the results of a step are examined. For example, after many steps that use session shell commands, the `ls` and `getproperties` commands are used to examine the results of the step. SQL*Plus is also used to show the results when objects are added to the database. If you follow each step in this Summary, you should have a better understanding of how to use the Oracle Servlet Engine to develop servlet-based applications.

Throughout this section, as in the remainder of this guide, we use the following notational conventions:

- `%` indicates an operating system prompt. Do not enter the '%'.
- `$` indicates a session shell prompt. Do not enter it.
- `SQL>` is a SQL*Plus command prompt.
- $ORACLE_HOME is the directory where Oracle is installed.
- The character '\' (backslash) in a command line means line continuation. Do not put a return (ENTER or RETURN key) in that place. The line is broken for page formatting purposes only.

Remember that each of the commands used is also documented in earlier sections of this chapter and in the *Oracle9i Java Tools Reference*.

Creating a Web Service

This step creates a Web service that supports a single Web domain. The service is called the `testService`, and is created to be used by the familiar schema `HR`.

Step 1a

To start off, enter the session shell as the SYS user:

```
% sess_sh -user SYS -password change_on_install -role SYSDBA -service \
    http://localhost:8080
```

> **Notes:** The session shell must be on your command path. It is in `$ORACLE_HOME/bin`.
>
> If `localhost` does not work in your environment, substitute the name of the host on which the OSE/OJVM is running.
>
> Port 8080 is the default port for the OSE/OJVM `admin` service, which supports the session shell on the server side.

Once in the session shell, list the contents of the root directory of the JNDI namespace:

```
$ ls /
```

In a newly installed Oracle platform, you should see something like this:

```
bin/        HRRoot/    system/
etc/        service/   test/
```

Some of these "directories" (or JNDI contexts) might be absent, depending on your installation. But the `bin`, `etc`, `service` and `system` entries must be there. `HRRoot` is the root of the `HRService`—a service and Web domain that supports the demonstration servlets that are installed with Oracle9*i*.

> **Oracle8*i* Users:** The `HRService` demos are not available in Oracle8*i*, Release 3. See the OTN Web page for more information.

Step1b

Create the Web service:

```
$ createwebservice -root /testRoot testService
```

```
$ ls
bin/          service/
etc/          system/        testRoot/
```

Verify that the service object was created in the /service context:

```
$ ls /service
admin   testService    HRService
```

Yes, it's there. Now add one endpoint to the service:

```
$ addendpoint -port 8088 -register testService testEndPoint
```

Look at the testService object in the /service context. What does it contain?

```
$ getproperties /service/testService
--group--=service
service.name=Service testService
service.description=Aurora HTTP Servlet Engine
service.version=1.0
service.vendor=Oracle Corp.
service.globalTimeout=60
service.root=/testRoot
service.presentation=http://testService
service.error.log=servicelogs/error
service.event.log=servicelogs/event
--group--=endpoint
endpoint.class=SYS:oracle.aurora.mts.ServiceEndpoint
endpoint.name=testEndPoint
endpoint.testEndPoint.interface=*
endpoint.testEndPoint.port=8088
endpoint.testEndPoint.backlog=50
endpoint.testEndPoint.min.threads=3
endpoint.testEndPoint.max.threads=5
endpoint.testEndPoint.timeout=30000
--group--=environment
--group--=contexts
--group--=mime
java=text/plain
html=text/html
...
```

Most of the properties under the group service are default values for the createwebservice command. Likewise the properties in the group endpoint

are defaults added by the `addendpoint` command. Only service name and the endpoint port number are not defaults.

It is important to now make the schema HR the owner of the objects in the service. Not doing so would open a big security hole in the server.

```
$ chown -R HR /testRoot
```

Verify the ownership:

```
$ ls -l /testRoot
Read     Write     Exec      Owner     Date Time      Type        Name
SYS      SYS       SYS       HR        Feb 24 12:02   Context     servicelogs
```

Note that SYS remains the owner of the service object:

```
$ ls -l /service/testService
Read     Write     Exec      Owner  Date Time     Type      Name
SYS      SYS       SYS       SYS    Feb 24 12:02  Service   service/testService
```

This is normal, and is fact is required. You now have a working Web service, that supports a single Web domain.

Creating a Web Domain

The Web domain is created in the `testRoot` service, and has that name. Because this is a single-domain Web service, the name of the Web domain is arbitrary. When you create a Web domain, you also specify the default document root for the domain. You specify a directory on a file system of the host machine. You can put a `welcome.html` file in there to be served as the default page for the domain. (The file names and locations are arbitrary—you do not have to imitate the ones shown here.)

Before creating the Web domain, have a look at the service root:

```
$ cd
$ ls testRoot
servicelogs/
```

Note that the only object there is the `servicelogs` context. This is created by the `createwebservice` command. Now reconnect to the session shell as the HR user, and create the Web domain. First exit from the session shell, then restart it as the HR user:

```
$ exit
%
```

```
% sess_sh -user HR -password hr -service http://localhost:8080

$ createwebdomain -docroot /tmp/testDomain /testRoot
```

Look again at the `/testRoot` context:

```
$ ls /testRoot
config          contexts/       logs/           servicelogs/
```

The `createwebdomain` command created a `contexts` "context", a `logs` context, and a `config` object in the root of the Web domain. What does the `config` object contain? Look at it:

```
$ cd /testRoot
$ getproperties config
--group--=environment
--group--=contexts
--group--=mime
java=text/plain
html=text/html
htm=text/html
body=text/html
xml=application/xml
...
# (and so on)
```

Not too much of interest here yet. We will come back to this `config` object later.

Important: Set the Java Permissions on the Document Root

This is a very important step. For clients to access the files in the document root, you must set the *Java* permissions on these files so that users can access them.

> **Note:** Java IO file permissions are granted from the database, using SQL commands. ***They are not the same as operating system permissions or access rights***, even though the files in the document root are static files in the OS filesystem.

You must first connect to the database as a user who has permissions to grant Java file access permissions to the required users. In this example we connect as the SYS user, but you can use any database user who has the right grant capability.

```
% sqlplus 'SYS/change_on_install as SYSDBA'
SQL> call dbms_java.grant_permission('HR', 'java.io.FilePermission',
```

```
'/tmp/testDomain/*', 'read,write');
SQL> Call completed.
SQL> exit
```

The OS directory does not have to exist when you grant these permissions. See the Security chapter of the *Oracle9i Java Developer's Guide* for more information about file permissions.

Load an HTML page into the document root, using OS commands. Here is an example welcome page that you can use:

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Welcome to the Test Context</title>
</head>

<body>
<h1 align="center"><b>Welcome to the Test Context</b></h1>
<p>This page is located in the doc root of the <i>test</i> context</p>
<p>The servlet context testContext is located at <a href="/ose/">/ose</a>
</body>

</html>
```

Bind the Classes for Service Event and Error Logging

When you create a Web service, you should bind the classes that perform the event and error logging into the service. See "System Classes" on page 3-17. Use these commands while connected to the session shell as SYS:

```
bind /testRoot/servicelogs/event -rebind \
    -c SYS:oracle.aurora.namespace.rdbms.TableStream \
    -f oracle.aurora.namespace.PublishedObjectFactory \
    -string table.name HR.EVENT$LOG

bind /testRoot/servicelogs/error -rebind \
    -c SYS:oracle.aurora.namespace.rdbms.TableStream \
    -f oracle.aurora.namespace.PublishedObjectFactory \
    -string table.name HR.ERROR$LOG
```

Creating a Servlet Context

Connect to the session shell as the HR user, if you are not already there:

```
% sess_sh -user HR -password hr -service http://localhost:8080
$
$ createcontext -virtualpath /ose -docroot /tmp/testDomain
      /testRoot testContext
```

Creating a Servlet

This section contains the code for an example servlet that you can use. The servlet accesses the HR.EMPLOYEES table in the database, which is part of the HR sample schema.

The servlet code uses JDBC statements to query the database. Note that the server-side internal driver (KPRB driver) is used. See the Oracle9*i JDBC Developer's Guide and Reference* for more information about this driver.

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import oracle.jdbc.*;

public class simpleHRServlet extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
    {
      PrintWriter out = new PrintWriter(resp.getOutputStream());
      Connection conn = null;

      try {
        // connect with the server-side internal driver
        OracleDriver ora = new OracleDriver();
        conn = ora.defaultConnection();

        if (conn != null) {
          Statement stmt = conn.createStatement();
         ResultSet rset = stmt.executeQuery("select EMPLOYEE_ID, LAST_NAME, HIRE_DATE from
                                                HR.EMPLOYEES order by HIRE_DATE");

          resp.setContentType("text/html");
          out.println("<html><head><title>Servlet JDBC Example</title></head><body>");
```

```
        out.println("<table border=3 cellspacing=2 cellpadding=4 bgcolor=#FFFFE7>");
        out.println("<tr><td>Employee Number<td>Last Name<td>Date Hired");

        int counter = 0;
        while (rset.next()) {
          out.println("<tr><td>" + rset.getInt(1) + "<td>" + rset.getString(2) + "<td>" +
             rset.getDate(3));
           counter++;
        }

        out.println("</table><br>");
        out.println("A total of "+ counter + " records");
        out.println("</body></html>");
        rset.close();
        stmt.close();
        conn.close();
     }

   } catch (java.sql.SQLException e) {
      e.printStackTrace(out);
      }

   out.flush(); out.close();
 }

 public void init(ServletConfig cfg) throws ServletException {
   super.init(cfg);
 }

 public void destroy() {
   super.destroy();
 }

 public String getServletInfo() {
   return "A simple JDBC servlet";
 }
}
```

Compiling the Servlet

Use a Java compiler compliant with JDK 1.2 to compile the source on the client.
Make sure that `servlet.jar`, `dt.jar`, `tools.jar`, and Oracle's

classes12.zip are all on the classpath when you compile. Here is a Solaris example, but you might need to modify this for your Oracle installation:

```
% javac -g -classpath .:$ORACLE_HOME/lib/servlet.jar:$ORACLE_
HOME/jdbc/lib/classes12.zip:$ORACLE_
HOME/sqlj/lib/translator.zip:/usr/local/jdk1.2.2/lib/dt.jar:/usr/local/jdk1.2.2/
lib/tools.jar simpleHRServlet.java
```

Loading the Servlet into the Database

Use the loadjava command to load the compiled class simpleHRServlet.class into the database, as follows:

```
% loadjava -verbose -oracleresolver -resolve -oci8 -user HR \
    -password hr simpleHRServlet.class
```

Publishing the Servlet

Use the session shell to publish the servlet to the OSE/OJVM, as follows:

```
% $ORACLE_HOME/bin/sess_sh -user HR/hr -service http://localhost:8080
    -command "publishservlet -virtualpath /simpleServlet
                /HRRoot/contexts/HRContext  simpleHRServlet HR:simpleHRServlet"
```

Note that the virtual path /simpleServlet was assigned to the class simpleHRServlet in the command part of the session shell invocation.

Accessing the Servlet

We can now combine all the elements that we created in the previous sections to derive the complete URL that is needed to access the simpleHRservlet. They are shown in Figure 3–1:

Figure 3–1   Accessing the Servlet: the URL
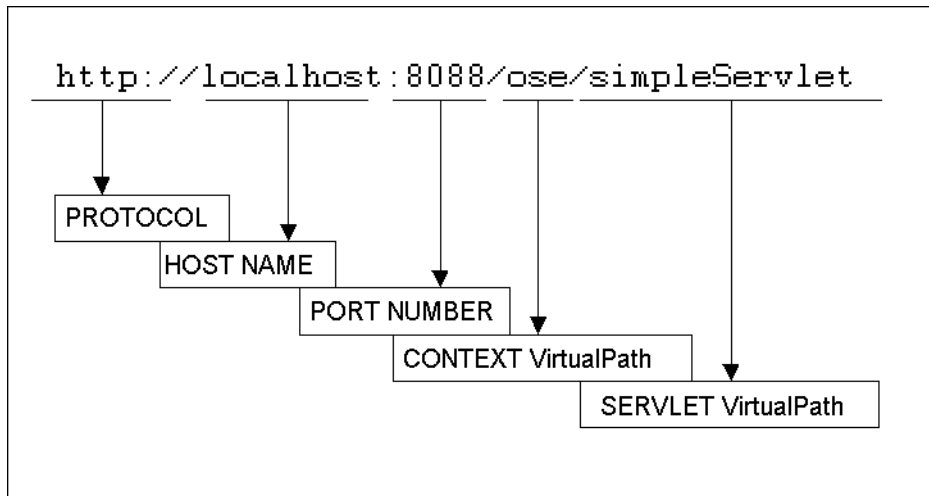
Use a Web browser to access the servlet, entering the URL shown in Figure 3–1.

Connection to the OSE

In the examples in this section, we have set up an endpoint for the Oracle listener, and the URL here assumes a direct connection to the Oracle listener and dispatcher. In your applications use Apache/mod_ose to access servlets. In that case, use the Apache listener port instead of **8088**. If Apache is running, and mod_ose has been configured correctly, with a /ose/ Location directive, the rest of the URL can serve as is.

Adding Logging Tables

Each Web domain requires tables in the database that hold event and error logging information. Create these using SQL*Plus or another Oracle administrative tool. See "Logging" on page 3-17 for more information.

```
-- The HTTP log table:
create table HR.HTTP$LOG$ (
      server_name VARCHAR2(80),
      timestamp DATE,
      remote_host RAW(4),
```

```
        remote_user VARCHAR2(80),
        request_line VARCHAR2(256),
        status NUMBER(3),
        response_size INTEGER,
        request_method RAW(1),
        referer VARCHAR2(80),
        agent VARCHAR2(80))
/

-- Here's where the event log is stored:
create table HR.EVENT$LOG ( id number, line number, text varchar2(4000) )
/
-- The event logging sequence:
create sequence HR.EVENT$LOG_ID
/
-- Here's where we store the error log
create table HR.ERROR$LOG ( id number, line number, text varchar2(4000) )
/
-- The sequence for controlling ordering in the error log table
create sequence HR.ERROR$LOG_ID
/
-- don't forget to commit, if auto commit is not on
commit;
```

Execute these statements using SQL*Plus or an Oracle database management tool.

Adding Security

In the simple example in this Summary section, we have said little about security. We did make sure that the JNDI objects published under the testService root were not owned by SYS, and we did make sure that the static HTML files under the doc root were accessible by at least the HR user.

But if you are using this simple scenario as a basis for a real Web application, you must make sure that you properly address client authentication and authorization issues. See Chapter 7, "Oracle Servlet Engine Security", in this guide. If you are using Oracle Single Sign-On, you should also refer to the documentation about mod_osso in "Using mod_osso with mod_ose" on page 4-14.

4

An Apache Module for OSE

This chapter describes `mod_ose`, the module that you use to connect to the OSE/OJVM from the *Oracle HTTP Server*. The Oracle HTTP Server (powered by Apache) is the Web server that runs in the middle tier, listens to HTTP requests, serves static pages to HTTP clients, and can forward HTTP requests to be handled by servlets or JSPs to the Oracle Servlet Engine running in the data tier.

This chapter describes the basic concepts and use of `mod_ose`. For specific information about configuring `mod_ose` to use with your applications, see Chapter 5, "Configuring mod_ose".

For additional Apache Web server documentation navigate to the Apache Web site at `http://httpd.apache.org/` or read one or more of the trade books that cover the Apache server.

Here are the topics covered in this chapter

- Overview

- Requirements

- mod_ose Connections

- Servlet Access Using mod_ose

- Secure Socket Layer Connection

- HTTP Request and Response Processing

- The AuroraLocationService Directive

- Topology of a Site Using mod_ose

- Using mod_osso with mod_ose

Overview

The Oracle HTTP Server, available with the Oracle Internet Application Server (iAS) Release 1 and to be available with Release 2, is a Web server powered by the Apache server engine. The Oracle HTTP Server is used with the OSE/OJVM for Web applications that serve many static Web pages, or that deploy stateless servlets to the Oracle9*i* server.

Why Use mod_ose?

When you combine Apache with the OSE/OJVM, you get an efficient applications solution that can serve static HTML pages directly from the file system of the server on which Apache is running, and can serve dynamic content from Java servlets and JSP pages. The servlets and JSP pages run under the OSE/OJVM, on the database server, and can benefit from the OSE's fast and efficient access to SQL data.

If you are using the OSE/OJVM and your application makes use of stateless servlets, you should always use mod_ose. Not using mod_ose, and connecting directly to the OSE/OJVM, could cause each HTTP request to start up a new database session and a new OJVM. This can be very expensive, both in the time involved and the resources required on the OSE/OJVM server. When you use Apache and mod_ose, a stateless connection is kept open to the Oracle instance that is running an OSE/OJVM session for each *httpd* process running on the Apache server system. A new client request, or a new stateless HTTP request from a previous client, can thus re-use existing connections. This avoids the overhead of opening a new OSE/OJVM session on the data tier server for each stateless HTTP request, and enables new HTTP requests to use any static information cached on the server by previous stateless requests.

Apache Architecture

An Apache Web server consists of the server, with its basic listening functionality, and a number of plug-ins or modules, which customize the Apache server and add functionality to it for specific applications. For example, the mod_perl module allows the Perl language interpreter to run inside each Apache httpd process, providing better efficiency for CGI or other scripts that use Perl. And the Oracle HTTP Server offers the mod_plsql module, that lets Apache access PL/SQL code running in an Oracle9*i* server.

mod_ose is the Oracle HTTP Server module that serves as a conduit from Apache to the OSE/OJVM. When mod_ose is configured into Apache, each httpd process that is started by the Apache listener incorporates mod_ose. Note that each

httpd/mod_ose process is independent of the others. These processes cannot share data.

mod_ose forwards HTTP requests that invoke servlets and JSP pages to the Oracle server, using a standard Oracle Net connection. This is the same type of connection that an OCI or a Forms application uses.

> **Note:** The communications protocol for the mod_ose connection uses HTTP tunnelling over the Oracle Net connection. The Oracle listener strips off the Oracle Net headers before the request is handed to the OSE, so the OSE sees it as a normal HTTP request.

Figure 4–1 shows how the Web server and the httpd/mod_ose processes relate to the OSE/OJVM.

Figure 4–1   OSE/mod_ose Components
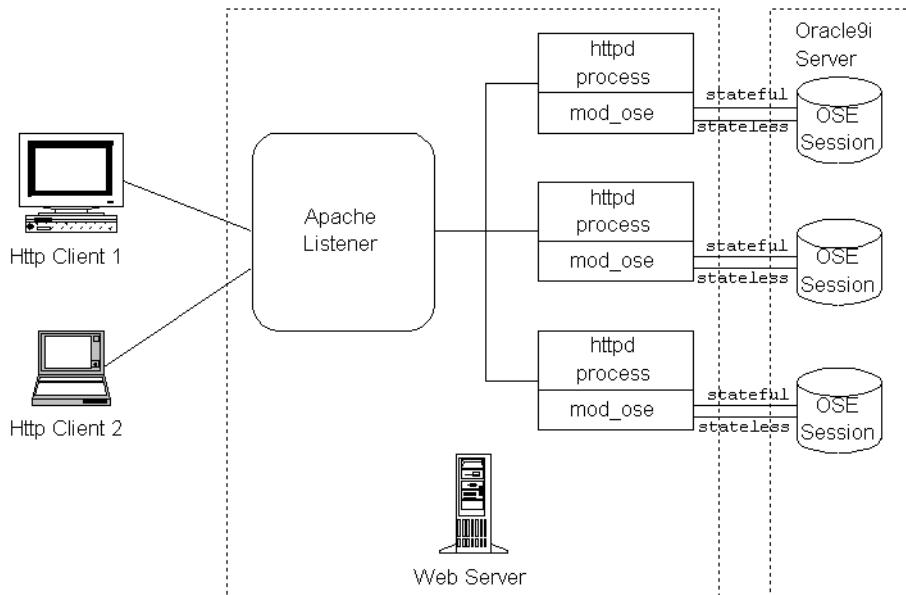
Configuration

The configuration of mod_ose determines which URLs cause HTTP requests to be forwarded to the OSE for processing, and which URLs are processed by the local Apache listener.

There are two files that are used by the Oracle HTTP Server to configure Apache and mod_ose. These are httpds.conf and ose.conf. Specific information about setting up these configuration files is available in Chapter 5, "Configuring mod_ ose".

Requirements

To use Apache and mod_ose, you must have the following components and capabilities:

- an installation of the Oracle Internet Application Server, which contains the Oracle HTTP server

- the mod_so library must be installed in the Oracle HTTP Server, or there must be other equivalent support for Dynamic Shared Objects (mod_so.c is compiled into the Oracle HTTP Server by default)

- mod_mime must be installed in the Oracle HTTP Server

- the machine executing the Oracle HTTP Server must have at the least an Oracle client-side configuration, including

    - Oracle Net networking support

    - an Oracle Net tnsnames.ora file to describe connect identifiers, **or**

    - support for other Oracle name services, such as an LDAP service

- if you are using the Oracle single sign-on server, you must have installed and configured mod_osso to use with mod_ose (see "Using mod_osso with mod_ose" on page 4-14)

Apache can run either on the same machine as the Oracle database server, or on one or more different machines. See the whitepapers from Oracle's Java Platform Group and the Internet Applications Group for more information about efficiently configuring a large application using both the Oracle HTTP Server and Oracle9*i*.

To configure mod_ose, see Chapter 5, "Configuring mod_ose".

Shared versus Dedicated Servers

The OSE runs under the JVM, and so must run in an Oracle9*i* shared server environment. (This is called the multi-threaded server for Oracle8*i*.) If you do not normally run a Oracle shared server configuration, it is still possible to use mod_ose by dedicating a dispatcher to a mod_ose channel. This can be done with adjustments to the Oracle listener's TNSNAMES.ORA initialization file. See the section "Non-Shared Server Installations" on page 5-12 for more information.

mod_ose Connections

mod_ose can provide either a stateless or a stateful channel to the OSE. Both a stateful and a stateless connection is opened by each httpd/mod_ose process started by the Apache listener, and the appropriate connection is used depending upon the requirements of the Web container object (servlet or JSP) and the HTTP client. The properties of the two connection types are:

stateless   Each httpd/mod_ose process keeps an Oracle Net connection open to the OSE server, and reuses this connection for HTTP clients and requests. The connection is permanently associated with a session in the OSE/OJVM server, so that a session gets serially reused by clients that are serviced by the corresponding Apache httpd/mod_ose process. In the stateless mode there are as many OSE connection/session pairs as there are httpd/mod_ose processes.

stateful   The OSE/OJVM associates a session, either an Oracle server or a data cache session, with each Apache client. Each OSE/OJVM session has a unique ID, which is sent back to the HTTP client in a cookie. The cookie is used to route further requests to the correct session. The stateful connection is kept according to the following criteria:

- If *KeepAlive* (set by the HTTP client) is ON, then the stateful connection is kept open for the duration of the HTTP client.

- If KeepAlive is OFF, then the stateful connection is closed after each client request.

In the current implementation routing only happens at connect time. This means that if you want to send a request to a particular session you must have a connection to this session. Because multiple requests from the same client can go to different Apache httpd/mod_ose processes, if *stateful* connections between mod_ose and the OSE server were kept open, the number of open connections could soon become excessive.

Servlet Access Using mod_ose

When the Apache listener receives an HTTP request from a client, it determines whether the request should be handled directly by Apache, or whether the request should be forwarded to a database server for processing.

The way the request is handled is determined by the total Apache configuration, both in the main Apache configuration file, which is ($APACHE_HOME/Apache/conf/httpds.conf), and in the mod_ose configuration file, which is located in $APACHE_HOME/modose/conf/, and is called ose.conf.

> **Note:** $APACHE_HOME on UNIX systems is normally $ORACLE_HOME/Apache. %APACHE_HOME% on Windows NT is %ORACLE_HOME%\Apache.

Apache usually serves static Web pages from the file system on its host system. For example, the host is dlsun164, Apache is listening on port 7777, and a request arrives that is

```
http://dlsun164:7777/
```

In this case, there is no URI (URL-path, or information in the URL after the hostname and the port), so Apache serves its default static HTML page. The location and name of the default HTML page is set up in the Apache main configuration file, using the DirectoryIndex directive.

After you have installed and configured both the Oracle HTTP Server and mod_ose, an HTTP client can send requests to the HTTP server, and have them forwarded to the OSE/OJVM server. The client must know the port number that Apache is listening on, if it is different from the default port 80. For example, if the Oracle HTTP Server is running on the syhstem known as dlsun164, and is listening for incoming requests on port 7777, then it can process a request from an HTTP client with a URL such as

```
http://dlsun164:7777/index.html
```

and serve the HTML page index.html, which is located in the document root of the Oracle HTTP server. Now, suppose that the HTTP client sends a URL such as

```
http://dlsun164:7777/ose/simpleServlet
```

This request is sent on to the OSE/OJVM if mod_ose has been configured to have the OSE serve pages or programs where the URL-path is /ose/* (or, specifically, /ose/simpleServlet). This routing information is located in the mod_ose configuration file ose.conf.

The module configuration file contains entries (directives) of the form <Location>, which specify the service that handles requests to Java servlets or

JavaServer Pages. For example, if the configuration file contains a `Location` entry with an Apache `SetHandler` directive:

```
<Location>
  SetHandler aurora-service /ose/*
  ...
</Location>
```

then all requests with the URL-path `/ose/<any servlet name>` are forwarded to the OSE/OJVM for processing. The OSE activates the servlet or JSP, and sends the servlet response object back to the HTTP client, through Apache. The `httpd/mod_ose` process does not process the servlet response, other than to perform de-chunking (see "Chunking" on page 4-10).

You can find specific information on configuring the Apache listener (through the `httpds.conf` file) and `mod_ose` (through the `ose.conf` file) in Chapter 5, "Configuring mod_ose".

Secure Socket Layer Connection

You can use `mod_ose` for Secure Socket Layer (SSL) connections to the Oracle server. For instructions on configuring `mod_ose` for SSL, see "SSL Configuration" on page 5-20.

HTTP Request and Response Processing

Processing the URL

When an HTTP request comes through Apache to mod_ose, the initial path information in the URL is discarded when the path is sent to the OSE/OJVM. This includes the protocol, Web domain, and port information.

Assume for example that Apache is listening on port 7777, and that the URL that Apache receives is

```
http://dlsun164:7777/ose/simpleRequest
```

This request is forwarded to the OSE/OJVM whenever the ose.conf configuration file specifies Location and SetHandler Apache directives that configure URLs with /ose/ to go to the OSE. For example, the ose.conf file contains

```
<Location /ose/ >
  SetHandler aurora-server
</Location>
```

In this case, all requests in which the URI starts with /ose/ are forwarded to the OSE for processing.

mod_ose strips off the protocol (http://), domain (dlsun164), and port (7777) information from the URL before forwarding it to the OSE.

After the servlet is executed, the servlet response is returned to the client. In general, the response is not altered by mod_ose or Apache. For one exception, see "Chunking" on page 4-10.

In the preceding example, if the OSE finds a servlet context with the virtual path ose/, and a servlet with the virtual path simpleRequest, then the servlet with the virtual path simpleRequest is activated to process the request and send a response back to the client, through mod_ose and Apache. If the servlet context or servlet is not found, then the OSE sends back a 404 HTTP error. (Or, it might send back the contents of an error page that is application specific, depending upon how the application is configured in OSE.

Chunking

The HTTP 1.1 specification mandates how a server should close or hold open a connection. Whether a connection is closed or held depends on the setting of

`KeepAlive` (by the HTTP client) and `ContentLength` (set by the servlet in the `setContentLength()` method of the `ServletResponse` interface).

When a connection is closed, the client reads the response up to the EOF. However if a connection is kept open, the client must know how many bytes to read to get the entire response. The following conditions determine this:

- If `KeepAlive` is OFF, the servlet engine always closes the connection. In this case, the client can read all the data in the response, up until EOF is detected.

- If `KeepAlive` is ON and the `ContentLength` of the response is **set** by the servlet, the connection is kept open.

However, a servlet might not set the content length parameter in its response to the client. Therefore, in order to keep stateless connections to the OSE database alive, `mod_ose` uses HTTP chunking for the responses from OSE to the Apache process. HTTP chunking is set by including the value "chunked" for transfer-encoding in the Transfer Encoding header field.

The responses are de-chunked before being passed to the client. A side-effect of this is that `mod_ose` always removes the content length header from the response, and so HTTP clients will disconnect the connection to Apache/`mod_ose`. This is not a major drawback, as most proxy servers, and most browsers, close connections by default after every request is processed.

For more technical information, see RFC 2616, section 3.6. This RFC is available at:

`http://www.w3.org/Protocols/`

Session ID for Parallel Clusters

When a connection is being made through `mod_ose` to a specific instance in a parallel cluster, the JSESSION cookie ID that is transmitted in the cookie that OSE sends identifies the instance as well as the session. In this case an alphanumeric identifier for the instance is appended to the session ID, following a semicolon. For example

`JSESSION=0235;<someInstanceName>`

The AuroraLocationService Directive

mod_ose normally connects to a single OSE service. This is the service specified by the AuroraService statement in the ose.conf configuration file. For example, the entry

```
AuroraService inst1_http
```

in the ose.conf file names a connection identifier in the tnsnames.ora file that connects to a specific Oracle listener. This is so because the connect string that mod_ose uses has a PRESENTATION clause that is a global parameter, used by all mod_ose requests.

This can be a limitation in some contexts. There are occasions when it would be desirable to have instances of mod_ose in a single Oracle HTTP Server that can connect to, say, an administration service and an application service. Or there might be a standard TCP protocol specified in one connect descriptor, and a Secure Socket Layer (TCPS) protocol in another connect descriptor. Or, you might want to connect to a different port number. In each of these cases a different connection identifier is required in the tnsnames.ora file. As an example, the connection identifier inst1_http connects to a standard service, inst1_https connects to an SSL service, and inst1_httpAdmin directs connections on standard TCP/IP to an administrative service.

To get around this limitation, Oracle9*i* provides the AuroraLocationService directive. This directive is set in the Apache Location directive of the ose.conf configuration file, and can route HTTP requests to more than one service, based on the servlet context specified in the Location directive.

See "AuroraLocationService" on page 5-17 for specific information about configuring the AuroraLocationService directive.

> **Note:** The AuroraLocationService directive is available only with the Oracle9*i* version of mod_ose.

Topology of a Site Using mod_ose

When using mod_ose, you can configure different network topologies with the system in your Web environment. Specifically, you can define configurations that do not have a single point of failure. In such configurations, when a node failure occurs, any available Oracle Listener can redirect requests to some other database instance if the one being used for the client state has failed. For this to work, the application must have been replicated in all nodes, and it must be able to handle the recovery from an expired database session.

You can create multi-node, in-tandem configurations. With these types of configuration, you can make a more scalable service than you can with OSE alone or with Apache alone. Before determining the number and types of server that your network requires, you should understand the basic configuration of the two servers: Apache and OSE. Figure 4–1 shows how the components relate.

With a multi-node server arrangement there is:

- no single point of failure
- fail-over configuration
- load balancing functionality

Since mod_ose works in a network in conjunction with Oracle Net, Oracle Net takes care of the fail-over and load balancing. The *Oracle Net Administrator's Guide* describes these provisions of Oracle Net.

Using mod_osso with mod_ose

HTTP security consists of two parts:

authentication	The process of determining the identity of an HTTP client making a request, and ensuring that the client really is who it claims to be. The simplest form of verification is asking a client for a password. Another type of authentication involves the client attaching a digital certificate to its request.
authorization	Enforcing the rules of access to different documents and applications on the Web server and servlet container. Authorization is based on information provided by a successful authentication step—the name of the client, also called the user or *principal*.

Oracle HTTP Server includes a module, *mod_osso*, that allows you to protect virtual paths on the server (authorization), while delegating the authentication step to the Oracle Single Sign-On Server (called the OSSO Server for short).

The basic function of `mod_ose` is to let the Oracle HTTP Server forward requests for certain virtual paths, so that they are actually executed by the OSE/OJVM.

It is possible to apply both `mod_osso` and `mod_ose` to the same virtual path. In Apache terminology this is described as *chaining* `mod_ose` after `mod_osso`.

For example, for a URL-path consisting of a servlet context virtual path plus a servlet virtual path, you can delegate user authentication to an OSSO server, check permission on a given virtual path for a particular authenticated user, and then forward the request to the OSE/OJVM for execution.

In this scheme, the advantages of Oracle Single Sign-On are preserved. That is, if the client authenticates successfully, it is not asked to repeat authentication for requests to other protected URLs in either the Oracle HTTP Server or in the OSE/OJVM, and the client credentials will be propagated to the OSE/OJVM for all requests that are forwarded there.

When `mod_osso` is part of the module chain forwarding requests, servlets are able to get the user name and a principal object using standard methods of the class `javax.servlet.http.HttpServletRequest`, namely `getRemoteUser()` and `getUserPrincipal()`.

The values returned for these methods will be based on the user identity, which was established during successful Oracle Single Sign-On.

The Oracle HTTP Server provides directives for fine-grained authorization which is functionally very similar to the same HTTP security protections that are provided by the OSE. These directives may be used independently, or in conjunction with OSSO authentication. The set of virtual paths redirected to OSE is a subset of all virtual paths serviced by the Oracle HTTP Server, so the same paths can be protected by both mechanisms.

When regular HTTP security is used with the OSE/OJVM, the user identity is not known on the Apache front end, so the only way to define access control is by using OSE HTTP authorization. However when you use OSSO authentication the situation is different. A call to the OSSO Server happens on Apache, and the user is known after that. If an access control rule requires checking the user's membership in a group, that information also needs to be obtained from the OSSO server. So, all virtual path protection should be configured on the Apache node, and any configuration for contexts protected with OSSO that may exist on OSE will be ignored.

For information about configuring the mod_osso module, see "Configuring mod_osso" on page 5-22. For more information about the Oracle Single Sign-On server, see the *Oracle Single Sign-On Application Developer's Guide*, part of the Oracle9*i* Internet Application Server documentation set.

5

Configuring mod_ose

This chapter describes how to configure the Oracle HTTP Server (powered by Apache) and the Apache `mod_ose` module that is used to connect to the Oracle Servlet Engine (OSE/OJVM).

This chapter contains the following topics:

- Steps to Take
- Configuration Files
- Oracle Net and Oracle Listener Configuration
- Generating a Configuration File
- Non-Shared Server Installations
- Configuration Utilities
- AuroraLocationService
- Specifying Stateful and Stateless Handlers in ose.conf
- SSL Configuration
- Configuring mod_osso
- Troubleshooting

Steps to Take

To set up the Oracle HTTP Server and the `mod_ose` module that is used to connect Apache to the Oracle Servlet Engine (OSE/OJVM), you need to take the following steps:

1. Make sure that the OSE/OJVM is running on the server.

   The OSE/OJVM requires an Oracle Shared Server configuration. If you do not normally have or use this configuration, see "Non-Shared Server Installations" on page 5-12 for a workaround.

2. Install a test servlet that you can access on the server. The examples in this section use an example servlet called `DBSession`, which is a stateful servlet. See "Examples" on page 3-9 for the instructions on coding and installing this servlet.

3. Be certain that you have the Oracle HTTP Server installed on an accessible system. The Oracle HTTP Server is part of the standard Oracle8*i* and Oracle9*i* Internet Application Server distributions.

4. Configure the Oracle server listener so that it accepts incoming Oracle Net requests from Apache/`mod_ose`. See "tnsnames.ora" on page 5-10 for specific information.

5. Edit the Apache configuration file to include the `mod_ose` configuration file. See "Including Configuration Files in httpds.conf" on page 5-8 for specific information.

6. Generate or edit a `mod_ose` configuration file for your application. For the test application described in this chapter, the file is shown in "Example ose.conf" on page 5-7.

7. Start or restart the Apache server, so that the newly-changed configuration files are read by Apache.

8. Try accessing the servlet using a Web browser.

9. If you cannot access the servlet, see the section "Troubleshooting" on page 5-25.

Starting mod_ose

The mod_ose module is configured into Apache by including its configuration file in the main Apache configuration file. In the mod_ose configuration file, the first line (apart from any comments) points to the location of the dynamic library that implements mod_ose, using the LoadModule directive. For example:

```
LoadModule ose_module     /private1/Apache/modose/bin/libjipa9i.so
```

Once mod_ose module is part of Apache, each daemon Apache process (httpd process) that the Apache listener starts includes mod_ose. There is no need to specifically start mod_ose. This is the normal way that Apache modules are included in the httpd processes.

See for more information about the LoadModule directive.

Configuration Files

When an Apache process starts up, it reads the Apache configuration file to determine how it should run. The Apache configuration file, and additional configuration files that it includes, determine which modules are included in Apache, and thus how HTTP requests are handled: whether they are handled directly by Apache, or routed to the servlet engine for handling there.

There are two configuration files that you must set up to get Apache and `mod_ose` running properly: `httpds.conf` and `ose.conf`.

httpds.conf

The main Apache configuration file used by the Oracle HTTP Server is `httpds.conf`. Where `$APACHE_HOME` is the Oracle HTTP Server home directory, this file is located in

`$APACHE_HOME/Apache/conf/`

> **Note:** `$APACHE_HOME` on UNIX systems is normally `$ORACLE_HOME/Apache`. `%APACHE_HOME%` on Windows NT is `%ORACLE_HOME%\Apache`.

This is the only file that Apache reads directly when it starts up. Other configuration files are read when they are included in `httpds.conf`, using the `include` directive.

> **Note:** The Apache configuration file that is used by the Oracle HTTP Server is **httpds.conf**, not `httpd.conf`.

Configuring Apache and `mod_ose` is not dynamic. When you change a configuration file, you must restart Apache. However under Solaris there is a way to restart Apache on the fly, so that the configuration files are read but active connections are still maintained. Use the OS command

`% $APACHE_HOME/bin/httpdsctl graceful`

In the `httpds.conf` file, you need to modify or verify the following for `mod_ose` support.

- The port that Apache is listening on for standard HTTP requests. This is the port number that the HTTP client must use to connect to Apache.

- If SSL is used, the port for SSL requests.

- Add an Apache `Include` directive that directly or indirectly includes the mod_ose configuration file `ose.conf`.

ose.conf

The configuration file for `mod_ose` is `ose.conf`. It is located in `$APACHE_HOME/modose/conf/`. This file determines which Oracle listener connections are supported, and how requests to OSE are routed through the Oracle listener to the appropriate Web service, servlet context, and servlet.

The `ose.conf` file contains standard Apache directives, such as `IfModule`, `LoadModule`, and `Location`. The structure of `ose.conf` is the following:

LoadModule	The `LoadModule` directive is the first in the file. It specifies a module name, and the location of the dynamic library that implements the module. For an example, see "Example ose.conf" on page 5-7.
IfModule	The IfModule directive includes the directives following if the module source has been compiled into Apache. The directive is `<IfModule mod_ose.c>`.
Location	The `Location` directive specifies URI components, and which handlers serve them. `<Location /ose/*>` specifies that any URI-path starting with `/ose/` should be handled by the `SetHandler` directive that follows. For example:

```
<Location /ose/>
  SetHandler aurora-server
</Location>
```

is a complete `Location` directive. |
| /IfModule | Terminates the `IfModule` directives. |

The `mod_ose`-specific directives are:

`AuroraService`	Mandatory directive that specifies the connect identifier (in the listener initialization file `tnsnames.ora`) for all HTTP requests, except those that contain a URL-path that is specified in an `AuroraLocationService` directive.
	This directive cannot be specified within a `Location` directive. It must be specified in the `IfModule` directive.
	Use the `-netservice` option of the OSE session shell `exportwebdomain` command to generate this directive, or specify the *netservice* key in the input data file for the `gencfg.pl` Perl script.
`AuroraWorkers PerProcess`	Optional directive that indicates the number of `mod_ose` worker threads that can be created per Apache/`mod_ose` process.
	This directive is not meaningful for current Solaris releases of the Oracle HTTP Server, as Solaris Apache is single-threaded. On Windows NT, each Apache process can execute more than one client request at a time.
	This value should be the same as the `ThreadsPerChild` value that is specified in the Apache configuration file. This applies only to Windows NT or Windows 2000 installations.
	Use the `-worker` option of the OSE session shell `exportwebdomain` command to generate this directive, or specify the *worker* key in the input data file for the `gencfg.pl` Perl script.
`AuroraLocation Service`	Optional directive included in the `Location` directive that indicates the TNS connect identifier to use for that location. Should not be used if the connect identifier is the same as the global connect identifier that is specified in the `AuroraService` directive.

These `mod_ose`-specific directives are generated in the configuration file using the session shell `exportwebdomain` command, or using the `gencfg.pl` Perl program.

Example ose.conf

Here is an example of an ose.conf file:

```
LoadModule ose_module     /private1/Apache/modose/bin/libjipa9i.so

#
# Apache configuration
# Domain: /system/admin
# Context: admin
#
<IfModule mod_ose.c>

AuroraService inst1_modosetest

AuroraWorkersPerProcess 1

#
# Context for VPATH /admin/
#

<Location /admin/deploywar >
SetHandler aurora-server
</Location>

<Location /admin/error_log_viewer.htm >
SetHandler aurora-server
</Location>

<Location /admin/errors/internal >
SetHandler aurora-server
</Location>

<Location /admin/event_log_viewer.htm >
SetHandler aurora-server
</Location>

<Location /admin/http_log_viewer.htm >
SetHandler aurora-server
</Location>

<Location /admin/shell >
SetHandler aurora-server
</Location>

<Location /ose/ >
```

```
SetHandler aurora-server
</Location>

<Location /test/ >
SetHandler aurora-server
</Location>

</IfModule>
#
# End of configuration
#
```

oracle_apache.conf

Contains includes for Oracle-specific modules, such as mod_plsql and mod_ose. Include the ose.conf file in this file, using an absolute file path, then make sure that the Apache main configuration file httpds.conf includes oracle_apache.conf.

Including Configuration Files in httpds.conf

You can include the mod_ose configuration file ose.conf in the Apache configuration file httpds.conf directly, using the Apache Include directive. However, Oracle recommends that you add the Include for ose.conf to the configuration file oracle_apache.conf, which also includes the configuration files for other Oracle-specific Apache modules such as mod_plsql ($APACHE_HOME/modplsql/conf/plsql.conf). The oracle_apache.conf file is located in $APACHE_HOME/Apache/conf/. Then include oracle_apache.conf in the httpds.conf file.

Always specify the files to be included using absolute paths to the files, as Apache cannot recognize relative paths. Also, do not use operating system environment variables, such as the UNIX shell variable $APACHE_HOME, or the Windows NT environment variable %APACHE_HOME% in the Include directive.

Oracle Net and Oracle Listener Configuration

Connections from Apache/mod_ose to the OSE use the Oracle Net protocol, the same as OCI or Forms client/server applications. mod_ose uses the same mechanism for finding connection descriptors as other Oracle clients, such as OCI clients. Depending on the configuration determined by the sqlnet.ora file, connections are defined either by

- the tnsnames.ora file located in the directory pointed to by the environment variable TNS_ADMIN, **or**

- another Oracle name service, such as an LDAP service, to retrieve the connection address

See the *Oracle Net Administration Guide* for more information about configuration of listeners and dispatchers for the Oracle server.

So to use mod_ose with the OSE/OJVM you must at least configure the Oracle listener's initialization file: tnsnames.ora, to include connect descriptors for the connections to the OSE server. You can find specific information about doing this in "tnsnames.ora" on page 5-10.

Generating a Configuration File

The `ose.conf` file contains Apache configuration commands and directives, such as `LoadModule`, `<IfModule>`, and `<Location>`. To create a configuration file, you could analyze your application, find the servlet contexts that `mod_ose` must recognize and transfer HTTP requests to, and create the `mod_ose` configuration file by hand, using a text editor. Or you could modify the example files that are supplied with Oracle.

These approaches, however, are very error-prone, and are not recommended.

Oracle recommends that you create a `ose.conf` file using the tools supplied to generate this file. With Oracle8*i*, you used the `exportwebdomain` session shell command to create the `mod_ose` configuration for specific Web applications. With Oracle9*i*, you can still use the `exportwebdomain` command, or you can use the new Perl script `gencfg.pl`. This script takes a descriptor file as input, and generates or modifies a `mod_ose` configuration file. This tool is described in "gencfg.pl" on page 5-14.

tnsnames.ora

This Oracle Net initialization file specifies the connect descriptors that are used by the Oracle listener. An example of this file is provided with all Oracle server installations, as well as with Oracle client-side installations. For a complete description of the entries in this file, see the *Oracle Net Administrator's Guide*. In some cases, you might have to copy the file from an example location before modifying it. For example, in an NT client configuration, copy the file from `%ORACLE_HOME%\network\Admin\Sample` up to `%ORACLE_HOME%\network\Admin`, and then modify it to add the `mod_ose`-specific connect identifiers.

Modify this file to add one or more connect identifiers that point to OSE services. Here is an example of a possible entry:

```
inst1_http =
  (DESCRIPTION=
    (ADDRESS=
      (PROTOCOL=tcp)(HOST=sales-server)(PORT=1521)
    )
    (CONNECT_DATA=
      (SERVICE_NAME=sales.us.acme.com)
      (SERVER=shared)
      (PRESENTATION=http://sales)
    )
```

```
)
```

In this example, the connect identifier `inst1_http` is associated with a connect descriptor that specifies the TCP/IP protocol, a host named `sales-server`, and a service that listens on the Net8 port 1521. A service name, `sales.us.acme.com` is specified, and the presentation is HTTP. The service name is the instance name in the INIT.ORA file (there can be multiple instance names in the case of Oracle parallel server installations. You create the service (the presentation) using the `createwebservice` session shell command.

connect identifier The connect identifier `inst1_http` is an arbitrary name. This connect identifier is generated into the `ose.conf` file, in the `AuroraService` directive, by the configuration utilities. For an example, see the section "gencfg.pl" on page 5-14.

address The address part of the connect descriptor specifies

- the network protocol, which must be TCP or TCPS for `mod_ose`

- the host (the server) that `mod_ose` should connect to

- the TCP port (socket number) that the HTTP service is registered on—see the `addendpoint` command, described in "addendpoint" on page 3-6 and in the *Oracle9i Java Tools Reference*.

connect_data This part of the entry provides a service name, the kind of server (shared or dedicated), and the presentation. The presentation is the HTTP service that you established for your application.

Non-Shared Server Installations

The `mod_ose` module requires a shared server installation to run. It will not run if the database is not configured with any shared dispatchers/servers. However, some users prefer not to run with shared servers as a standard configuration. Oracle recommends the following work around to let you use Apache and `mod_ose` in a dedicated server environment.

In the database initialization file (`INIT.ORA`), create one or more dedicated dispatchers and servers with a specific service name. You can use any service name that follows the `INIT.ORA` conventions as long as it differs from the database service name. The following examples use "MODOSE":

```
dispatchers="(PROTOCOL=tcp)(SERVICE_NAME=MODOSE)"
dispatchers="(PROTOCOL=tcps)(SERVICE_NAME=MODOSE)"
```

The connect identifier in the tnsnames entry (`tnsnames.ora` file) should use this service name rather than the database service name.

```
inst2_http =
  (DESCRIPTION=
    (ADDRESS=(PROTOCOL=tcp)(HOST=server27)(PORT=5521))
      (CONNECT_DATA=
        (SERVICE_NAME=MODOSE)
        (SERVER=shared)
        (PRESENTATION=http://admin)
      )
  )
```

You will now have a shared dispatcher and server that will not be used by other clients—only by `mod_ose`.

Configuration Utilities

The Oracle9*i* servers support the two utilities for generating configuration files for `mod_ose`. They are described in the following sections.

exportwebdomain

The configuration of Web applications running on OSE is specified in the JNDI namespace on the OSE server. The session shell command `exportwebdomain` extracts the information about the applications installed in a Web domain and generates the corresponding configuration file.

Use the `exportwebdomain` command to generate the structure of a Web domain in a configuration file for mod_ose. The export utility works in two stages:

- Generates in XML format the structure of a Web domain or contexts within a domain.

- Applies transformations to the XML, producing configuration files specific to `mod_ose`. Generates a configuration file at the shell level.

This session shell tool can be used to generate the `mod_ose` configuration file. `exportwebdomain` takes as an argument the name of the Web domain for which you want to generate a configuration. For example

```
exportwebdomain [options] /HRRoot
```

generates a configuration file for the Web domain `HRRoot`, which is located at the root of the JNDI namespace in the OSE server. The options to `exportwebdomain` are the following:

-context	Optional parameter that specifies the name of the servlet context to support. If not specified, all contexts in the domain are configured.
-netservice	The name of the service defined in the `tnsnames.ora` file.
-format	The XSLT transformation type. For example `-format Apache` for `mod_ose` configurations.
-worker	Optional parameter that specifies the number of worker threads per Apache process. In the release, this option applies only to NT installations, since the current release of Solaris Apache is always single-threaded.

-nodefault	Optional parameter that indicates not to map the default context, unless indicated by the `-context` option.
-nodocs	Optional parameter that specifies to not forward URLs mapped into `doc_root` to the Servlet engine. This assumes that such static pages will be served directly by the external Web server.

gencfg.pl

gencfg.pl is a Perl program that generates or modifies configuration files for mod_ose based on an input description file. The description file contains key:value pairs in a text format. The output is generated in the file ose.conf in the same directory. If the file already exists, new material is added to it.

The Perl script provides for two modes of configuration:

- simple: uses the existing ose.conf file in the same directory as gencfg.pl and updates the AuroraService and AuroraWorkersPerProcess directives if they already exist.

- exportwebdomain: uses the session shell exportwebdomain command. The arguments to the exportwebdomain command are constructed by the Perl script, based on the data either in the input description file webdata.cfg, or in another file specified in an argument on the command line. The session shell is then executed by the Perl program. The OSE database must be running when you use this mode.

The keys and possible values in webdata.cfg are:

type	Optional key. If the value is *simple*, use the simple mode. If this key is not provided, use the *exportwebdomain* mode.
netservice	Mandatory key. The Oracle Net connect name to be used in generating the AuroraService directive in the output configuration file. See "ose.conf" on page 5-5 for more about this directive.
worker	Optional key. The number of worker threads to be used for each Apache process. Used only for NT Apache in this release. The default is 1 when not specified.
domains	Web domain for which the configuration is to be generated. Mandatory if *type* is *exportwebdomain*.

context   Optional key. The servlet context to be used for Web domain.
If not provided, configuration is generated for all the contexts
in that domain.

alias   The database to which the session shell should connect.
Optional—if not supplied, uses the local database

nodocs   Optional key. If true, do not generate configuration for
documents.

nodefault   Optional key. If true, do not generate configuration for default
servlets.

An example of key:value pairs in a description file (`webdata.cfg`) is

```
netservice:inst1_http
nodocs:true
nodefault:true
workers:1
context:admin
domains:/system/admin
```

When `gencfg.pl` is executed with this input file, it generates the following
`ose.conf` configuration file:

```
#
# Apache configuration
# Domain: /system/admin
# Context: admin
#
<IfModule mod_ose.c>

AuroraService inst1_http
AuroraWorkersPerProcess 1
#
# Context for VPATH /admin/
#

<Location /admin/deploywar >
SetHandler aurora-server
</Location>

<Location /admin/error_log_viewer.htm >
SetHandler aurora-server
</Location>
```

```
<Location /admin/errors/internal >
SetHandler aurora-server
</Location>

<Location /admin/event_log_viewer.htm >
SetHandler aurora-server
</Location>
```

AuroraLocationService

Use this directive in the following way:

- In the `config` object for the servlet context (in the OSE server's JNDI namespace), add a group entry called `context.properties`.

- Set the `context.locationservice` property in this group to the connect identifier for the desired service, as specified in the `tnsnames.ora` listener initialization file. Use the session shell `addgroupentry` command to do this. For example, in the session shell, do

```
$ cd /system/admin/contexts/default
$ addgroupentry config context.properties context.locationservice
            inst2_https
$ cd /HRRoot/contexts/HRService
$ addgroupentry config context.properties context.locationservice inst3_http
```

When you execute the `exportwebdomain` session shell command to create a `mod_ose` configuration file, either directly or by using the `gencfg.pl` Perl program, the `Location` directives for the configured locations will contain the `AuroraLocationService` directive. For example

```
<Location /system/admin/contexts/default/ >
  AuroraLocationService inst2_https
  SetHandler aurora-server
</Location>

<Location /HRRoot/contexts/HRService/ >
  AuroraLocationService inst3_http
  SetHandler aurora-server
</Location>
```

> **Note:** The locations specified in the `AuroraLocationService` directives override the location specified in the `AuroraService` directive of the `IfModule` clause.

> **Note:** Even if you specify a stateless handler
> (aurora-stateless-server) in the SetHandler clause when
> specifying AuroraLocationService, the semantics are those of
> the stateful handler. That means that the connection is closed either
> after the request completes (when KeepAlive is OFF), or when the
> client completes (when KeepAlive is ON).

Specifying Stateful and Stateless Handlers in ose.conf

Specify whether a request uses the stateful or the stateless connection by indicating which Apache *handler* to use. You indicate the handlers in Location directives in the ose.conf configuration file. For example:

- SetHandler aurora-stateless-server

  This handler specifies a stateless connection. If a servlet being served by a stateless connection attempts to create an HttpSession object, it is considered an error.

- SetHandler aurora-statefull-server

  This handler specifies a stateful connection. It allows a servlet to create an HTTPSession object, and requires Apache to define a separate session to service the requests.

- SetHandler aurora-server

  This specifies the default mode, which is stateful.

SSL Configuration

mod_ose can accommodate SSL connections between Apache and the OSE/OJVM, to support SSL connections between the client and OSE/OJVM. You need the following to set up SSL service:

- a current and correct wallet

- a TCPS connect identifier, specified in the tnsnames.ora configuration file

- the Aurora service referenced to a connect identifier in tnsnames.ora, with the TCPS protocol and a TCPS listener port. For example:

```
inst1_https  = (DESCRIPTION=
                  (ADDRESS=(PROTOCOL=tcps)(HOST=server27)(PORT=5524))
                  (CONNECT_DATA=
                     (SERVICE_NAME=rdbms817.rdbms.dev.us.oracle.com)
                     (SERVER=shared)
                     (PRESENTATION=http://admin)
                  )
                )
```

Oracle Net uses the service name rather than the SID that was used in earlier versions of the Oracle server.

listenerAddress	An ADDRESS_LIST (host, port, protocol) in the case of multiple Oracle Listeners on the back end (multiple nodes). This allows for load balancing and fail-over configurations, as well as the use of CMAN for connection concentration.
	Read the *Oracle Net Administrator's Guide* for information about how to set this parameter.
serviceSpec	*group of database instances:* specify SERVICE_NAME Defines a group of instances that can be used interchangeably. When there are multiple database instances, mod_ose load balances the connections between the different instances.
	mod_ose guarantees stateful requests from a client are sent to the same database instance so they can be associated with the same database session.
	single database instance: specify INSTANCE_NAME Indicates an specific instance within the service group should be used.
presentationSpec	Indicates which HTTP service should be used for this connection.
	<JndiServiceName> is a place holder for the name of a service in the JNDI namespace (for example, /service/ServiceName) that understands HTTP and has an Oracle Net end-point associated with it. See the session shell command, createwebservice.

Configuring mod_osso

See "Using mod_osso with mod_ose" on page 4-14 for basic information about mod_osso. This section tells you how to configure mod_osso to work with the Oracle HTTP Server and mod_ose.

> **Note:** mod_osso is not available with Oracle8*i* Release 3.

To Configure on the Apache Side

Before chaining mod_osso with mod_ose, you must register the Oracle HTTP Server with an Oracle Single Sign-On server. See the documentation on the Oracle HTTP Server for more information.

To use mod_osso, just protect a virtual path with mod_osso. To do this, define a Location directive in the Apache configuration file httpds.conf as follows:

```
<Location /<context_virtual_path>/<servlet_virtual_path/>
    AuthName "OSSO Server on machine1"
    AuthType Basic
    require valid-user
</Location>
```

Enter this Location directive in the <IfModule mod_osso.c> section of the Apache httpds.conf configuration file. For more information about this, including other required options, see the Apache documentation, and the *Oracle HTTP Server Administration Guide* (available with Oracle9*i* Internet Application Server V2).

Next, configure a virtual path to be forwarded to the OSE/OJVM. For example, add the following Location directive to the ose.conf configuration file:

```
<Location /<context_virtual_path>/<servlet_virtual_path/>
    SetHandler aurora-server
</Location>
```

Enter this in the <IfModule mod_ose.c> section of ose.conf, and use the same virtual servlet context and virtual servlet path as in the Location directive in the mod_osso.c section of httpds.conf.

To Configure on the OSE Server Side

Secure the virtual path in the OSE/OJVM with the OSSO authentication method by publishing a special OSSO realm for each Web service in the OSE/OJVM. If you

plan to service requests which have been authenticated by an OSSO server, you must have an OSSO HTTP security realm published in each service root that is accessed by such a request.

Publish an OSSO Realm

Publish the special OSSO realm by using the session shell `realm publish` command with the `-type OSSO` option, as shown in the following example:

```
$ realm publish -w[ebservice] <serviceRoot> -type OSSO
```

It is not necessary to specify the realm name in the `-add` parameter. The realm with authentication type OSSO will always be called `ossoRealm`. In fact, any other name you specify in the `-add` parameter is ignored.

> **Note:** There can be at most one realm with the authentication type OSSO for each service root.

To Remove an OSSO Realm

You can remove an OSSO realm just as any other realm. Use the session shell `realm publish` command with the `-remove` option, as shown in this example:

```
$ realm publish -w[ebservice] <webserviceRoot> -remove ossoRealm
```

Securing a Servlet Context with the OSSO Security Servlet

You can configure OSSO authentication at the level of servlet context. As with regular HTTP security, you use the session shell `realm secure` command, as shown in this example:

```
$ realm secure -s /testRoot/contexts/mycontext -osso
```

Note the special `-osso` flag. It tells the realm command that OSSO authentication should be enabled for this servlet context. The `realm secure` command binds a JNDI object called `httpSecurity` into the /testRoot/contexts/mycontext. You can use the session shell `ls` command to verify the existence of the `httpSecurity` servlet.

A standard `realm secure` command, without the `-osso` option, also binds an `httpSecurity` object into the context. The difference is in the class bound to `httpSecurity`. For a regular HTTP Security method, `httpSecurity` is bound to a servlet of class `oracle.aurora.mts.http.security.HttpSecurity`. In the case of OSSO authentication it is bound to `oracle.aurora.mts.http.security.OSSOSecurity`. You can use the session shell `getproperties` command to verify the class binding, as shown in

```
$ getproperties /testRoot/contexts/mycontext/httpSecurity
```

This returns:

```
servlet.class=SYS:oracle.aurora.mts.http.security.OSSOSecurity
```

Servlet contexts are secured with the HTTP Security method *by default* when they are created. So if you need OSSO authentication, you must explicitly use the `realm secure` command with the `-osso` option to change the type of authentication to OSSO.

> **Note:** For the `realm secure -osso` command to succeed, you must have already created the special OSSO realm in the service root, as described in "Publish an OSSO Realm" on page 5-23.

It is also possible to configure a servlet context for OSSO authentication when creating the servlet context using a WAR file. For more information about this, see "Authenticating with Oracle Single Sign-On" on page 8-14.

Troubleshooting

If you have trouble connecting through mod_ose to your servlets or JSPs running in the OSE/OJVM server, you can take the following steps to find the source of the problem.

1. Make sure that the servlet has been installed correctly on the OSE/OJVM server. Follow the troubleshooting steps in Chapter 3, "OSE Configuration and Examples". See [[need ref in that chapter]] for specific steps.

2. Make sure that the Oracle listener is running, and has been configured properly. Be certain that the AuroraService directive in the ose.conf file specifies the correct entry in the tnsnames.ora file, and that this entry has the correct service name, service type, protocol type, and port number. See "tnsnames.ora" on page 5-10 for more information.

You can also use the tnsping utility to test the tnsnames.ora entry.

3. Try stopping and restarting the Oracle listener, using the lsnrctl command.

4. Use the lsnrctl status command to view the status of each listener port.

5. Is the Apache server running? If it is, an OS process status command (such as ps -e under UNIX, or the TaskManager under Windows NT), should show one or more httpds processes running.

6. Try stopping and restarting the Apache server, to make sure that your latest httpds.conf and ose.conf files have been read by Apache.

7. If the connection is refused by the server:
   - check that the proper port was being used for the request
   - make sure that the correct host name is being used for the request

8. If an HTTP 404 error occurs:
   - check the URL contains a virtual path string that specifies the correct servlet context
   - make sure that the URL also contains a virtual path string that specifies the correct servlet within the servlet context
   - disable, as much as possible, caching in any Web browsers being used to test HTTP content
   - if the servlet was just published, or configuration information just changed, wait for any active sessions to timeout or close all current browsers and start a completely new browser

- make sure the `doc_root` specified for that service and context is valid

6

Calling EJBs

This chapter tells you how to call Enterprise JavaBeans (EJBs) from servlets running in the OSE/OJVM servlet container.

This chapter covers the following topics:

- Overview
- EJB Example

Overview

When you call a server-side EJB from a client application, you must use a network protocol that involves an ORB, such as RMI over IIOP. However, calling out from the servlet to an external object that is in the same session as the servlet can be much simpler and faster than calling from a client. When the servlet and the other object are running in the same server session, the connection between them does not involve network traffic, only in-memory resource-sharing. Also, no ORB is involved, making the calling code in the servlet code simpler. All you need to do is specify the name of the object to look up.

The remainder of this chapter documents an example that demonstrates calling an EJB from a servlet in the same Oracle Java Virtual Machine session.

EJB Example

In this example, the EJB accesses the Oracle database to retrieve the employee ID number for an employee whose last name was passed to the EJB from the servlet. So the example comprises four elements:

- a client, such as a Web browser, that invokes the servlet
- the servlet
- the EJB, that is called by the servlet
- the Oracle database, from which the EJB retrieves an employee ID number

The example is kept simple, so that you can easily see how the four elements interact.

In order to compile and run the example, the Oracle9*i* server must be installed and running, and the HR example schema must be installed in the database. (The HR schema is preconfigured.)

> **Note:** The HR schema is not available in the Oracle8*i* Release 3 distribution. But this example would work for Oracle8*i* (Release 3) if you change the SQL statement in the EJB code to access the SCOTT.EMP table, and change the Makefile or batch file to access the SCOTT schema, rather than the HR schema.

There are two main pieces of code that make up the example:

- `IDServlet.java`—the servlet code
- `IDServer/IDBean.java`—the EJB code

In addition to these, there are the EJB Home and Remote Interface specifications:

- `IDCommon/ID.java`—the remote interface
- `IDCommon/IDHome.java`—the home interface

Finally, there are several support files required for any application that uses EJBs:

- `ID.xml`—the EJB descriptor
- `IDMap.xml`—the Oracle-specific EJB descriptor

Servlet

The servlet is called from a client, such as Web browser. The client must pass the employee last name to the servlet in a query string. See "Accessing the Servlet" on page 6-10 for an example of the kind of URL that you could use to invoke the servlet.

The servlet code is listed in this section. The servlet performs the following steps:

1. Imports the required Java packages, including the EJB home and remote interfaces.

2. Declares Java variables for the last name, and for the ID number that is returned by the bean.

3. Gets the last name from the query string.

4. Gets a new Initial Context for calling the EJB.

5. Looks up the EJB home interface, using the name of the bean as published in the JNDI namespace ("/test/IDBean", see "Compiling and Deploying the Example" on page 6-9).

6. Creates a reference to the remote interface.

7. Calls the EJB getID(String) method to retrieve the employee's ID number, passing it the last name obtained from the query string.

8. Prints the ID number to the HTTP response output stream.

The listing for the code, IDServlet.java, is shown here:

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import IDCommon.ID;
import IDCommon.IDHome;
import javax.naming.Context;
import javax.naming.InitialContext;

public class IDServlet extends HttpServlet
{
  public void doGet (HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException
  {
    String EmpID = null;
    String LastName = null;
    PrintWriter out = new PrintWriter(resp.getOutputStream());
```

```
        LastName = req.getQueryString();

        try {
          InitialContext ic = new InitialContext();
          IDHome ID_home = (IDHome)ic.lookup ("/test/IDBean");
          ID id = ID_home.create ();
          EmpID = id.getID(LastName);
          out.println (LastName + "'s employee ID is " + EmpID);
        } catch (Exception e) {
          EmpID = "Error";
        }
        out.flush(); out.close();
    }


    public void init(ServletConfig cfg) throws ServletException {
      super.init(cfg);
    }


    public void destroy() {
      super.destroy();
    }

    public String getServletInfo() {
        return "Servlet calling EJB example";
    }
}
```

EJB

The Enterprise JavaBean that is called by the servlet takes the employee's last name that is passed from the servlet, and looks up the ID number in the database. The EJB performs the following steps:

1. Imports the required packages. Note that the SQL and the Oracle JDBC packages must be imported for the JDBC code in the example to work (or even compile).

2. The bean implements a method `getID(String LastName)` to get the ID number from the database server.

3. The bean method declares a JDBC connection `conn`, a JDBC result set `rset`, and a `String EmpID` for the ID. (The ID number could have been retrieved into

an `int`, but the `String` is used as a "quick and dirty" way of returning a SQL error if the JDBC connection or query fails.)

4. The first two Java statements in the `try{}` block open an Oracle server-side internal JDBC connection.

5. The `if{}` block sets up a prepared JDBC statement. Note the '?' placeholder for the last name in the WHERE clause of the SQL statement.

6. `pstmt.setString(1, LastName)` sets the value of the argument into the SQL statement.

7. The query is executed, and the next value for the query is obtained. Note that if there is more than one person with the same last name in the table, only the first person's ID is retrieved.

8. The `getString()` method on the result set object gets the ID value.

9. If the JDBC-related statements fail, the `catch{}` block is executed. The text string for the SQL exception is placed in the return value `EmpID`. Oracle does **not** recommend this technique as a standard way for applications to handle JDBC errors. It is done here to keep the example simple. See the Oracle9*i JDBC Developer's Guide and Reference* for better ways to handle errors of this kind.

10. Finally, the ID number is returned as a string to the EJB caller—the servlet.

EJB Code

The code for `IDBean.java` is shown here:

```
package IDServer;

import javax.ejb.SessionBean;
import javax.ejb.CreateException;
import javax.ejb.SessionContext;
import java.rmi.RemoteException;
import java.sql.*;
import oracle.jdbc.*;

public class IDBean implements SessionBean
{
  public String getID (String LastName) throws RemoteException {
    Connection conn = null;
    ResultSet rset = null;
    String EmpID = null;
    try {
      OracleDriver ora = new OracleDriver();
```

```
      conn = ora.defaultConnection();
      if (conn != null) {
        PreparedStatement pstmt = conn.prepareStatement
          ("select employee_id from hr.employees where last_name = ?");
        pstmt.setString(1, LastName);
        rset = pstmt.executeQuery();
        if (rset.next()) {
          EmpID = rset.getString(1);
        else
          EmpID = "Unknown";
        }
      }
    } catch (java.sql.SQLException e) {
      EmpID = e.getMessage();
    }
    return EmpID;
  }

  // Methods of the SessionBean

  public void ejbCreate () throws RemoteException, CreateException {}
  public void ejbRemove() {}
  public void setSessionContext (SessionContext ctx) {}
  public void ejbActivate () {}
  public void ejbPassivate () {}

}
```

EJB Interfaces

The EJB home interface (IDCommon/IDHome.java) specification is:

```
package IDCommon;

import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import java.rmi.RemoteException;

public interface IDHome extends EJBHome
{
  public ID create () throws RemoteException, CreateException;
}
```

The remote interface (IDCommon/ID.java) is:

```
package IDCommon;
```

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface ID extends EJBObject
{
  public String getID (String LastName) throws RemoteException;
}
```

EJB Descriptors

The EJB descriptor for this example is:

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems Inc.//DTD Enterprise JavaBeans 1.1
//EN" "ejb-jar.dtd">
<ejb-jar>
 <enterprise-beans>
   <session>
   <ejb-name>IDBean</ejb-name>
   <home>IDCommon.IDHome</home>
   <remote>IDCommon.ID</remote>
   <ejb-class>IDServer.IDBean</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
   </session>
 </enterprise-beans>
 <assembly-descriptor>
   <security-role>
     <role-name>OraclePublicRole</role-name>
   </security-role>
   <method-permission>
     <role-name>OraclePublicRole</role-name>
     <method>
       <ejb-name>IDBean</ejb-name>
       <method-name>*</method-name>
     </method>
   </method-permission>
   <container-transaction>
     <method>
      <ejb-name>IDBean</ejb-name>
      <method-name>*</method-name>
     </method>
     <trans-attribute>Required</trans-attribute>
   </container-transaction>
```

```
 </assembly-descriptor>
</ejb-jar>
```

The Oracle-specific descriptor is:

```
<?xml version="1.0"?>
<!DOCTYPE oracle-ejb-jar PUBLIC "-//Sun Microsystems Inc.//DTD Enterprise JavaBe
ans 1.1//EN" "oracle-ejb-jar.dtd">
<oracle-ejb-jar>
 <oracle-descriptor>
   <ejb-name>IDBean</ejb-name>
   <mappings>
     <ejb-mapping>
       <ejb-name>IDBean</ejb-name>
       <jndi-name>test/IDBean</jndi-name>
     </ejb-mapping>
     <security-role-mapping>
        <security-role>
          <role-name>OraclePublicRole</role-name>
        </security-role>
        <oracle-role>PUBLIC</oracle-role>
     </security-role-mapping>
     <transaction-manager>
       <default-enlist>True</default-enlist>
     </transaction-manager>
   </mappings>
 </oracle-descriptor>
</oracle-ejb-jar>
```

Compiling and Deploying the Example

Follow these steps to compile and deploy the example:

1. Use a JDK 1.2-compliant Java compiler to compile

    - `ID.java`

    - `IDHome.java`

    - `IDBean.java`

    - `IDServlet.java`

2. Use `loadjava` to load the `IDServlet.class` file into the Oracle server.

3. Publish the servlet to the JNDI namespace. For the URL specified below, we assume that the servlet has been published to the `named_servlets` directory

in the HR servlet context (`/HRRoot/contexts/HRContext`). See Chapter 3, "OSE Configuration and Examples" for more information about `loadjava` and `publishservlet`.

4. Deploy the EJB to the server using the `deployejb` command. Here is a possible `deployejb` command for this example:

```
% deployejb -republish -addclasspath .:<your_classpath> -temp temp \
   -user HR -password hr
   -service sess_iiop://<host_name>:<your_port_number>:<your_Oracle_SID>
   -descriptor ID.xml -oracledescriptor IDMap.xml    server.jar
```

There is a Makefile, and Windows NT batch files, that show how to compile this demo, in the demo samples directory of your Oracle installation.

Accessing the Servlet

You can access the servlet from a Web browser, for example Netscape Navigator or Internet Explorer. To invoke the servlet, specify a URL of the general form:

```
http://<your_host_name>:<port_number>/
<context_virtual_path>/<servlet_virtual_path>?<employee_last_name>
```

(But all on one line.) Here is a specific URL that was used in testing this example:

```
http://dlsun1497:8060/ose/testID?Kumar
```

Where `dlsun1497` is the author's workstation (inside the Oracle firewall, sorry), `8060` is the port number that the HTTP service listens on, `/ose` is the virtual path for the `HRContext` servlet context (see Chapter 3, "OSE Configuration and Examples"), and `/IDtest` is the virtual path that `IDServlet` was published under in step 3 above.

If you do not specify a query string in the URL, the Web client prints:

```
null's employee ID is Unknown
```

If you specify an employee name that does not exist in the `HR.EMPLOYEES` table, you get:

```
xyz's employee ID is Unknown
```

(You can see why returning the error in the String is not a very good idea for a real application.)

7

Oracle Servlet Engine Security

This chapter covers basic aspects of security for the Oracle Servlet Engine, running in the Oracle Java Virtual Machine. This chapter focuses on security as established using the session shell tool. Chapter 8, "Oracle WAR Deployment" describes those aspects of security that are implemented using Web Archive (WAR) deployment files.

The topics discussed in this chapter are:

- Overview
- JNDI Security
- HTTP Security
- Examples
- Troubleshooting

Overview

Security for the OSE/OJVM includes three security mechanisms:

1. Oracle9*i* server security, which involves database schemas and roles.

2. The JNDI protection mechanism, that is based on Oracle database security.

3. The HTTP security mechanisms, involving *realms*, *groups*, and *principals*, and the access permissions associated with principals.

The first two aspects of OSE/OJVM security and the third are virtually orthogonal. The slight exception to their almost total independence comes when an HTTP security realm type uses database schemas as the principals. This is described in "The DBUSER Type" on page 7-7.

This chapter first describes the JNDI protection mechanism that is based on Oracle database security, then describes how HTTP security is implemented by the OSE/OJVM. Although knowledge about basic database security and Java security in the database is helpful in reading this chapter, it is not essential. If you need more information on these topics see the following Oracle guides:

■   Oracle9*i Application Developer's Guide*

■   *Oracle9i Java Developer's Guide*

This Guide assumes some basic knowledge about HTTP security. For more information, you can look up the HTTP 1.1 specification RFC, which is available at

```
http://www.w3.org/Protocols/rfc2068/rfc2068
```

More accessible documentation about HTTP security is available in any of the trade press books that cover the Apache Web server. Two such are

■   *Professional Apache*, by Peter Wainwright (Wrox)

■   *Apache, the Definitive Guide*, by Ben Laurie and Peter Laurie (O'Reilly)

JNDI Security

To create entries in the OSE/OJVM JNDI namespace, you must be in possession of a valid Oracle database schema name and password. Using the session shell tool requires a connection to a database session, which requires a database server login. For example, to create a Web service you must connect to the OSE session shell as the database schema SYS. If SYS then changes ownership of the service root to schema HR, then you have to be connected through the session shell as HR to do things like publish servlets, create HTTP security objects such as realms, add principals to realms, and so on.

When a client accesses the OSE, from a Web browser for example, and runs a servlet that is owned by a schema, then that servlet runs with all the database privileges associated with that schema. The servlet can query database tables or other database objects, update tables and other objects, run stored procedures, and do all the other things that a database user can do, *exactly in accord with the database permissions that the schema possesses*.

What HTTP security allows you to do is permit and restrict access to servlets and other JNDI objects (JSPs, text files, and so on) above and beyond the database access permissions. For example, you might not want all Web browser users to be able to access servlets that in turn access the HR database schema. So you can add authentication and authorization requirements to the HR servlets using HTTP security, which is described in "HTTP Security" on page 7-5.

JNDI Security Implementation

JNDI security is implemented using a UNIX-like permissions scheme. Each OSE JNDI object, for example a published servlet in a `named_servlets` context, has an owner. The owner has a combination of three types of permission: *read*, *write*, and *execute*. The exact semantics of each permission type are described in the *Oracle9i Java Tools Reference*. In summary, read permission allows the session shell user to "read" the object: list it, get its properties, and actually read it to the extent that it is readable. Write permission allows the user to modify the object: delete it, substitute another object, write it to the extent it is writable (a text file, for example), add objects or properties to it (if it is a context, or a realm). Execute permission allows the user to have the OSE activate the object, if it is for example a servlet, or to search the object, if it is a context.

In addition to owner's permissions, each OSE JNDI object has an access control list. (This is where JNDI permissions differ somewhat from UNIX permissions, as the UNIX group concept is not directly implemented in the OSE JNDI namespace

implementation.) So an object can have a list of database users (schemas), each of which can have a different set of access permissions.

Servlet Permissions

Normally, OSE JNDI objects in a servlet context inherit the ownership and permissions of the owner of the Web domain. However in many cases it is desirable for the owner of a domain to grant the right for other schemas to publish servlet contexts in that domain, and to then effectively publish servlets in those contexts. For example, the HR schema owns a domain HRRoot. HR can grant the schema BENEFITS the right to publish servlets in its own context in that domain.

Run As Owner

In the normal course of events, the OSE would use the domain owner's database permissions when executing servlets in a servlet context in the domain. However, the domain owner can establish that a servlet context is to run with the database permissions of the servlet context owner. This is set by adding a group property to the domain config object, as the follow example, for the HRRoot service shows

```
$ cd /HRRoot
$ addgroupentry config context.properties context.runAsOwner true
```

Granting Permissions

OSE JNDI object ownership is controlled using the session shell chown command, and permissions and access control lists are controlled by the chmod command. These commands are documented in the *Oracle9i Java Tools Reference*.

HTTP Security

HTTP security allows you to extend and refine the basic security provided by the JNDI/Oracle database security model. The OSE/OJVM supports the most popular aspects of the HTTP security model, including BASIC and DIGEST authentication, as well as authentication using Oracle Single Sign-On (OSSO).

While the database and JNDI security covers for the most part access to database objects, HTTP security determines who can access servlets from a Web client, and what HTTP requests clients can use.

Access to a protected Web service resource involves *authentication* and *authorization*. Authentication is the validation of submitted credentials, which establish that a client is known and validated by the system. Authorization is the determination that an authenticated user is allowed to perform the requested action.

There are four steps in setting up HTTP security for a Web application:

- establishing the principals of a Web service

- determining which resources are to be protected and how they are to be protected

- establishing the permissions of principals within the servlet context

- setting up or validating a security servlet in the root of the servlet context

Follow these steps to ensure that the correct base information is established to define HTTP security for your Web resources. If one or more of these steps is not followed, security can become either non-existent, or access to protected resources can be denied to users who should have it.

Establishing the Principals

Principal is the generic term for either a servlet engine user, or a group of users. A group contains users or other groups. The realm is an object in the Web service that contains and organizes the declared principals. Figure 7–1 shows that the `realm` objects are at the top level of the Web service, in the `realms` context, which is at the same level as the `config` object for the Web service.

Figure 7–1   Realm objects in a Web service
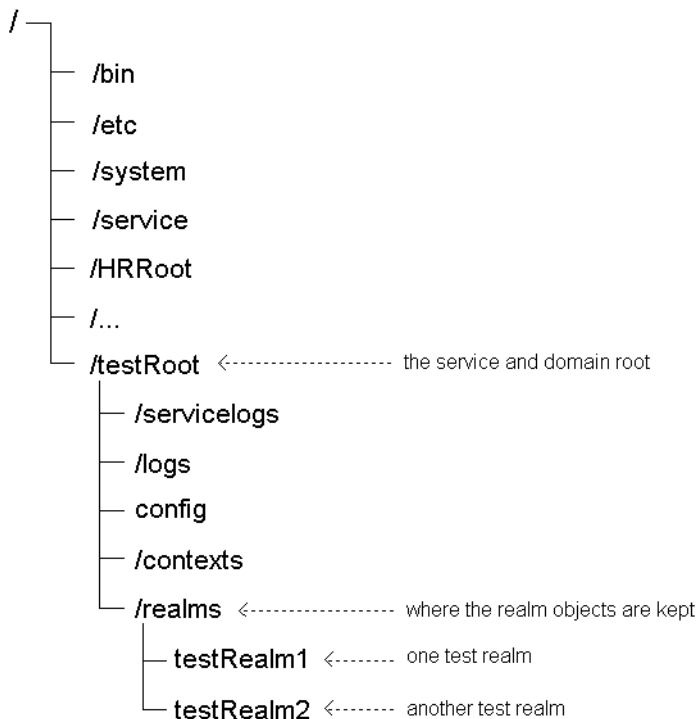
Groups

Groups contain other principals (users or other groups). Individual members of a
group inherit the permissions of the group object.

Users

Users are single objects. Unlike a group, there are no subsets of other principals
belonging to a user.

Realms

The realm is the basic unit of HTTP security in the OSE/OJVM. Each realm defines
a separate set of principals. A Web service can contain multiple realms, as shown in
Figure 7–1. The realm is the source of the valid set of principals, and the types of

principals that are handed to the server. The realm is the source of all principals, and it also determines what kind of credentials are to be used to authenticate a principal.

Each realm has a type, which determines the way the realm information originates and the way it is stored. There are four types of realms, which are:

DBUSER   Uses only database schemas as principals.

RDBMS   Keeps all the realm information directly in a database table.

JNDI   Stores all realm information in objects in the OSE JNDI namespace.

OSSO   Used for Oracle Single Sign-On management.

The DBUSER Type

A realm that has the DBUSER type derives principal definitions and permissions from the users and roles defined in the database. The implications of this are:

- No principal management is allowed using any security commands. You manage principal creation, deletion, and role membership through SQL commands in the database server, not by using the session shell `realm` commands.

- DBUSER-type realms in different Web services are the same, for identical database instances.

- The OSE/OJVM performs no case translations in the form of the principal name. This means that unless case was explicitly specified case when the database schema was created, the entire principal name is uppercase. For example, SYS and PUBLIC are uppercase, but a schema can be created with a lower case name, such as:

```
create user "steve" identified by boss;
```

In this case the schema, and hence the principal name, is "steve"—lowercase.

> **Note:** The uppercase/lowercase distinction is important when supplying user names and passwords from the browser.

The RDBMS and JNDI Types

These realm types behave the same, only the way the information about principals and groups is stored differs. You manage realms of this type using the session shell `realm` commands.

The OSSO Type

The OSSO realm type is described in "Configuring mod_osso" on page 5-22.

> **Note:** The OSSO realm type is not available in Oracle8*i* Release 3.

The Session Shell Realm Commands

The session shell `realm` commands are the tools that you use to establish, configure, and remove realms. Use these session shell commands to:

- find out what realm commands are available: `realm`

- list the realms available in a Web service: `realm list ...`

- add a new realm to a service, or remove a realm: `realm publish ...`

- add a new principal to a realm, remove an existing one, or list the users in a realm: `realm user ...`

- add a new group to a realm, or remove an existing one: `realm group ...`

- add a principal to a group, or remove an existing one: `realm parent ...`

- protect paths to Web resources: `realm map ...`

- set HTTP security permissions for HTTP requests: `realm perm ...`

This chapter lists some common ways to use the `realm` commands. Complete documentation for the `realm` commands is available in the *Oracle9i Java Tools Reference*.

Realm Configuration

To Create or Remove a Realm

To create a realm, use the `realm publish` command. Here is an example:

```
$ realm publish -webservice /testRoot -add testRealm1 -type JNDI
```

You can also remove a realm using `realm publish` with the `-remove` option. An example is:

```
$ realm publish -webservice /testRoot -remove testRealm1
```

Realm declarations reside in the JNDI namespace. You could deploy a custom realm type that you have written using the `-classname` option. Here is an example:

```
$ realm publish -w /testRoot -add myRealm -classname steve:foo.bar.MyRealm
```

In this example, the realm name and the class name are the same, but they do not have to be so.

To Create or Remove a Principal

Create a user with the `realm user` command. An example is:

```
$ realm user -webservice /testRoot -realm testRealm1 -add steve -p boss
```

To create a group use realm group, as follows:

```
$ realm group -webservice /testRoot -realm testRealm1 -add HRgroup -p gpswd1
```

With either of these commands, if the password is left blank, the principal name is used for the password.

You can delete a user as follows:

```
$ realm user -webservice /testRoot -realm testRealm1 -remove steve
```

To delete a group use the realm group command. An example is:

```
$ realm group -webservice /testRoot -realm testRealm1 -remove HRgroup
```

To List Users and Groups

Use the `realm user` command to list the users in a realm, as shown in this example:

```
$ realm user -webservice /testRoot -realm testRealm1
```

Use realm group to list groups in a realm. For example:

```
$ realm group -webservice /testRoot -realm testRealm1
```

To Add, Remove, or List the Principals for a Group

Use the `parent` variant of the `realm` command to add a principal to a group. Here is an example:

```
$ realm parent -w /testRoot -realm testRealm -group group1 -add user1
```

Remove a principal from a group also using `realm parent`, with the `-remove` option, as shown in:

```
$ realm parent -w /testRoot -realm testRealm -group group1 -remove user1
```

You can also list principals within a group by using the `realm parent` command. For example:

```
$ realm parent -w /testRoot -realm testRealm -group group1
```

To query which groups a principal is a member of use the `-query` option. For example:

```
$ realm parent -w /testRoot -realm testRealm -query user1
```

> **Notes:** Not all realms support the query option. For example, DBUSER realms do not support this kind of principal manipulation.

Where Realms Are Located

When you declare realms for a service, they are located in a `realms` subcontext of the service. For JNDI-type realms, there are additional subcontexts within the `realms` context that contain the realm's principal declarations.

Removing the Web service `realms` context removes all realm definitions for the service, such as user and group names, permission mappings, and so on. However any external resources, such as table entries, would still remain. For efficient realm management, it is much better to use the session shell realm commands.

Removing subcontexts of realms can affect JNDI-type realms.

RDBMS-type realms use the following database tables:

- `JAVA$HTTP$REALM$PRINCIPAL$`— contains all principals and encoded versions of their passwords
- `JAVA$HTTP$REALM$GROUP$`— contains principal/group relationships

Note that creating an RDBMS-type realm also creates a /realms context in the Web service root, and entries in this context. But no subcontexts are created for RDBMS-type realms.

Protecting Web Resources

Realms are containers for principals, groups, *and the protection schemes that protect Web resources.* OSE HTTP security resource protection is local to the servlet context.

Resource Protection Schemes

When you need to protect a Web resource, you declare a protection scheme. The syntax for a protection scheme is

```
<authType>:<realmName> | NONE
```

So, you specify an authentication method, followed by the name of the realm to which the authentication applies, or no protection (NONE).

There are only two valid authentication methods for the OSE/OJVM, as shown below.

BASIC	BASE64 encoding, which is very insecure.
DIGEST	In the DIGEST scheme, both parties keep the password, and pass encrypted codes. The DIGEST scheme is documented in RFC 2069, at

```
http://www.w3.org/Protocols/rfc2069/rfc2069
```

Form-based and SSL schemes are not supported, though they can be plugged in through namespace entries.

Although DIGEST is far more secure than BASIC, not all browsers support it.

You can also declare resources not to be protected. This is useful when the servlet context root is to be protected. However, when the root is protected, the error pages, being part of the tree, are also protected. Delivering an error page is part of the authentication process. If the error page is protected, cycles develop, and the desired behavior is not observed.

Instead of letting the error page default as part of the tree, explicitly declare the error pages as not being protected. Use a protection scheme of <NONE>. For example:

```
$ realm map -s /testRoot/contexts/myContext -a /system/* -scheme <NONE>
```

```
$ realm map -s /testRoot/myService/contexts/myContext -a /* -scheme \
    basic:testRealm1
```

Using "realm map" to Protect Resources

The protected path is local to the servlet context. Internally, that path is normalized, enabling stable, predictable patterns for matching. This may cause the internal representation to differ from the original path used to create the protection scheme. HTTP Security will use the longest, most exact match possible when trying to apply the protection rules.

Here is an example that protects paths to resources with the BASIC protection scheme:

```
$ realm map -s /testRoot/contexts/myContext -a /doc/index.html -scheme \
    basic:testRealm1
$ realm map -s /testRoot/contexts/myContext -a /doc -scheme basic:testRealm2
$ realm map -s /testRoot/contexts/myContext -a /doc/* -scheme basic:testRealm3
```

When declarations are made, as shown in the previous example, the paths are matched to realms as in the following examples:

```
/doc/index.html -> testRealm1
/doc/foo -> testRealm3
/doc -> testRealm2
/doc/ -> testRealm2
/doc/index -> testRealm3
```

You can remove the protection on a path using the realm map command, as shown here:

```
$ realm map -s /testRoot/contexts/myContext -r /doc/index.html
```

To list all protected paths within a servlet context, use the realm map command as shown here:

```
$ realm map -s /testRoot/contexts/myContext
```

You can explicitly declare that a path not be protected. Here is an example:

```
$ realm map -s /testRoot/contexts/myContext -a /system/* -scheme <NONE>
```

To list all protected paths within a servlet context, just use realm map and specify only the service root. For example:

```
$ realm map -s /testRoot/contexts/myContext
```

The JNDI entry for protection mappings is located in the `policy` subcontext of the servlet context. Within the policy subcontext there is an object called `httpMapping`. This creates the object responsible for handling the security servlet protection mapping. By default, `httpMapping` is used as an index into the `JAVA$HTTP$REALM$MAPPING$` database table. The HTTP realm mapping table contains all the mapped paths. Using JNDI entry manipulation you could introduce a customized version of `httpMapping`.

Declaring Permissions

Permissions are the most complicated of all HTTP security declarations, because they tie service-scoped entities with servlet context-scoped entities and reside in the servlet context themselves.

To set up a permission declaration, supply the following information:

- the Web service
- the realm within the specified service
- the servlet context within the specified Web service
- principal(s) within the realm
- the path to which the permission applies
- whether the permission is being granted or denied
- HTTP request types to be protected

Given all the pieces that are being tied into one permission declaration, it is easy to see why these are the most complicated declarations.

HTTP Request Types HTTP security permissions concern only valid HTTP request methods: GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS.

Examples of Permission Declarations

Declare a granted permission on `/foo/index.html` for `user1` for GET and POST:

```
$ realm perm -w /testRoot -realm testRealm1 -s /testRoot/contexts/myContext -n \
    user1 -u /foo/index.html + get,post
```

Declare a denied permission on `/foo/*` for `user1` for PUT and DELETE:

```
$ realm perm -w /testRoot -realm testRealm1 -s /testRoot/contexts/myContext -n \
    user1 -u /foo/* - put,delete
```

Clear granted permissions on `/foo/index.html` for `user1`:

```
$ realm perm -w /testRoot -realm testRealm1 -s /testRoot/contexts/myContext -n \
    user1 -u /foo/index.html +
```

List all permissions for `user1`:

```
$ realm perm -w /testRoot -realm testRealm1 -s /testRoot/contexts/myContext -n \
    user1
```

In the `policy` subcontext of a servlet context, there is a `config` object. This entry is used to create the object responsible for all permission declaration checks. The object is used as a key into the permissions table: `JAVA$HTTP$REALM$POLICY$`.

Declaring A Security Servlet

All HTTP security is declared through JNDI namespace entries. This is also true for the servlet that does the enforcing of security. In the servlet context, if there is a `PrivilegedServlet` named `httpSecurity`, that servlet is added as the first pre-filter for all requests within that servlet context.

Any customization is allowed as long as the `PrivilegedServlet` interface is implemented. The purpose of this servlet is to either:

■   raise an `AccessControlException` during its
    `service(HttpRequest.PrivilegedAccess, HttpRequest, HttpResponse)` if there is a perceived security violation

or

■   not raise an exception if the request is to be allowed

After authentication and authorization have taken place, the servlet must set specific authenticated principal values on the request itself. This is the user information that can be retrieved from the request by any executing servlet.

Creating a Security Servlet

You can use `realm secure` to create a security servlet. For example:

```
$ realm secure -s /testRoot/contexts/myContext
```

Removing the security servlet removes all security enforcement in a servlet context. If the entry is missing, the Web server continues execution with no security enforcement.

To remove a security servlet, type:

```
$ rm /myDomain/contexts/myContext/httpSecurity
```

> **Note:** The servlet is not published in `named_servlets` but within the servlet context directory itself.

Examples

There are two security-related examples in the $ORACLE_HOME/demo directory of your distribution. The demos are reproduced here.

rdbmsRealm

This example protects the path `/event_log*` with a realm that uses a database table for its source of principals. Note that paths in permission declarations are relative to the servlet context mapping.

This example presupposes that the sample database has already been setup. The prerequisites are:

- the Web server is installed and operational

- a Web domain is located in `/HRRoot`

- there is a context `/HRRoot/contexts/HRContext` that has a virtual path-mapping of `ose`

- there is no realm named `docRealmExample` in the domain

First, make sure that there is a security servlet for the servlet context:

```
$ realm secure -s /HRRoot/contexts/HRContext
```

Publish a realm that uses a database table for its users:

```
$ realm publish -w /HRRoot -add docRealmExample -type rdbms
```

Create a user in the realm:

```
$ realm user -w /HRRoot -realm docRealmExample -add alex -p welcome
```

Create a group in the realm:

```
$ realm group -w /HRRoot -realm docRealmExample -add docGroup -p welcome
```

Add "alex" to the `docGroup`:

```
$ realm parent -w /HRRoot -realm docRealmExample -group docGroup -add alex
```

Allow `docGroup` to execute HTTP requests with the GET and POST methods:

```
$ realm perm -w /HRRoot -realm docRealmExample -s \
    /HRRoot/contexts/HRContext -name docGroup -path /event_log + get,post
```

Protect the resource `/event_log`:

```
$ realm map -s /HRRoot/contexts/HRContext -add /event_log -scheme \
    Basic:docRealmExample
```

Now, when a client tries to access /ose/event_log the browser prompts for a username and password. Be sure to type in the username with the correct capitalization ("alex"). Username/password is alex/welcome.

You could also enter username: docGroup password: welcome

To remove the password protection without removing the realm declaration, execute the following session shell command:

```
$ realm map -s /HRRoot/contexts/HRContext -remove /event_log
```

dbUserRealm

dbUserRealm is a simple example that protects the path /doc*, and only allows the database user HR access to it. Note that paths in permission declarations are relative to the servlet context mapping.

This example presupposes that the sample database has already been setup. The prerequisites are:

- the Web server is installed and operational

- a Web domain is located in /HRRoot

- there is a servlet context /HRRoot/contexts/HRContext that has a virtual path mapping ose

- the database schema HR has been installed

- there is no realm named dbUserExample in the domain

First, be sure that there is a security servlet for the /HRRoot/contexts/HRContext servlet context:

```
$ realm secure -s /HRRoot/contexts/HRContext
```

Next, publish a realm that uses database users as for its principals:

```
$ realm publish -w /HRRoot -add dbUserExample -type dbuser
```

Allow HR to execute HTTP requests with the GET and POST methods:

```
$ realm perm -w /HRRoot -realm dbUserExample -s \
    /HRRoot/contexts/HRContext -name HR -path /http_log + get,post
```

Protect the resource `/http_log`:

```
$ realm map -s /HRRoot/contexts/HRContext -add /http_log -scheme \
     Basic:dbUserExample
```

Now, when a client tries to access `/ose/http_log` for the HR demo server the browser prompts for username and password. Be sure that the letter case matches exactly. In this case, the username is literally "HR", and the password is "hr".

To remove the password protection without removing the realm declaration, execute the following session shell command:

```
$ realm map -s /HRRoot/contexts/HRContext -remove /http_log
```

Troubleshooting

There are several layers of suspected problems to eliminate when debugging HTTP security. This minimal checklist helps you get started trouble shooting.

- Check spelling of all realm names, user names, and URI specifications.

- If using a DBUSER-type realm, make sure that the case is correct for principals.

- Set your browser cache to check for newer versions of pages every time.

- Clear browser cache(s).

- After setting a Web server sessions property, make sure you are testing against a new Web server session. The information may not be propagated to current active sessions. Do this by closing all running browsers and starting a new browser.

- Be sure that all four stages of security declarations are in place. If any are missing or incorrect, the results are unpredictable.

- Be sure that the type of authentication specified is supported by your browser. For example, by default, Netscape 4.7 does not support Digest authentication. Netscape will treat it as just Basic authentication (raising a dialog box). However, the Basic authentication response does not work for Digest authentication. This is misleading when the expected Netscape prompt displays, because it actually appeared for the wrong reasons.

- Use the shell to query the entities involved. Check that the information is declared in a way that defines your security goals.

- For example, if /doc/index.html is to be accessible only to user1 in myRealm, using BASIC authentication, then the following must exist:

  - A realm named myRealm within the domain.

  - The realm must contain a user named user1, with a known password.

  - A mapping of /doc/index.html or some more general path to a protection scheme BASIC:myRealm within the servlet context.

  - A security servlet declared for the servlet context.

  - A permission granting GET rights to the user named user1 for /doc/index.html (or a more general path)

8

Oracle WAR Deployment

This chapter describes Web archive (WAR) deployment to the Oracle9*i* database for installation and execution of Web applications in the Oracle Servlet Engine. Although the Oracle implementation follows the general WAR deployment standard, there are special considerations and logistics for execution in the Oracle9*i* JVM. Oracle offers utilities to help with these logistics. The following topics are discussed in this chapter:

- Standard Web Applications and Hierarchies

- Overview of WAR Deployment to the Oracle9i Database

- Oracle Auxiliary Descriptor

- Oracle WAR Deployment Tool Functionality

- Oracle WAR Deployment Tool Usage

- Sample Application Hierarchy and Descriptor Files

- Current Restrictions

> **Important:** Wherever the Oracle9*i* database is mentioned, this documentation applies equally to the Oracle8*i* Release 3 database.

Standard Web Applications and Hierarchies

A Web application is a collection of components that are bundled and run as an integrated unit. Web application components can include HTML pages (or other static components, such as image files or sound files), servlets, JavaServer Pages (JSP pages) and tag libraries, JavaBeans, Java utility classes, and other resources. (It is typical for libraries of classes and resources to be packaged in JAR files.) You can run such an application in any standard container (servlet engine) from any vendor. In addition, there must be some kind of top-level meta information to tie all the components together.

This section briefly summarizes the standard concepts of Web applications and Web archive (WAR) files, which provide the mechanism for deploying a Web application to a target environment. The following topics are introduced:

- Web Application Servlet Contexts
- Web Application Hierarchies
- Web Application Deployment Descriptors
- Web Application Deployment and WAR Files

Web Application Servlet Contexts

Each Web application is represented by a *servlet context*, which maintains application state information and which you can think of as an application container. As defined in the servlet 2.2 specification, a particular servlet context is represented in Java as an instance of a class that implements the standard `javax.servlet.ServletContext` interface. See "Servlet Contexts" on page 2-32 for more information about servlet contexts.

In general (not considering the Oracle9*i* JVM in particular), this would be state information for all instances of the different application components running within a given Java virtual machine. This is similar to the way a session maintains state information for a single client on the server; however, in most environments a servlet context is not specific to any single user and can potentially handle multiple clients.

The model differs in the Oracle9*i* JVM, however, where there is just a single user for each JVM session. In this case, any particular servlet context instance is managing state information for just a single instance of the application and for just a single user.

Web Application Hierarchies

A Web application has a hierarchy that is rooted at a specific directory path within a Web server. As described in the servlet 2.2 specification, this hierarchy can exist in a file system, an archive file (such as a WAR file, described in "Web Application Deployment and WAR Files" on page 8-6), or some other form for deployment.

When you create a servlet context for an application, you associate its root location, or document root, with a *context path* (a name that you choose), and the context path becomes part of the URL to access the application.

Publishing an application component (such as a servlet or JSP page) is a process to make the component available for execution. When you publish a component, you typically specify a virtual path (or servlet path), which determines the rest of the URL to access the component. The virtual path typically represents the location of the component within the application hierarchy.

For example, if a customer service application on host `www.corphomepage.com` has a context path of `custservice`, then all requests using URLs that start with the following prefix will be routed to the servlet context that represents the customer service application:

```
http://www.corphomepage.com/custservice
```

An `index.html` file in the application document root directory, for example, would be served as a result of a request to the following URL (and you can say that `index.html` is the virtual path):

```
www.corphomepage.com/custservice/index.html
```

If a JSP page, `mypage.jsp`, has a virtual path of `jsp/mypage.jsp` (and so presumably, but not necessarily, is in a `jsp` subdirectory under the document root directory), you would access it as follows:

```
www.corphomepage.com/custservice/jsp/mypage.jsp
```

There is a `WEB-INF` subdirectory of the document root, according to the servlet 2.2 specification, which contains the application deployment descriptor (described in the next section, "Web Application Deployment Descriptors"). Subdirectories of `WEB-INF` contain any components of the application other than document pages to be served to the client. For example, components under the `WEB-INF` directory might include servlets, JavaBeans, and utility classes. Components outside the `WEB-INF` directory include HTML pages and perhaps JSP pages.

Following are the contents of the `WEB-INF` directory, according to the servlet 2.2 specification:

- `/WEB-INF/web.xml` file (the deployment descriptor)

- `/WEB-INF/classes` directory (for `.class` files for servlet, JavaBean, and utility classes)

- `/WEB-INF/lib` directory (for JAR files containing libraries of Java `.class` files and resources)

Here is a sample hierarchy for a Web application (in a WAR file):

```
/index.html
/welcome.jsp
/images/logo.gif
/WEB-INF/web.xml
/WEB-INF/lib/beans.jar
/WEB-INF/classes/com/custservice/servlets/CSLogServlet.class
```

Web Application Deployment Descriptors

The servlet 2.2 specification provides a mechanism known as a Web application *deployment descriptor* to specify the elements and configuration of an application. The deployment descriptor is an XML file, named `web.xml` by convention.

The `web.xml` file includes information and settings for such items as servlet context configuration parameters, session configuration parameters, servlet and JSP definitions, servlet and JSP mappings, MIME type mappings, error pages, and security.

The DTD that defines XML grammar for a `web.xml` file is included in Chapter 13 of the Sun Microsystems *Java Servlet Specification, Version 2.2.*

Oracle provides a local copy of this DTD and uses it for validation by default if you do not specify a DTD in the `web.xml` DOCTYPE declaration.

Following is a basic example of a `web.xml` file (taken from the servlet 2.2 specification):

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
   <display-name>A Simple Application</display-name>
   <context-param>
      <param-name>Webmaster</param-name>
```

```
      <param-value>webmaster@mycorp.com</param-value>
   </context-param>
   <servlet>
      <servlet-name>catalog</servlet-name>
      <servlet-class>com.mycorp.CatalogServlet</servlet-class>
      <init-param>
         <param-name>catalog</param-name>
         <param-value>Spring</param-value>
      </init-param>
   </servlet>
   <servlet-mapping>
      <servlet-name>catalog</servlet-name>
      <url-pattern>/catalog/*</url-pattern>
   </servlet-mapping>
   <session-config>
      <session-timeout>30</session-timeout>
   </session-config>
   <mime-mapping>
      <extension>pdf</extension>
      <mime-type>application/pdf</mime-type>
   </mime-mapping>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
      <welcome-file>index.html</welcome-file>
      <welcome-file>index.htm</welcome-file>
   <welcome-file-list>
   <error-page>
      <error-code>404</error-code>
      <location>/404.html</location>
   </error-page>
</web-app>
```

As described in "Distributable Applications and the Oracle Servlet Engine" on page 8-7, a Web application must be designed as distributable to run in the Oracle Servlet Engine. The web.xml file specifies this through the presence of a distributable element, defined as follows in the web.xml DTD:

```
<!ELEMENT distributable EMPTY>
```

For example:

```
<distributable />
```

> **Note:** According to the servlet 2.2 specification, a `web.xml` file can use the `env-entry`, `ejb-ref`, and `resource-ref` elements to refer application components to external resources and EJBs without explicit knowledge of their location or organization. The `ejb-ref` element, for example, would contain information to help a servlet find the home interfaces of an EJB. For more information, see the Sun Microsystems *Java Servlet Specification, Version 2.2*.
>
> The Oracle WAR implementation does not currently support these tags, however. See "Current Restrictions" on page 8-59 for more information.

Web Application Deployment and WAR Files

A Web archive (WAR) file is a single file that contains all the components of a Web application and is structured according to the hierarchy of these components. It is the vehicle for deploying a Web application to the target environment where the application is intended to run.

At the target system, a deployment tool will unpackage the WAR file and place the application components according to the hierarchy specified by the WAR file structure. ("Overview of the Oracle WAR Deployment Tool" on page 8-8 describes the Oracle tool.)

WAR files are identified by the `.war` file name extension and can be created using any standard Java archive (JAR) tool that allows any or all of the application components to be signed.

Overview of WAR Deployment to the Oracle9*i* Database

This section provides an overview of details, considerations, and mechanisms in deploying and running a Web application in the Oracle9*i* database. This discussion includes the following topics:

- Distributable Applications and the Oracle Servlet Engine
- Overview of the Oracle Auxiliary Descriptor
- Overview of the Oracle WAR Deployment Tool
- Security Preparations
- Database Sessions, Servlet Context Ownership, and Application Privileges

Distributable Applications and the Oracle Servlet Engine

The Sun Microsystems servlet 2.2 specification introduces the concept of *distributable* Web applications, where application components can be deployed across multiple Java virtual machines, running on either the same host or different hosts. (Within an application that is marked as distributable, all requests that are part of a particular session can be handled only on a single JVM at any one time.)

With the scalable nature of the Oracle Servlet Engine, it is typical for many client sessions to be simultaneously active and accessing the same Web application at any given time, with each client session executing in its own virtual JVM. This means that a Web application deployed to run in OSE must be distributable.

For an application to be distributable, you must take the following steps:

- Follow a more restrictive set of rules during development of the application, as detailed throughout the servlet 2.2 specification.
- Mark the application as distributable, using the `distributable` tag in the application deployment descriptor (`web.xml`, discussed in "Web Application Deployment Descriptors" on page 8-4).

When you deploy an application to the Oracle9*i* database, the Oracle WAR deployment tool checks the application deployment descriptor and issues a warning if the application is not marked `distributable`. (Deployment will continue despite the warning.) Use the warning as a reminder to check that your application does not make assumptions about running in a single JVM instance. Update the deployment descriptor to mark the application `distributable` after you are satisfied with your analysis and have made the necessary changes.

Overview of the Oracle Auxiliary Descriptor

"Web Application Deployment Descriptors" on page 8-4 describes the standard Web application deployment descriptor file, `web.xml`. This file is a vehicle for standard configuration instructions for a Web application and is portable to any runtime environment supporting the servlet 2.2 specification. However, `web.xml` cannot provide all the information necessary to deploy an application to a particular servlet container, because each vendor is free to extend standard functionality with their own set of features. Furthermore, some aspects of deployment may be intentionally left out of the standard `web.xml` descriptor. The servlet 2.2 specification, therefore, suggests that each vendor provide an additional descriptor file for configuration of features unique to that vendor's runtime environment.

Oracle specifies and supports such an additional descriptor, known as the *Oracle auxiliary descriptor*. As with the `web.xml` deployment descriptor, the auxiliary descriptor is in XML format. Oracle provides a DTD to specify supported elements and attributes. You can choose any file name for the auxiliary descriptor, but the `.xml` file name extension is recommended.

See "Oracle Auxiliary Descriptor" on page 8-17 for more information.

Overview of the Oracle WAR Deployment Tool

Oracle offers a tool that deploys a Web application to the Oracle9*i* database for execution in the Oracle Servlet Engine. The WAR deployment tool requires that the application be packaged in a WAR file, and the tool can be invoked in any of the following ways:

- from the server, by using the Oracle session shell `deploywar` command (requires you to first manually upload the WAR file and auxiliary descriptor)

- from the server, from Java code or a PL/SQL call specification through the `oracle.mts.http.deployment.DeployWar.main(String[] args)` method (this also requires you to first manually upload the WAR file and auxiliary descriptor)

- from any HTTP client, by invoking the Oracle deployment servlet (presuming the `oracle.aurora.mts.http.deployment.DeploymentServlet` servlet class has been published to the Oracle Servlet Engine in advance, which occurs as part of Oracle WAR deployment installation)

  (Oracle supplies a special form, `deploywar.htm`, as a convenient way to invoke the deployment servlet.)

- from an Oracle client (any system with an Oracle client installation), through a client-side deployment script (`deploywar` on UNIX or `deploywar.bat` on Windows NT)

- from a non-Oracle client (any system without an Oracle client installation), by executing the WAR deployment tool wrapper, `HttpClientWrapper`, directly from Java

> **Note:** The client-side scripts and client-side wrapper are just convenient front ends that invoke the deployment servlet.

"Vehicles for Invoking the Oracle WAR Deployment Tool" on page 8-46 discusses each of these ways to invoke the Oracle WAR deployment tool.

When you invoke the WAR deployment tool, it performs several automated steps, presuming that your `web.xml` file and Oracle auxiliary descriptor are configured in some appropriate way. These steps include the following:

- loading of the deployment descriptors (`web.xml` and the auxiliary descriptor)

- file loading from the WAR file, using the Oracle `loadjava` utility

  The `loadjava` utility loads Java classes, Java resources, and JSP pages into the Oracle9*i* database, including JSP translation where applicable. It also copies static pages of the application to the OSE document root directory.

- creation of a servlet context for the application

- publishing of application components (servlets and JSP pages)

- securing the application (implementing any URI protections and login/password requirements, assuming appropriate OSE security preparations were made)

"Oracle WAR Deployment Tool Functionality" on page 8-35 describes these steps in more detail.

> **Note:**   The Oracle WAR deployment tool will translate JSP pages
> (`.jsp` and `.sqljsp` files). It will not, however, translate or compile
> `.sqlj` or `.java` files. (The servlet 2.2 specification assumes that
> only compiled classes are being loaded.) Any `.java` or `.sqlj` files
> in the WAR file will be treated as Java resources and loaded as is.
> You will have to perform any server-side translation or compilation
> manually.

Security Preparations

If an application being deployed defines any security restrictions, you must
complete (or verify) appropriate preparations before deployment:

- Create the necessary security realm.

- Create users and groups for role names.

- Create a group for all realm users (if necessary).

- Decide what principals (users or groups) in OSE will correspond to role names
  in any `security-role` elements in the `web.xml` file. (This is necessary only
  for roles involved in security constraints.)

For background information about OSE security, see the *Oracle9i Oracle Servlet
Engine User's Guide.*

The application developer must provide a description of security roles and different
authorization arrangements that are required. The appearance of `login-config`,
`security-role`, or `security-constraint` elements in the `web.xml` file
indicates that the application has security features.

Create Security Realms Prior to Deployment

All HTTP security protections for an application apply within one HTTP security
realm. The realm is specified in the `login-config` element of the `web.xml` file or
Oracle auxiliary descriptor. Security roles and virtual path protections should be
found and established within this HTTP security realm.

Before deploying the WAR file, it is your responsibility to make sure the HTTP
security realm for the application has been created in OSE. You can accomplish this
using the Oracle9*i* session shell `realm publish` command, as in the following
example (`$` is the session shell prompt):

```
$ realm publish -w someService -add catalogRealm -type RDBMS
```

For more information about the `realm publish` command, see the *Oracle9i Java Tools Reference.*

The `login-config` element includes a subelement that provides the realm name, as in the following example:

```
<login-config>
  <auth-method>basic</auth-method>
  <realm-name>catalogRealm</realm-name>
</login-config>
```

The HTTP security realm name specified in the `realm-name` subelement (`catalogRealm` in this example) must match the name of the realm that was created prior to deployment. This example also specifies that the application will be authenticated (by the Oracle Servlet Engine) according to the `basic` authentication method.

The `login-config` element is optional in the `web.xml` file and can be specified in the Oracle auxiliary descriptor instead, which is useful if you are deploying the application to different environments and do not want to change the `web.xml` file each time. The value in the Oracle auxiliary descriptor takes precedence over the value in `web.xml`.

If the `login-config` element is not specified in either descriptor, but the `web.xml` file contains some security constraints, the Oracle WAR deployment tool will issue an error. (It is impossible to configure any security constraints without knowing the security realm and authentication method.)

Create Users and Groups for Role Names Prior to Deployment

The Oracle WAR deployment tool will protect Web resources according to directives indicated by `security-constraint` elements in the `web.xml` file. The following example illustrates the structure of these elements:

```
<security-constraint>
    <web-resource-collection>
      ...
    </web-resource-collection>

    <auth-constraint>
       <role-name>catalogUser</role-name>
       <role-name>catalogBuilder</role-name>
    </auth-constraint>

    <user-data-constraint>
```

```
    ...
    </user-data-constraint>
</security-constraint>
```

Web resources (such as the list of relative virtual paths within the application context) and HTTP methods such as GET, POST, and HEAD (by which these paths are accessed) must be protected so that only certain authenticated principals can access the resources. The `web-resource-collection` subelement in `web.xml` defines the Web resources. The principals—users and groups—are listed in `role-name` subelements of the `auth-constraint` subelement.

During deployment, the Oracle WAR deployment tool will ensure appropriate protection of Web resources at runtime, but cannot decide what groups of users in your realm correspond to the roles used by the application.

Before deploying the WAR file, it is your responsibility to make sure users and groups for all role names used in application security constraints have been created. Users and groups must be created as appropriate for all role names specified by `role-name` subelements of any `security-role` elements in the `web.xml` file.

> **Note:** The servlet 2.2 specification requires that each role name used in an authorization constraint be declared in a separate `security-role` element in the `web.xml` file. If this requirement is not followed, the Oracle WAR deployment tool will issue an error.

Create a Group for All Realm Users Prior to Deployment (if necessary)

The `web.xml` DTD in the servlet 2.2 specification allows a `security-constraint` element with an `auth-constraint` subelement that does not contain any role name, as in the following example:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>SalesInfo</web-resource-name>
    ...
    </web-resource-collection>

    <auth-constraint>
       <description>
          all authorized users in realm should be allowed access
       </description>
    </auth-constraint>
```

```
</security-constraint>
```

The specification does not, however, clarify what such a constraint means. It is reasonable to assume it means that all users of the application realm are authorized to access the Web resource collection. Such protection is sensible because it requires that all clients authorize themselves in the given realm before they can use a Web resource.

There is no direct way in the OSE HTTP security mechanism to protect a resource for all existing and future users in the realm, but there is a convenient workaround. To achieve the desired effect, you can create a group that consists of all users in the realm, and give appropriate permissions to that group. You also must edit the web.xml file to add a role-name subelement with the name of the newly created group.

For example, without a workaround the above security constraint will cause the following error message during WAR deployment:

```
WARNING: Web Resource Collection "SalesInfo" is defined,
but the list of Security Role names is empty in its
<auth-constraint> element.
In this release you need to provide a Security Role name in order to protect a
web resource.
Web resource "SalesInfo" will not be protected.
```

To remedy the situation, issue an appropriate realm group command from the Oracle9*i* session shell. Consider the following example, which assumes the application is deployed in the domain testRoot:

```
realm group -w testRoot -realm catalogRealm -add allUsrs
```

Continuing the example, you must then use appropriate realm parent commands, such as the following, to put desired principals into the allUsrs group:

```
realm parent -w testRoot -realm catalogRealm -group allUsrs -add tyrone
```

(This adds the OSE principal tyrone to the group allUsrs.)

In addition, you must add a new security-role element to the web.xml file, as follows:

```
<security-role>
   <role-name>allUsrs</role-name>
</security-role>
```

You must also add the following subelement under the `auth-constraint` subelement of the `web.xml` `security-constraint` element:

```
<role-name>allUsrs</role-name>
```

For more information about Oracle9i session shell commands such as `realm group` and `realm parent`, see the *Oracle9i Java Tools Reference.*

Authenticating with Oracle Single Sign-On

A simpler way to authenticate is to use the Oracle Single Sign-On server (OSSO). With OSSO authentication all of the authorization checks also take place on the Oracle HTTP Server prior to the request being forwarded to the OSE. Thus, any `<security-constraint>` and `<security-role>` elements that are present in either the `web.xml` or in the Oracle auxiliary descriptor file do not make sense for OSSO, and are ignored. This allows administrators to reuse shrink-wrapped WAR files with minimal changes configure applications to use OSSO authentication.

For additions to the DTD to support OSSO, see "login-config" on page 8-28.

Database Sessions, Servlet Context Ownership, and Application Privileges

The Oracle Servlet Engine, and any components of your application, will execute within an Oracle9*i* JVM instance and, therefore, within a particular database session. Additionally, when you run the Oracle WAR deployment tool, the tool will run in a particular database session, and this session is used to deploy your application. In particular, when the new servlet context for your application is created, it will be owned by the database schema of the deployment session.

For your application to run properly, you must be aware of the database schema that will own the servlet context, the database schema where your application classes and resources will be loaded, and the database schemas that are likely to be used in running your application. These considerations will determine whether your application will have the necessary privileges to run properly and access desired database entities such as tables and stored procedures.

The remainder of this section describes what determines the schema of the deployment database session, and discusses special considerations regarding application privileges.

Deployment Database Session Initiation and Schema

Depending on how you invoke the Oracle WAR deployment tool, the deployment database session schema may be determined either explicitly or implicitly.

Explicit Deployment Session Initiation There are a couple of ways to explicitly determine the deployment database session schema:

- using the session shell `deploywar` command

  When you log in to the Oracle9*i* session shell, you supply a database user name and password, and a database session is started for that user schema. Any subsequent session shell command, including `deploywar`, is executed within that session, by that schema. When you run `deploywar`, this schema becomes the deployment database session schema.

- using Java or a PL/SQL call specification to invoke `DeployWar.main()`

  When you log in to the database and invoke the `DeployWar.main()` method, either directly from Java or through a PL/SQL call specification, the schema you logged in as becomes the deployment database session schema.

Implicit Deployment Session Initiation All WAR deployment tool vehicles that invoke the deployment servlet—including the `deploywar.htm` form, the client-side script (for Oracle clients), or the client-side deployment tool wrapper (for non-Oracle clients)—start a deployment database session in which the deployment servlet runs.

According to general OSE policy, the schema of this database session is determined as follows:

- Generally speaking, the deployment database session schema is the schema that owns the domain of the servlet context in which the deployment servlet runs. (This is the servlet context in which it was published.) This is `SYS` for the copy of the deployment servlet that is published automatically when an Oracle9*i* database is created. If you publish additional copies of the deployment servlet, or if you install the Oracle WAR deployment tool onto an existing Oracle8*i* 8.1.7 database, then the schema depends on how you published the deployment servlet.

- If the servlet context in which the deployment servlet runs was published with the `run-as-owner` property set to `true`, then the deployment database session schema is the schema that owns the servlet context, instead of the schema that owns the domain.

> **Note:** Initially, when a servlet context is created, it is owned by the schema that created it. You can change the owner later through the session shell `chown` command. You can verify the owner through the following session shell command (`$` is the session shell prompt):
>
> ```
> $ ls -l <context_name>
> ```

Special Deployment Considerations Regarding Application Privileges

To help ensure that your application will run properly, consider the following points when you deploy it:

- If the `run-as-owner` attribute of the `context-descriptor` element is set to `true` in the Oracle auxiliary descriptor during deployment, all servlets in your application will run with privileges of the owner of the servlet context that was created during deployment of your application.

- If your Oracle auxiliary descriptor specifies the `schema` subelement under the `jserver-loader` subelement of the `class-loader-descriptor` element, then that is the schema where your application code will be loaded. If you do not specify this `schema` element, then your application will be loaded into the deployment database session schema. ("Auxiliary Descriptor Element and Attribute Descriptions" on page 8-23 describes these elements.)

Oracle Auxiliary Descriptor

This section describes the Oracle auxiliary descriptor, introduced in "Overview of the Oracle Auxiliary Descriptor" on page 8-8. The auxiliary descriptor, used in conjunction with the standard `web.xml` file, is for application settings specific to the Oracle environment.

This section is organized as follows:

- Auxiliary Descriptor DTD
- Auxiliary Descriptor Element and Attribute Descriptions
- Sample Auxiliary Descriptor

Auxiliary Descriptor DTD

This section contains the complete DTD for the Oracle auxiliary descriptor.

```
<!--
This is the XML DTD for the Oracle Auxiliary Descriptor
Version 1.0.1, Dec. 2000
-->

<!--
The oracle-auxiliary-descriptor element is the root element of the Oracle
specific deployment descriptor.
-->
<!ELEMENT oracle-auxiliary-descriptor (description?, context-descriptor,
log-descriptor?, jsp-info?, class-loader-descriptor?, login-config?,
security-role-mapping*)>

<!--
The description element is used to provide descriptive text about the parent
element.
-->
<!ELEMENT description (#PCDATA)>

<!--
The context-descriptor contains the servlet context information.
-->
<!ELEMENT context-descriptor (description?, default-info?, accept-info?)>


<!--
The debug flag specifies whether to print servlet debug information.
```

```
-->
<!ATTLIST context-descriptor debug (true|false) "false">

<!--
The run-as-owner flag specifies whether to allow the user to have the owner's
permissions if the user schema is not the same as the serlvet owner's.
-->
<!ATTLIST context-descriptor run-as-owner (true|false) "false">

<!--
The browse-dirs flag specifies whether to allow the user to see the directory
structure if the welcome file is missing.
-->
<!ATTLIST context-descriptor browse-dirs (true|false) "false">

<!--
The name attribute is the JNDI name of the servlet context to be created.
-->
<!ATTLIST context-descriptor name CDATA #IMPLIED >

<!--
The virtual-path attribute is the virtual path to the servlet context.
-->
<!ATTLIST context-descriptor virtual-path CDATA #IMPLIED >

<!--
The doc-root attribute is the full path of the doc root directory on a file
system.
-->
<!ATTLIST context-descriptor doc-root CDATA #IMPLIED >

<!--
The stateless flag specifies whether the servlet context is stateless.
-->
<!ATTLIST context-descriptor stateless (true|false) "false">

<!--
The default-info element specifies NLS information for the server to use in
interpreting requests.
-->
<!ELEMENT default-info (description?, charset*, language*)>

<!--
The accept-info element specifies NLS information for the server to use in
sending responses.
```

```
-->
<!ELEMENT accept-info (description?, charset?, language?)>

<!--
The charset element contains the name of a character set.
-->
<!ELEMENT charset (#PCDATA)>

<!--
The language element contains the name of a langauge.
-->
<!ELEMENT language (#PCDATA)>

<!--
The log-descriptor element contains http access, event, and error log
information.
-->
<!ELEMENT log-descriptor (description?, error-log?, event-log?,
http-access-log?)>

<!--
The error-log element contains the information on how errors are logged.
-->
<!ELEMENT error-log (system-log | rdbms-log | file-log)>

<!--
The event-log element contains the information on how events are logged.
-->
<!ELEMENT event-log (system-log | rdbms-log | file-log)>

<!--
The http-access-log element contains the information on how http-accesses are
logged.
-->
<!ELEMENT http-access-log (system-log | rdbms-log)>

<!--
The optional name attribute of error-log is the name of the JNDI node,
containing the error log object, relative to the context directory.
-->
<!ATTLIST error-log name CDATA #IMPLIED>

<!--
The optional name attribute of event-log is the name of the JNDI node,
containing the event log object, relative to the context directory.
```

```
-->
<!ATTLIST event-log name CDATA #IMPLIED>

<!--
The optional name attribute of http-access-log is the name of the JNDI node,
containing the http-access log object, relative to the context directory.
-->
<!ATTLIST http-access-log name CDATA #IMPLIED>

<!--
The system-log element specifies the log is written into System.out
-->
<!ELEMENT system-log EMPTY>

<!--
The rdbms-log element specifies the log is written into the table, given by the
'table' attribute.   It is advisable to specify database schema as the prefix in
table name.  If schema is omitted, the table will be created in the schema used
for loading Java objects of the application.
-->
<!ELEMENT rdbms-log EMPTY>

<!--
The required table attribute of rdbms-log element is the table name for writing
the log into a database.
-->
<!ATTLIST rdbms-log table CDATA #REQUIRED>

<!--
The file-log element specifies the log is written into the file, given by the
'file' attribute.
-->
<!ELEMENT file-log EMPTY>

<!--
The required file attribute of file-log element is the file name for writing the
log into a file.
-->
<!ATTLIST file-log file CDATA #REQUIRED>

<!--
The jsp-info element contains setting information for JavaServer Pages.
-->
<!ELEMENT jsp-info (description?, hotload?)>
```

```
<!--
The hotload element specifies the JSP pages to be hotloaded.
-->
<!ELEMENT hotload (description?, jsp*)>

<!--
The all attribute of the hotload element overrides the jsp page list if it is
true.
-->
<!ATTLIST hotload all (true|false) "false">

<!--
Each jsp element contains the name of a jsp page that is in the WAR file and is
to be hotloaded.
-->
<!ELEMENT jsp (#PCDATA)>

<!--
The class-loader-descriptor element provides information necessary to load
application classes and resources.
-->
<!ELEMENT class-loader-descriptor (description?, jserver-loader)>

<!--
The jserver element holds information necessary to load java objects into the
database.
-->
<!ELEMENT jserver-loader ( schema?, resolver?)>

<!--
The schema element must be a valid database schema name. If omitted, the web
application will be loaded into the schema which invoked the deployment tool.
Schema element is case sensitive.
-->
<!ELEMENT schema (#PCDATA)>

<!--
The resolver element must use syntax for the '-resolver' option of the
'loadjava' tool. If omitted, the default oracle resolver is used.
-->
<!ELEMENT resolver (#PCDATA)>

<!--
The login-config element takes precedence over the login-config in web.xml.
-->
```

```
<!ELEMENT login-config (description?, auth-method, realm-name?)>

<!--
The auth-method element is the name of the security authentication method.
-->
<!ELEMENT auth-method (#PCDATA)>

<!--
The realm-name element is the name of the OSE security realm.
-->
<!ELEMENT realm-name (#PCDATA)>

<!--
The security-role-mapping element maps a logical security role name to a
principal in the OSE HTTP security realm.
-->
<!ELEMENT security-role-mapping (description?, security-role, ose-principal)>

<!--
The security-role element contains the logical security role name.
-->
<!ELEMENT security-role (description?, role-name)>

<!--
The role-name element contains the name of a logical role. It must also appear
in a security-role element of the web.xml application descriptor.
-->
<!ELEMENT role-name (#PCDATA)>

<!--
The ose-principal must be the existing user or group in the HTTP security
domain, defined by the login-config element in this descriptor or in the
web.xml.
-->
<!ELEMENT ose-principal (#PCDATA)>

<!--
The ID mechanism is for future references to the elements of this auxiliary
deployment descriptor.
-->

<!ATTLIST oracle-auxiliary-descriptor id ID #IMPLIED>
<!ATTLIST description id ID #IMPLIED>
<!ATTLIST context-descriptor  id ID #IMPLIED>
<!ATTLIST default-info id ID #IMPLIED>
```

```
<!ATTLIST accept-info id ID #IMPLIED>
<!ATTLIST charset id ID #IMPLIED>
<!ATTLIST language id ID #IMPLIED>
<!ATTLIST log-descriptor id ID #IMPLIED>
<!ATTLIST error-log id ID #IMPLIED>
<!ATTLIST event-log id ID #IMPLIED>
<!ATTLIST http-access-log id ID #IMPLIED>
<!ATTLIST system-log id ID #IMPLIED>
<!ATTLIST rdbms-log id ID #IMPLIED>
<!ATTLIST file-log id ID #IMPLIED>
<!ATTLIST jsp-info id ID #IMPLIED>
<!ATTLIST hotload id ID #IMPLIED>
<!ATTLIST jsp id ID #IMPLIED>
<!ATTLIST class-loader-descriptor id ID #IMPLIED>
<!ATTLIST jserver-loader id ID #IMPLIED>
<!ATTLIST schema id ID #IMPLIED>
<!ATTLIST resolver id ID #IMPLIED>
<!ATTLIST login-config id ID #IMPLIED>
<!ATTLIST auth-method id ID #IMPLIED>
<!ATTLIST realm-name id ID #IMPLIED>
<!ATTLIST security-role-mapping id ID #IMPLIED>
<!ATTLIST security-role id ID #IMPLIED>
<!ATTLIST role-name id ID #IMPLIED>
<!ATTLIST ose-principal id ID #IMPLIED>
```

Auxiliary Descriptor Element and Attribute Descriptions

Element names used in the auxiliary descriptor DTD are largely self-explanatory given an understanding of Web servers in general and the Oracle Servlet Engine in particular.

This section summarizes and briefly describes the elements, subelements, and attributes. For background information, see the *Oracle9i Oracle Servlet Engine User's Guide* and *Oracle9i Java Tools Reference* (particularly information about the session shell createcontext command).

> **Note:** All elements have an optional description subelement where you can enter a descriptive comment, but these are omitted in the following discussion.

context-descriptor Use subelements and attributes of this top-level element to indicate properties of the servlet context.

Subelements:

- `default-info`: This has subelements to specify NLS information for the server to use in interpreting requests.

  Subelements:

  - `charset`: Use `charset` subelements of `default-info` to specify NLS character sets for the server to try in interpreting each request.

  - `language`: Use `language` subelements of `default-info` to specify NLS language codes for the server to try in interpreting each request.

  A `default-info` element can have multiple `charset` or `language` subelements. They will be tried in the order presented.

- `accept-info`: This has subelements to specify NLS information for the server to use in sending responses.

  Subelements:

  - `charset`: Use a `charset` subelement of `accept-info` to specify the preferred NLS character set of the server. This character set will be returned through an `accept-charset` header in the response.

  - `language`: Use a `language` subelement of `accept-info` to specify the preferred NLS language encoding of the server. This language encoding will be returned through an `accept-language` header in the response.

  An `accept-info` element can have, at most, one `charset` subelement and one `language` subelement.

  Note: Refer to the HTTP 1.1 specification for more information about these NLS-related elements.

Attributes:

- `name`: This is the JNDI name of the servlet context to be created (for example, `testContext`).

- `virtual-path`: This is the virtual path to the servlet context.

- `doc-root`: This is the full file-system path of the document root directory for the servlet context.

- **debug** (`true` or `false`; default `false`): When the `debug` attribute is enabled, debugging information—headers and request line of the request, and headers and response line of the response—is printed into the HTTP access log.

- **run-as-owner** (`true` or `false`; default `false`): A servlet context exists inside a domain. By default, any servlet in any servlet context in a given domain executes with permissions of the domain owner. However, any particular servlet context can have a separate owner. (The owner can be changed through a JNDI `chown` command.) By setting `run-as-owner` to `true`, you ensure that servlets in your application execute with permissions of the owner of the particular context, instead of with permissions of the owner of the domain.

- **browse-dirs** (`true` or `false`; default `false`): If the document root does not have `welcome` files (as specified in the `welcome-files-list` element in the `web.xml` file), enabling `browse-dirs` allows you to still see and browse the directory structure under the document root when you make a request such as the following:

  ```
  http://foo.com/myDir
  ```

- **stateless** (`true` or `false`; default `false`; applicable only when accessing OSE through the Apache `mod_ose` module): Enabling `stateless` specifies that the application associated with the servlet context is a stateless application. (For a stateless application, the Oracle9*i* JVM database session is reused between client requests. Multiple clients are hosted by the same session, saving startup costs. Servlets are not allowed to create HTTP session objects.)

The `name`, `virtual-path`, `doc-root`, and `stateless` settings correspond to the `context_name`, `-virtualpath`, `-docroot`, and `-stateless` settings used by the WAR deployment tool in the session shell `createcontext` command to create the servlet context. Following is the format of this command (`$` is the session shell prompt; this is a single wrap-around command):

```
$ createcontext -virtualpath <path> [-recreate] [-properties <prop_groups>]
[-docroot <location>] [-stateless] <domain_name> <context_name>
```

For information about the `createcontext` command, see the *Oracle9i Java Tools Reference.*

> **Note:** Specify the domain name, which corresponds to the *domain_name* setting used by the `createcontext` command, through an Oracle WAR deployment tool command-line option. See "Oracle WAR Deployment Tool Options and Parameters" on page 8-43.

log-descriptor Use subelements of this top-level element to specify information about the error log, event log, and HTTP access log.

> **Note:** See "Sample Auxiliary Descriptor" on page 8-31 for examples of how to use the subelements of `log-descriptor`.

Subelements:

- `error-log`: This has subelements and attributes to specify where errors are logged.

  Subelements (only one of the following can be used):

  - `system-log`: Use this (empty) subelement to log errors to the `System.out` device.

  or:

  - `rdbms-log`: Use this (empty) subelement and its attribute to log errors to a database table.

    Attribute:

    * `table`: The name of the database table to use for the log. It is advisable to include the schema name in the table name, for clarity (for example, `HR.tab1` instead of `tab1`). If you do not specify a schema, then the schema used will be the one into which classes are being loaded (inferred from the `schema` subelement, if present, of the `jserver-loader` element under `class-loader-descriptor`, explained below).

      Columns for this table are as follows:

      ```
      (ID NUMBER, LINE NUMBER, TEXT VARCHAR2(4000) );
      ```

  or:

- – `file-log`: Use this (empty) subelement and its attribute to log errors to an operating system file.

  Attribute:

  - * `file`: the full path and name of the operating system file to use for the log.

  Attributes:

  - – `name`: This optional `error-log` attribute specifies the name of the JNDI node containing the log object, relative to the servlet context directory.

- ■ `event-log`: This has subelements and attributes to specify where events are logged.

  Subelements (only one of the following can be used; usage is the same as for `error-log`):

  - – `system-log`

  or:

  - – `rdbms-log`

    Attribute:

    - * `table`

  or:

  - – `file-log`

    Attribute:

    - * `file`

  Attributes:

  - – `name`

- ■ `http-access-log`: This has subelements and attributes to specify where HTTP access information is logged (this is a standard HTTP log).

  Subelements (only one of the following can be used; usage is the same as for `error-log`, except there is no possible `file-log` subelement):

  - – `system-log`

  or:

  - – `rdbms-log`

Attribute:

* table

Attributes:

– name

jsp-info Use subelements and attributes under this top-level element to specify whether to hotload JavaServer Pages.

Subelements:

- `hotload`: Use subelements and attributes of this element to specify the JSP pages to be hotloaded.

    Subelements:

    – `jsp`: Use `jsp` elements to specify JSP pages to be hotloaded (overridden by the `all` attribute). There can be multiple `jsp` elements, with one JSP page per element.

    Attributes:

    – `all` (`true` or `false`; default `false`): Use this attribute to specify that all JSP pages are to be hotloaded (overrides the `jsp` elements).

For general information about hotloading, see the *Oracle JavaServer Pages Developer's Guide and Reference.*

login-config Use subelements of this top-level element to specify login security configuration information. Usage is the same as for the `login-config` element in the standard `web.xml` file.

> **Important:** Any `login-config` settings in the auxiliary descriptor take precedence over `login-config` settings in `web.xml`.

Subelements:

- `auth-method`: This is the security authentication method, either `BASIC`, `DIGEST`, or `OSSO`. (`FORM` and `CLIENT-CERT` are not currently supported.)

    In addition to already documented values, the `<auth-method>` can also be `OSSO`. The effect is the same as securing the context in the session shell with the `realm secure -osso` command (see Securing a Servlet Context with the

OSSO Security Servlet on page 5-23). The special OSSO realm `ossoRealm` is created as a result of an `auth-method` element equal to OSSO, if it did not exist before. OSSO is the only case where <realm-name> element is optional, and may be omitted in the <login-config>, as shown in this fragment:

```
<login-config>
  <description> Context Authentication and Authorization happens on Apache
      front end and OSSO server
  </description>
  <auth-method>
    OSSO
  </auth-method>
</login-config>
```

- `realm-name`: This is the name of the OSE security realm (for example, `testRealm`). You can omit `realm-name` if the `auth-method` is OSSO.

class-loader-descriptor Use subelements under this top-level element to provide required information for loading application classes and resources.

Subelements:

- `jserver-loader`: This has subelements to specify the schema and resolver for loading application classes and resources into the Oracle database.

    Subelements:

    – `schema`: This is the schema where the application is to be loaded. It must be a valid database schema. If no schema is specified, then the application will be loaded into the schema from which the WAR deployment tool was invoked.

    > **Important:**   The `schema` element is case-sensitive. The string for the schema name must exactly match what would be returned by the following query when the user in question is logged in to the database:
    >
    >   SELECT USER FROM DUAL;

    – `resolver`: This is the resolver to be used during loading of the application. The syntax must match what would be used for the `loadjava` `-resolver` option. See the *Oracle9i Java Tools Reference* for information, or see the examples in "Sample Auxiliary Descriptor" immediately below. If no resolver is specified, then the default Oracle resolver is used. (The default

resolver for the schema specified in the `schema` subelement described above.)

Note: In a future release, when OSE will run outside as well as inside the database, there will also be a `jdk-loader` element.

---

security-role-mapping Use subelements of this top-level element, in conjunction with `security-role` elements in the `web.xml` file, to create mappings between logical security role names in the deployment environment and principals (users or groups) in the OSE HTTP security realm.

Subelements:

- `security-role`: Use subelements of this element to specify logical security role names.

  Subelements:

  - `role-name`: This is a logical security role name (such as `manager`). This role name must also appear in a `security-role` element in the `web.xml` file, and can be used in a `role-name` subelement of an `auth-constraint` subelement of a `security-constraint` element in `web.xml`.

- `ose-principal`: This is an OSE user or group in the OSE HTTP security realm that is to be granted HTTP access permissions associated with a particular security role (specified in the `role-name` element, as described immediately above). The specified principal must be an existing user or group in the security realm specified in the `realm-name` subelement of the `login-config` element in the auxiliary descriptor or `web.xml` file.

There can be multiple `security-role-mapping` elements defining `security-role` to `ose-principal` mappings.

As an example, presume the following entries in the `web.xml` file (based on the example in section 13.3.2 of the Sun Microsystems *Java Servlet Specification, Version 2.2*, with typographical errors corrected):

```
<security-role>
   <role-name>manager</role-name>
</security-role>
...
<security-constraint>
   ...
   <web-resource-collection>
      <web-resource-name>SalesInfo</web-resource-name>
```

```
      <urlpattern>/salesinfo/*</urlpattern>
      <http-method>GET</http-method>
      <http-method>POST</http-method>
   </web-resource-collection>
   ...
   <auth-constraint>
      <role-name>manager</role-name>
   </auth-constraint>
   ...
</security-constraint>
```

And assume the following entries in the auxiliary descriptor:

```
security-role-mapping>
   <security-role>
      <role-name>manager</role-name>
   </security-role>
   <ose-principal>DBA</ose-principal>
</security-role-mapping>
```

In this situation, a logical security role, manager, is given HTTP access permission to a Web resource collection, SalesInfo. Specifically, manager is allowed GET and POST HTTP requests to URLs with a context-relative virtual path that starts with the following:

```
/salesinfo/
```

In the auxiliary descriptor, the security-role-mapping element maps the manager logical role name to DBA, an existing user or group in the OSE security realm.

The auxiliary descriptor mapping is optional. Instead, you could create an OSE principal named manager. But the mapping functionality in the auxiliary descriptor provides additional flexibility, allowing a better fit for application security roles in a previously existing or general-purpose OSE security realm.

You can map an OSE principal, such as DBA, to multiple security roles.

Sample Auxiliary Descriptor

This section provides a sample Oracle auxiliary descriptor, based on the DTD in "Auxiliary Descriptor DTD" on page 8-17.

```
<?xml version="1.0"?>
<!DOCTYPE oracle-auxiliary-descriptor PUBLIC "-//Oracle Corporation//DTD Oracle
```

```
Auxiliary web Descriptor//1.0/EN" "otn.oracle.com">

<oracle-auxiliary-descriptor>
    <description>
        This is an example Oracle Auxilary Descriptor for
        Version 1.0, Dec. 2000
        webdomain for servlet context is given on command line as:
        $ deploywar -webdomain /serviceroot/10.1.1.20/cavist.com foo.war foo.xml
    </description>
    <context-descriptor debug="true" run-as-owner="false"
        browse-dirs="true" stateless="false" name="winecellar"
        virtual-path="/cellar" doc-root="/usr/htdocs/web-static-content">
        <description>
            The full JNDI context name is:
            /serviceroot/10.1.1.20/cavist.com/winecellar
        <default-info>
            <charset>
                ISO-8859-1
            </charset>
            <language>
                en-gb
            </language>
            <language>
                en
            </language>
        </default-info>
        <accept-info>
            <charset>
                ISO-8859-1
            </charset>
            <language>
                en
            </language>
        </accept-info>
    </context-descriptor>
    <log-descriptor>
        <description>
            error log is written to System.out
            event log - to database table "$OSE$ERROR_LOG$"
                        (HR schema, as specified)
            http access log - to table "$OSE$ACCESS_LOG$"
                        (HR schema, as inferred from jserver-loader)
            JNDI node in context "winecellar" will be called "logs/audit"
        </description>
        <error-log>
```

```
            <system-log/>
        </error-log>

        <event-log>
            <rdbms-log table="HR.$OSE$ERROR_LOG$"/>
        </event-log>

        <http-access-log name="logs/audit" >
            <rdbms-log table="$OSE$ACCESS_LOG$"/>
        </http-access-log>
</log-descriptor>

<jsp-info>
    <hotload all="false">
        <jsp>
            /cellar/jsp/wine_1.jsp
        </jsp>
        <jsp>
            /cellar/jsp/cell_1.jsp
        </jsp>
    </hotload>
</jsp-info>
<class-loader-descriptor>
    <jserver-loader>
        <schema>
            HR
        </schema>
        <resolver>
            ((* HR) (* PUBLIC) )
        </resolver>
    </jserver-loader>
</class-loader-descriptor>
<login-config>
    <description>
        This login-config will override the login-config in web.xml file.
        Valid authorization method values are BASIC or DIGEST.
    </description>
    <auth-method>
        BASIC
    </auth-method>
    <realm-name>
        wineRealm
    </realm-name>
</login-config>
<security-role-mapping>
```

```
                <security-role>
                    <description>
                        Mapping application's "manager" role to OSE HTTP Security
                    </description>
                    <role-name>
                        manager
                    </role-name>
                </security-role>
                <ose-principal>
                    DBA
                </ose-principal>
            </security-role-mapping>
        </oracle-auxiliary-descriptor>
```

Oracle WAR Deployment Tool Functionality

This section describes steps the Oracle WAR deployment tool takes in deploying an application, represented by a WAR file, to the Oracle9*i* database:

- Loading Files from the WAR File

- Creating a Servlet Context

- Publishing Servlets and JavaServer Pages

- Securing the Application

Any necessary WAR deployment tool parameter settings are noted. For information about how to invoke the deployment tool and set its input parameters, see "Oracle WAR Deployment Tool Usage" on page 8-43.

Loading Files from the WAR File

A WAR file can generally have an arbitrary directory structure, but a `WEB-INF` subdirectory is required. This subdirectory contains the `web.xml` file and, in further subdirectories, all Java classes and resources used in the application. There are many possible kinds of resources, but examples include external resource (`.res`) files for JSP pages, serialized resource (`.ser`) files for SQLJ JSP pages, and tag library descriptor (`.tld`) files for JSP tag libraries. (See the *Oracle JavaServer Pages Developer's Guide and Reference* for more information about `.res`, `.ser`, and `.tld` files.)

JSP pages themselves are outside the `WEB-INF` directory, and the Oracle WAR deployment tool assumes that any other files outside the `WEB-INF` directory are static documents to be served as is to any requests at runtime.

The remainder of this section gives more information about how the deployment tool loads deployment descriptors, Java files, and static files.

Deployment of Descriptor Files to the Oracle9*i* Database

The Oracle WAR deployment tool uses the Oracle `loadjava` tool to load the `web.xml` descriptor file and the Oracle auxiliary descriptor file into the database as Java resources. The `web.xml` file is extracted from the `WEB-INF` directory of the WAR file; the auxiliary descriptor is specified through a parameter of the session shell `deploywar` command, the deployment servlet, or the client-side script (as described in "Oracle WAR Deployment Tool Options and Parameters" on page 8-43).

The descriptors are loaded as follows, where *context_name* is the name of the servlet context that is created, and is taken from the `name` attribute of the `context-descriptor` element in the Oracle auxiliary descriptor.

■   The `web.xml` file is loaded as:

   context_name/WEB-INF/web.xml

■   The auxiliary descriptor file is loaded as:

   context_name/oracle_aux.xml

Deployment of Java Files to the Oracle9*i* Database

To run in the Oracle Servlet Engine, Java code must be loaded into the Oracle9*i* database. The Oracle WAR deployment tool examines the WAR file and uses the Oracle `loadjava` tool to load the following:

■   contents of the `WEB-INF` directory

   This includes Java class files under the `/WEB-INF/classes` subdirectory in the WAR file, and JAR files under the `/WEB-INF/lib` subdirectory.

■   JSP pages

If `loadjava` encounters problems, the WAR deployment tool will report them to you. Note that errors in individual Java objects do not prevent deployment of remaining objects. After deployment has finished, you have the option of correcting any errors and manually using `loadjava` to reload the Java objects that caused the errors. Alternatively, you can repackage the WAR file and redeploy the application.

Be aware of the following:

■   Use the `-replace` option of the Oracle WAR deployment tool when you want to redeploy an application that has been deployed previously (whether or not there were errors during the previous deployment).

■   Before invoking the Oracle WAR deployment tool, set the `schema` subelement under the `class-loader-descriptor` element in the Oracle auxiliary descriptor file. This is to specify the database schema where Java objects will be loaded. (Alternatively, you can let the deployment tool default to the schema of the user session from which you are invoking the deployment tool.)

■   The Oracle WAR deployment tool translates JSP pages (`.jsp` and `.sqljsp` files) in the WAR file, but will not translate or compile `.sqlj` or `.java` files. (The servlet 2.2 specification assumes that only compiled classes are used as

application code.) Any `.java` or `.sqlj` files in the WAR file will be treated as Java resources and loaded as is. If your initial WAR file does not conform to the specification, then you must manually perform any server-side translation or compilation and then repackage the WAR file.

Deployment of Static Files to the Document Root

Most Web applications, in addition to dynamic components such as servlets and JSP pages, use static pages (such as HTML pages) whose output does not change from request to request. WAR file contents that are not under the `WEB-INF` directory, with the exception of JSP source files, are assumed to be static pages. These files must be copied to the appropriate document root directory during OSE deployment.

For any Web application that will run in OSE and use OSE directly as the Web server, the Oracle WAR deployment tool will place static documents into the specified document root directory of the servlet context that corresponds to the application. The doc root is a directory in the file system of the machine on which the database (including its Oracle Servlet Engine) is installed.

Before invoking the deployment tool in situations where OSE will be used directly as the Web server, set the `doc-root` subelement under the `context-descriptor` element in the Oracle auxiliary descriptor file to specify the application doc root. If necessary, the deployment tool will create the file system directory and the `doc_root` object in the application's servlet context (once it has created the servlet context).

Regarding file permissions, you must verify the following:

- If the WAR deployment tool will have to create the doc root directory, ensure that the user (login schema) has `write` permission for the parent directory so that the deployment tool can write to it.

- Ensure that the user has `read` permission for the deployed WAR file and Oracle auxiliary descriptor.

You can address both of these issues through `dbms_java.grant_permission` calls (through SQL*Plus or equivalent) for `FilePermission` and `PropertyPermission`, as in the following examples:

```
call dbms_java.grant_permission
    ('HR', 'SYS:java.io.FilePermission',
     '/usr/htdocs/web-static-content/-', 'read/write');
call dbms_java.grant_permission
    ('HR', 'SYS:java.util.PropertyPermission', '*', 'read,write');
```

This example assumes that `HR` is the user name and `/usr/htdocs/web-static-content` is the parent directory. The directory name must be the full path of the parent directory, followed by a hyphen.

> **Note:** If the WAR file contains static files and you did not specify a doc root directory, or if the Oracle WAR deployment tool cannot create or write into the specified doc root directory, then a warning message will be issued. The rest of the application deployment will proceed.

Using a Front-End Web Server In many circumstances, OSE is not used directly as the Web server. Instead, a front-end Web server such as the Oracle HTTP Server powered by Apache is used. In this case there may be no need to specify a doc root directory for deployment, but you must manually copy your static files to the doc root of the front-end Web server. (The Oracle auxiliary descriptor `doc-root` subelement under the `context-descriptor` element is optional.)

> **Note:** A doc root is still required, however, for any request dispatcher functionality such as dynamic `include` or `forward` commands.

Creating a Servlet Context

As described in "Web Application Servlet Contexts" on page 8-2, a Web application is represented by a servlet context, which acts as an application container. A servlet context is represented in Java as an instance of a class that implements the standard `javax.servlet.ServletContext` interface. The Oracle WAR deployment tool will create a servlet context during deployment of your application and will define initial servlet context parameters according to `context-param` settings, if this element is present in the `web.xml` file.

When deploying the application, you must specify the servlet context name and the URI path or paths you want mapped to the context (and therefore, to the application). To provide this information, set the following attributes of the `context-descriptor` element in the Oracle auxiliary descriptor file:

- `name`
- `virtual-path`

When you invoke the Oracle WAR deployment tool, you must specify the domain where you want the deployment tool to place the context (for example, through the `-webdomain` parameter in the session shell `deploywar` command or client-side script).

(Refer to the *Oracle9i Oracle Servlet Engine User's Guide* for background information about domains, context names, and virtual paths.)

Notes:

- Another aspect of creating a servlet context for an application is specifying a file system directory for the doc root (the target directory for loading static files). This was described earlier, in "Deployment of Static Files to the Document Root" on page 8-37.

- If you are redeploying an application that had previously been deployed, enable the `-replace` option. This informs the WAR deployment tool that you are willing to re-create the servlet context for the application. (See "Oracle WAR Deployment Tool Options and Parameters" on page 8-43.)

Publishing Servlets and JavaServer Pages

When you deploy an application to the Oracle9*i* database, the Oracle WAR deployment tool will publish any servlets and JSP pages in the application. This process makes them available for execution in the Oracle Servlet Engine. (Refer to the *Oracle9i Oracle Servlet Engine User's Guide* for background information about publishing.)

The remainder of this section describes the publishing process for servlets and JSP pages, and any necessary preparations.

Preparation for Servlets

For an application to follow the servlet 2.2 specification, any class that is to be accessed as a servlet must have a corresponding `servlet` element, with `servlet-name` and `servlet-class` subelements, in the `web.xml` file. If no `servlet` element is present, then the Oracle WAR deployment tool will load the servlet class into the database, but will not publish it as a servlet in the application servlet context.

If a servlet will be accessed through one or more virtual paths within the servlet context, then the `web.xml` file must also contain a `servlet-mapping` element for each virtual path.

Implicit Deployment of JavaServer Pages

According to the servlet 2.2 specification, a JSP page would have a `web.xml` `servlet` element with a `jsp-file` subelement instead of a `servlet-class` subelement. However, use of the `servlet` element is not required for JSP pages in Oracle WAR deployment.

If the Oracle WAR deployment tool encounters a `.jsp` file or `.sqljsp` file outside the `WEB-INF` directory, and the page is not explicitly listed in the `web.xml` file, the page will still be translated, compiled, and published. In this case, however, a user would invoke the page with a servlet path that is determined by the context-relative path of the file in the WAR file hierarchy.

Consider the following example.

- The servlet context for an application is accessible by the following virtual path:

  `/mycontext`

- A JSP page in the application, `imajsp.jsp`, is located in `jsp/imajsp.jsp` in the WAR file. This makes the following its context-relative path:

  `jsp/imajsp.jsp`

Therefore, a user would invoke the page as follows:

`http://`*host*`[:`*port*`]/mycontext/jsp/imajsp.jsp`

Notes:

- If a JSP page is encountered *inside* the `WEB-INF` directory, then it is loaded into the Oracle9*i* database as a Java resource. It is not translated, compiled, or published.

- You cannot install a JSP page into the doc root manually and expect it to be served to clients. It would not be translated or published, and attempts to execute it would expose its source code. To add a JSP page after the WAR file has been deployed, you must explicitly publish it. (Use the session shell `publishjsp` command.)

Translation, Compilation, and Publishing of JavaServer Pages

The Oracle WAR deployment tool performs the following steps for each `.jsp` or `.sqljsp` file that it deploys into the Oracle9*i* database:

- It invokes the OracleJSP translator to translate the page. The page implementation class produced by the translator is essentially a servlet class. All errors and warnings produced by the OracleJSP translator are reported.

- It compiles the resulting JSP page implementation servlet and any inner classes that were generated during JSP translation, and loads them into the application schema. The Java class name and package of the page implementation servlet class will be derived from the name and the directory path of the JSP file as it appears in the WAR file.

    For example, a directory path of `myroot/myjsp` will result in the following package:

    `_myroot._myjsp`

    (For more information about package naming by the OracleJSP translator, including its use of leading underscores, see the *Oracle JavaServer Pages Developer's Guide and Reference.*)

- It maps the servlet to as many virtual paths as there are corresponding `servlet-mapping` elements in the `web.xml` file (if any).

- It publishes the page implementation servlet. This servlet is given a JNDI name as follows:

    – If there is a `servlet` element for the JSP page in the `web.xml` file, then the JNDI name is the same as the `servlet-name` subelement of the `servlet` element.

    – If there is no `servlet` element, then the servlet name is determined by the path of the JSP page in the WAR file. First, `implicit_` is prepended, then each slash ("/") in the path is replaced by an underscore ("_"). For example, `myroot/myjsp/page1.jsp` would have the following JNDI name:

    `implicit_myroot_myjsp_page1.jsp`

Note: The servlet name (JNDI name) is used in the implicit OSE `publishservlet` command that is executed by the WAR deployment tool.

Securing the Application

To protect a virtual path, the Oracle WAR deployment tool essentially uses the same sequence of session shell commands that you would use manually. For example, to allow users in the group catalogBuilder in a realm CatalogRealm to perform POST operations to the relative virtual path publishedCatalogs/* (after they authenticate themselves through the basic authentication method), the deployment tool issues the following session shell commands:

```
$realm map -s <servletContextPath> -add publishedCatalogs/*
   -scheme basic:CatalogRealm
$realm perm -w <webserviceRoot> -realm CatalogRealm -s <servletContextPath>
   -name catalogBuilder -path publishedCatalogs/* + POST
```

Any messages printed by the realm commands are output by the WAR deployment tool. For example, the following messages indicate that the two realm commands above succeeded:

```
mapping 'publishedCatalogs/*' with scheme 'basic:CatalogRealm' added
permissions post granted on publishedCatalogs/* for catalogBuilder
```

If, however, you did not create a user or a group named catalogBuilder in the realm CatalogRealm prior to deployment, you will see the following error message:

```
no such user
```

> **Note:** For information about preliminary steps you must take before establishing application security, see "Security Preparations" on page 8-10.

Oracle WAR Deployment Tool Usage

This section describes the Oracle WAR deployment tool, introduced in "Overview of the Oracle WAR Deployment Tool" on page 8-8. The deployment tool takes an application WAR file and Oracle auxiliary descriptor and deploys the application into the Oracle9*i* database accordingly. You can then execute the application in the Oracle Servlet Engine.

The following subsections describe the different ways to invoke the deployment tool, and document the options and input parameters that the deployment tool supports:

- Oracle WAR Deployment Tool Options and Parameters
- Vehicles for Invoking the Oracle WAR Deployment Tool

Oracle WAR Deployment Tool Options and Parameters

Each of the vehicles for invoking the Oracle WAR deployment tool supports the same set of key options and parameters, and the deployment servlet and client-side script support additional parameters as appropriate.

Table 8–1 summarizes these options and parameters. (The session shell WAR deployment command, deployment servlet, and client-side script are described in "Vehicles for Invoking the Oracle WAR Deployment Tool" on page 8-46.)

Table 8–1   Oracle WAR Deployment Tool Options and Parameters

Functionality	Session Shell Command Parameter	Deployment Servlet Parameter	Client-Side Script Parameter
WAR file name	(enter file name on command line)	`warfile`	`-warfile` (or `-wa`)
Oracle auxiliary descriptor file name	(enter file name on command line)	`auxdescriptor`	`-auxdescriptor` (or `-a`)
Web domain name (where the application's servlet context will be created)	`-webdomain` (or `-w`)	`webdomain`	`-webdomain` (or `-w`)
Replacement of a previously deployed application (**disabled by default**)	`-replace` (or `-r`)	`replace`	`-replace` (or `-r`)
Verbose output from tool (**disabled by default**)	`-verbose` (or `-ver`)	`verbose`	`-verbose` (or `-v`)

Table 8–1 Oracle WAR Deployment Tool Options and Parameters (Cont.)

Functionality	Session Shell Command Parameter	Deployment Servlet Parameter	Client-Side Script Parameter
XML validation for `web.xml` (recommended; **enabled by default**)	on by default; `-noxmlvalidate` or `-nox` to disable	`xmlvalidate` (on by default; set to `false` to disable)	`-xmlvalidate` (on by default; `-noxmlvalidate` or `-nox` to disable)
Upload directory for files being loaded (WAR file and Oracle auxiliary descriptor)	n/a	`upload`	`-upload` (or `-up`)
URL of deployment servlet	n/a	n/a	`-target` (or `-t`)
Schema (and optionally password) where WAR file and auxiliary descriptor will be deployed	n/a to `deploywar` command (the schema is specified when the session shell is started)	n/a to deployment servlet (inferred from the schema used in executing the servlet)	`-user` (or `-u`)
Password for target schema (if not supplied with user name)	n/a	n/a	`-password` (or `-p`)
Help (brief usage notes)	`-help`	n/a	`-help` (or `-h` or `-?`)

Notes:

- The session shell command parameters also apply if you invoke the WAR deployment tool through server-side Java code or a PL/SQL call specification.

- The client-side script parameters also apply when you invoke the client-side deployment tool wrapper directly from Java.

- The deployment servlet parameters also apply when you invoke the `deploywar.htm` HTML form.

General Option/Parameter Notes:

WAR file and Oracle auxiliary descriptor	If you are using the session shell `deploywar` command, server-side Java, or a PL/SQL call specification to invoke the Oracle WAR deployment tool, it is assumed that you have already manually uploaded the WAR file and Oracle auxiliary descriptor into an accessible location in the file system of the target system.
"Webdomain" option	This is always required to specify the (existing) Web domain where the WAR deployment tool is to create the servlet context for the application being deployed. In an OSE single-domain scenario, this would simply be a name such as the following:

```
/testRoot
```

In an OSE multi-domain scenario, however, you must include the IP address and DNS name such as in the following example:

```
/testRoot/10.1.1.20/cavist.com
```

You must always specify the absolute domain name (starting with "/"), regardless of your current directory in the session shell.

"Replace" option	If an application has already been deployed, even if it deployed with errors, OSE will not allow creation of a servlet context again unless you enable this option.
"Verbose" option	Enabling this option significantly increases the amount of output (especially from the `loadjava` tool) during deployment, but is useful if you are trying to troubleshoot deployment errors.

XML validation (can be disabled for `web.xml`)	You can disable XML validation for `web.xml`, but this is generally advisable only if you are migrating an application from a servlet container that did not validate XML, and you cannot immediately correct validation errors. XML validation verifies the `web.xml` file against the DTD specified in the servlet 2.2 specification, and is enabled by default. The Oracle auxiliary file is always validated (this cannot be disabled).
	(Severe errors in the `web.xml` file may prevent deployment even if XML validation is disabled.)
	This option may not be supported in future releases.
Upload directory	The Oracle WAR deployment tool must have `write` permission for this directory.
User and password	These are not applicable for the session shell `deploywar` command, where the user was already established during session shell login, or for the deployment servlet, where any security filtering would have occurred before the servlet was invoked.
General comments about client-side script parameters	True/false parameters are optional; all other parameters are mandatory and must be explicitly set.
	You have choices in how to format your command-line option settings for the client-side scripts (such as whether they are preceded by a hyphen). See "Oracle Client-Side Deployment Scripts (Oracle Client)" on page 8-50 for information.

Vehicles for Invoking the Oracle WAR Deployment Tool

This section describes each of the ways to invoke the Oracle WAR deployment tool, providing details about option settings and command-line syntax as applicable. The following topics are covered:

- WAR Deployment Tool Session Shell Command

- WAR Deployment Tool Invocation from Server-Side Java or PL/SQL Call Spec

- Oracle WAR Deployment Servlet

- Oracle Client-Side Deployment Scripts (Oracle Client)

- Oracle Client-Side WAR Deployment Tool Wrapper (Non-Oracle Client)

When using the session shell command or invoking the WAR deployment tool from server-side Java or a PL/SQL call specification, you must first manually place the WAR and auxiliary descriptor files onto the file system of the target server. Loading these files is generally done for you if you deploy through one of the client-side vehicles that Oracle provides.

WAR Deployment Tool Session Shell Command

From the Oracle9*i* session shell, you can use the `deploywar` command to deploy a Web application packaged in a WAR file. Here is the command-line syntax (this is a single wrap-around command line; $ is the session shell prompt):

```
$ deploywar [-replace|-r] [-verbose|-ver] [-noxmlvalidate|-nox]
-webdomain|-w <domain_name> <WAR-file-name> <Oracle-aux-descriptor>
```

Notes:

- You must manually upload the WAR and auxiliary descriptor files onto the file system of the target machine before executing the `deploywar` command.

- Parameter values for the `deploywar` command are case-sensitive.

The `-webdomain` option, WAR file name, and Oracle auxiliary descriptor file name are mandatory arguments.

The WAR and auxiliary descriptor file names must include full paths to the files and must be specified on the command line in the positions shown in the example.

The `-replace`, `-verbose`, and `-noxmlvalidate` settings are optional. The order of these three options and the `-webdomain` option amongst themselves is not significant.

For general information about deployment options and parameters, see "Oracle WAR Deployment Tool Options and Parameters" on page 8-43.

For general information about the Oracle9*i* session shell and how to run it, see the *Oracle9i Java Tools Reference.*

WAR Deployment Tool Invocation from Server-Side Java or PL/SQL Call Spec

To invoke the Oracle WAR deployment tool from server-side Java code or through a PL/SQL call specification, you can use the following public static method:

```
void oracle.mts.http.deployment.DeployWar.main(String[] args)
```

Arguments are the same as for the session shell deploywar command. See "WAR Deployment Tool Session Shell Command" on page 8-47.

> **Note:** As with the deploywar session shell command, first you must manually upload the WAR and auxiliary descriptor files onto the target machine.

Oracle WAR Deployment Servlet

Provided with the Oracle WAR deployment tool is a servlet that you can use to start deployment of a WAR file to the Oracle9*i* database from a client.

This deployment servlet is preloaded in the database SYS schema and published in the admin domain of the Oracle Servlet Engine, under the /deploywar virtual path. Alternatively, you can publish it in any other desired context with a virtual path of your choice.

On the server, the Java class for the servlet is as follows:

```
oracle.aurora.mts.http.deployment.DeploymentServlet
```

For convenience, Oracle also provides a form called deploywar.htm that you can use to invoke the servlet from a client. In Oracle9*i* releases, this form is in the doc root, [ORACLE_HOME]/jis/public_html, of the default context of the OSE admin service. For the Oracle8*i* 8.1.7.0 and 8.1.7.1 releases, it is in the deploywar.jar and packager.jar files that are included with the Oracle Technology Network download.

Once deploywar.htm is placed in an appropriate doc root directory, access it from a browser, supplying settings for the following parameters when prompted:

- replace=true or false

- verbose=true or false

- `xmlvalidate=true` or `false`

- `uploaddir=<target dir for WAR file and aux descriptor>`

- `webdomain=<Web domain for servlet context>`

- `warfile=<WAR file name>`

- `auxdescriptor=<Oracle auxiliary descriptor file name>`

The `deploywar.htm` form prompts you for all of these, but only `uploaddir`, `webdomain`, `warfile`, and `auxdescriptor` are required. The `replace` parameter defaults to `false`, `verbose` defaults to `false`, and `xmlvalidate` defaults to `true`.

For general information about these options and parameters, see "Oracle WAR Deployment Tool Options and Parameters" on page 8-43.

Notes:

- Parameter values for the deployment servlet are *not* case-sensitive.

- If you want to access the deployment servlet through your own HTTP client, you can do so through the `doPost()` method of the `DeploymentServlet` class. You must send multipart-encoded form data containing the parameter settings in standard format, separated by standard HTTP boundaries.

DeploymentServlet Permissions The remainder of this section discusses considerations in ensuring that the deployment servlet will have sufficient permissions when it executes.

When you deploy your application, its servlet context is created in the domain that you specified through the `webdomain` parameter. You must ensure that the database session executing the deployment servlet has sufficient JNDI permissions to browse the domain in question, and to create new JNDI objects in that domain.

One possible way to handle these permission and ownership issues would be to publish the deployment servlet in every domain where you anticipate executing it (publishing it in the default context of each domain, for example).

However, this liberal approach may not fit your security model. You may find it desirable to keep tighter control over application deployment and to use only the copy of the deployment servlet that is automatically published in the `admin`

domain. This copy of the deployment servlet is owned by the `SYS` schema, which can publish your application in any domain and also has the broadest file creation permissions.

The danger of using the `SYS` schema, however, is that the servlet context created for your application during deployment will therefore be owned by `SYS`. This fact, combined with setting the `run-as-owner` property of the `context-descriptor` element to `true` in the auxiliary descriptor, would give the application unlimited access to the database. However, you can prevent this potential security breach by using the session shell `chown` command to change ownership of the servlet context immediately after application deployment.

Also be aware that in the current release, you cannot control the database session owner by means of HTTP security authentication. Although you can protect access to the deployment servlet by using the HTTP security `realm` command of the session shell and requiring a user to be authenticated before invoking the deployment servlet, be aware that the user being authenticated for HTTP access comes from an OSE HTTP security domain. When the user is authenticated successfully and is found to have permissions to the servlet, servlet execution is allowed to start. In general, however, this has no relation to the database schema that actually executes the servlet.

Currently, you can control which database users can execute the deployment servlet only as follows:

- through ownership of the domain of the deployment servlet

- through ownership of the servlet context of the deployment servlet, in combination with the `run-as-owner` context property

- by making the deployment servlet class execute with definer's rights, which would be `SYS`

Oracle Client-Side Deployment Scripts (Oracle Client)

For deploying your application from a system with an Oracle client installation, an alternative to invoking the deployment servlet from the `deploywar.htm` form or an HTTP client is to use a client-side deployment script that Oracle provides and which uploads the WAR and auxiliary descriptor files for you.

> **Important:** The client-side deployment script requires an Oracle client installation. For a non-Oracle client, see "Oracle Client-Side WAR Deployment Tool Wrapper (Non-Oracle Client)" on page 8-54.

On Solaris, run the following script (`%` is the UNIX prompt):

```
% deploywar <parameters>
```

On Windows NT, run the following batch file from a DOS prompt or from the `Run` command line in Windows:

```
deploywar.bat <parameters>
```

Running either of these scripts will invoke the Oracle deployment servlet, which is preloaded in the Oracle9*i* database and published in the `SYS` schema of the `admin` domain. (Or the deployment servlet, along with the rest of the Oracle WAR deployment software, can be added manually to an Oracle8*i* 8.1.7 database. Refer to the Oracle WAR deployment README file.)

> **Note:**   Parameter values for the client-side deployment script are *not* case-sensitive.

Formats for Setting Options/Parameters   You can set parameters for the client-side deployment scripts in a variety of ways.

The simplest way to set a true/false parameter is to precede its name with "-" to enable it or with "-no" to disable it, such as in the following examples:

```
-replace -verbose -noxmlvalidate
```

or, to use the accepted shortcuts:

```
-r -v -nox
```

These settings enable the `-replace` option and the `-verbose` option (which are disabled by default) and disable the `-xmlvalidate` option (which is enabled by default).

Set other options (those other than true/false) using an equals sign (=) as follows:

```
-target=http://www.acme.com:8000/DeploymentServlet
-user=HR -password=hr -upload=/tmp
-webdomain=/testRoot
```

or, to use the accepted shortcuts:

```
-t=http://www.acme.com:8000/DeploymentServlet
-u=HR -p=hr -up=/tmp
-w=/testRoot
```

For help, use -help, -h, or -? to get the deployment script parameter list.

> **Notes:**   For parameters other than true/false parameters, you can optionally omit the "-" or replace the "=" with a space, *but not both*, such as in the following examples:
>
> ```
> -user HR
> -u HR
> user=HR
> u=HR
> ```
>
> You can explicitly set true/false parameters to true or false, optionally omitting the "-", as in the following examples:
>
> ```
> -replace=true
> -r=true
> replace=true
> r=true
>
> -xmlvalidate=false
> -x=false
> xmlvalidate=false
> x=false
> ```
>
> You must always use "=" (you cannot use a space) when setting true/false parameters explicitly.

Setting the User Name and Password As a shortcut, you can set the user name and password simultaneously through the user option, by placing a forward-slash ("/") between the settings. In this case you do not need to use the password option.

All of the following are equivalent:

```
-user=HR/hr
```

or:

```
-user=HR -password=hr
```

or:

```
-user HR/hr
```

or:

```
-user HR -password hr
```

or:

```
user=HR/hr
```

or:

```
user=HR password=hr
```

Specifying the WAR file and Auxiliary Deployment Descriptor File names with the default file name extension are recognized by the client-side deployment script as being WAR files or auxiliary descriptors, in which case you can enter the file names on the command line without the `warfile` or `auxdescriptor` parameter name:

- The `.war` file name extension indicates a WAR file.

- The `.xml` file name extension indicates an Oracle auxiliary descriptor file.

Sample Deployment Script Command Line Following is an example of running the client-side deployment script from a Solaris system:

```
% deploywar
    t=http://localhost:8080/DeploymentServlet
    u=HR/hr
    up=/tmp
    -r -v -nox
    w=/testRoot
    myTestApp.war
    /somedir/aux.xml
```

This example accomplishes the following:

- It accesses the deployment servlet on the local host in the schema HR with password hr.

- The `-up` (upload) setting causes the WAR file and Oracle auxiliary descriptor to be uploaded to the server's `/tmp` directory.

- The `-r` and `-v` settings enable the `-replace` and `-verbose` options.

- The `-nox` setting disables XML validation of the `web.xml` file.

- The `-w` option specifies that the servlet context for the application is to be created in the `testRoot` domain.

- Because the Oracle auxiliary descriptor (`aux.xml`) and the WAR file
  (`myTestApp.war`) have default file name extensions, the `warfile` and
  `auxdescriptor` parameter names are not required.

Oracle Client-Side WAR Deployment Tool Wrapper (Non-Oracle Client)

On a system without an Oracle client installation, you can do the equivalent of
running the client-side deployment script by executing the client-side WAR
deployment tool wrapper directly from Java, as follows.

UNIX (this is a single wrap-around command):

```
java -classpath $CLASSPATH
oracle.aurora.mts.http.deployment.client.HttpClientWrapper <parameters>
```

Windows NT (this is a single wrap-around command):

```
java -classpath %CLASSPATH%
oracle.aurora.mts.http.deployment.client.HttpClientWrapper <parameters>
```

Specify the WAR file, auxiliary descriptor, and other input parameters just as you
would for the client-side deployment scripts, as described above in "Oracle
Client-Side Deployment Scripts (Oracle Client)" on page 8-50.

Note: The Oracle client-side deployment files, `packager.jar`
and `http_client.jar`, must be in your classpath. You can obtain
these files from the Oracle Technology Network or from a product
CD for the Oracle9*i* database or Oracle9*i* Application Server.

Sample Application Hierarchy and Descriptor Files

This section provides the hierarchy, `web.xml` file, and Oracle auxiliary descriptor for a sample application. The following topics are covered:

- Sample Hierarchy
- Sample Descriptor Files
- Creating and Deploying the WAR File

Sample Hierarchy

Consider a sample Web application with the following hierarchy from the root directory:

```
index.html
WEB-INF/
WEB-INF/classes/
WEB-INF/classes/BasicQuery.class
WEB-INF/lib/
WEB-INF/lib/ForTag.jar
WEB-INF/web.xml
images/
images/mailbox.gif
jsp/
jsp/tag/
jsp/tag/tagexample.jsp
jsp/tag/exampletag.tld
jsp/welcome.jsp
```

The static file `index.html` is at the top of the hierarchy in the doc root. There is an `images` directory for the one graphic (`mailbox.gif`).

In accordance with the servlet 2.2 specification, the `WEB-INF` directory contains the `web.xml` file, a `classes` subdirectory for compiled Java class files (only `BasicQuery.class` in this case), and a `lib` subdirectory for compressed JAR or ZIP files with utility libraries (`ForTag.jar` in this case).

The `jsp` directory contains a JSP welcome page, and the tag subdirectory has a JSP tag example page and the tag library descriptor (TLD file) for the tag library. JSP pages will be translated by the Oracle WAR deployment tool during deployment.

Sample Descriptor Files

This section provides the sample `web.xml` file and Oracle auxiliary descriptor.

Sample web.xml Deployment Descriptor

Following is the `web.xml` deployment descriptor file for the sample application.
Note the `servlet` and `servlet-mapping` elements for the `BasicQuery` servlet
and the `welcome.jsp` and `tagexample.jsp` JSP pages. For JSP pages, the
`servlet` element has a `jsp-file` subelement instead of a `servlet-class`
subelement.

This `web.xml` file also specifies possible welcome pages and an error page.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <distributable/>
    <servlet>
      <servlet-name>exampletag</servlet-name>
      <jsp-file>jsp/tag/tagexample.jsp</jsp-file>
    </servlet>
    <servlet>
      <servlet-name>Basic</servlet-name>
      <servlet-class>BasicQuery</servlet-class>
    </servlet>
    <servlet>
      <servlet-name>welcome</servlet-name>
      <jsp-file>jsp/welcome.jsp</jsp-file>
    </servlet>
    <servlet-mapping>
      <servlet-name>exampletag</servlet-name>
      <url-pattern>/jsp/tag/tagexample.jsp</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>Basic</servlet-name>
      <url-pattern>/basic</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>welcome</servlet-name>
      <url-pattern>jsp/welcome.jsp</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>
            index.html
        </welcome-file>
```

```
        </welcome-file-list>
</web-app>
```

Sample Oracle Auxiliary Descriptor

Following is the Oracle auxiliary descriptor file for the sample application. This file specifies a servlet context, MyCtx, in an OSE domain of your choosing, with /myapp as its virtual path. The doc root will be the context-relative /tmp/myapp_docroot directory.

The application will be deployed into the HR schema, using the default resolver of that schema (because no resolver is specified).

```
<?xml version="1.0"?>
<oracle-auxiliary-descriptor>
    <description>
        This is Oracle Auxilary Descriptor for
        application MyApp. You may deploy in any Domain you have.
    </description>
    <context-descriptor name="MyCtx"
        virtual-path="/myapp"
        doc-root="/tmp/myapp_docroot">
         <description>
            The full context name is "Domain/contexts/MyCtx".
         </description>
    </context-descriptor>
    <class-loader-descriptor>
        <jserver-loader>
            <schema>
                HR
            </schema>
        </jserver-loader>
    </class-loader-descriptor>
</oracle-auxiliary-descriptor>
```

Creating and Deploying the WAR File

Assuming the application component files are in the directory structure indicated by the WAR file example above, and you run the JAR utility from the application root directory, you can create a WAR file named myapp.war with the following command:

```
jar cvf myapp.war index.html WEB-INF/ images/ jsp/
```

The WAR file will have an internal structure that reflects the application hierarchy (see "Sample Hierarchy" on page 8-55).

Using the client-side deployment script (as an example), the following operating system command will deploy the WAR file, `myapp.war`, and the Oracle auxiliary descriptor, `MyAppAux.xml`:

```
% deploywar target=http://host[:8080]/admin/deploywar user=HR/hr  \
   upload=/tmp webdomain=/MyDomain warfile=myapp.war auxdescriptor=MyAppAux.xml
```

> **Note:** By default, the deployment servlet is automatically installed in the `admin` domain, using port 8080.

Or, using parameter name shortcuts and dropping `warfile` and `auxdescriptor` (allowable because the WAR file and auxiliary descriptor use default file name extensions):

```
% deploywar t=http://host[:8080]/admin/deploywar u=HR/hr up=/tmp w=/MyDomain
myapp.war MyAppAux.xml
```

Or, to enable the `-replace` option (so the servlet context can be re-created if the application has previously been deployed) and the `-verbose` option (for informative status output from the Oracle WAR deployment tool and the other tools that it calls):

```
% deploywar t=http://host[:8080]/admin/deploywar u=HR/hr up=/tmp w=/MyDomain -r
-v myapp.war MyAppAux.xml
```

Current Restrictions

Some features are not yet implemented in the current release of Oracle WAR deployment. Following is a list of the current restrictions.

- The `web.xml load-on-startup` element has no effect; OSE does not pre-start servlets. If this element is encountered, the Oracle WAR deployment tool will print a warning message with the order number that was specified in the `web.xml` file.

- Generally, error-page mapping for an application may be defined in two ways: 1) HTTP error-code mapping to error-page location; or 2) Java exception-type mapping to the error-page location. OSE, however, does not currently support the latter. Such a mapping in the `web.xml` file will result in a warning message and will otherwise be ignored.

- OSE does not currently support the `web.xml env-entry` and `resource-ref` elements. Attempts to use these elements will result in a warning message and will otherwise be ignored.

- OSE does not currently support the `web.xml ejb-ref` element. Attempts to use this element will result in a warning message and will otherwise be ignored. It is possible, however, to refer to an EJB from a servlet by explicitly using its `Home` and `Remote` interfaces.

- OSE does not currently support the `web.xml security-role-ref` subelement of the `servlet` element. Attempts to use this subelement will result in a warning message listing names and role links of all security role references found in a servlet definition. If any of these reference names are used in the servlet code, the code must be modified to work without them.

- JSP pages cannot use any `taglib` directive that specifies a tag library descriptor (`.tld` file) in a JAR file. A `taglib` directive must specify a `.tld` file directly. This is because when a JAR file is included in a WAR file, the Oracle WAR deployment tool will distribute the contents according to the WAR file hierarchy. The JAR file would no longer be in its original form, and OracleJSP would no longer be able to find the `.tld` file that had formerly been embedded in the JAR file.

  The same restriction exists when the JAR file is published using the session shell `publishjsp` command.

- OSE does not currently support transport guarantee directives, defined in the `user-data-constraint` subelement of the `web.xml` `security-constraint` element.

- OSE does not currently support form-based authentication; therefore, it does not support the FORM value for the auth-method subelement or form-login-config subelement of the login-config descriptor element.

- OSE does not currently support the CLIENT-CERT value for the auth-method subelement of the login-config descriptor element.

9

Writing PL/SQL Servlets

This chapter describes new and changed features for Oracle8*i* Release and Oracle9*i*. The topics in this chapter include:

- Overview of PL/SQL Servlets
- Configuring Database Access Descriptors from an Application
- Package DBMS_EPGC

Overview of PL/SQL Servlets

When you use the Internet Application Server (*i*AS), you typically access PL/SQL stored procedures over the Web by using the `mod_plsql` module. This module is recommended for stateless PL/SQL procedures, where the transaction state and values of package variables are not preserved once the original procedure call is finished.

You can also run PL/SQL stored procedures using the Oracle Servlet Engine through the `mod_ose` module of *i*AS. Oracle recommends this module for stateful PL/SQL procedures, which behave similar to Java servlets. These procedures can preserve state (such as package variables and transaction state) across multiple HTTP requests.

For detailed information about running PL/SQL procedures over the Web, see the documentation about using `mod_plsql` in the Oracle HTTP Server documentation.

Configuring mod_ose to Run PL/SQL Servlets

To run PL/SQL stored procedures as servlets, you must first load and publish one servlet that serves as a gateway (known as the embedded PL/SQL gateway). This is a one-time operation. The PL/SQL procedures can then run over the Web without any code changes or loading/publishing steps for each procedure.

From SQL*Plus, connect as `SYS` and run the script `rdbms/admin/initplgs.sql` to load the embedded PL/SQL gateway servlet into the database server.

From the system command line, use the session shell `publishservlet` command to publish the servlet so that it can be accessed through a URL. This operation registers a virtual path, and every request for a document using that virtual path is handled by the embedded PL/SQL gateway servlet, which runs the appropriate PL/SQL stored procedure. For example:

```
% $ORACLE_HOME/jis/bin/unix/sess_sh -s http://webserver:portnumber -u \
sys/change_on_install
--Session Shell--
--type "help" at the command line for help message
$ publishservlet -virtualpath pls/* /webdomains/contexts/default plsGateway \
   SYS:oracle.plsql.web.PLSQLGatewayServlet
```

This publishes the gateway servlet under the name `plsGateway` with a default context. In this example:

- You can access PL/SQL stored procedures using the virtual path `/pls`. You might specify different virtual paths to set up multiple instances of the servlet, each with different settings.

- You can choose a different name in place of `plsGateway`. This is the name that you use when forwarding requests from another servlet.

- You must specify the `SYS:` parameter as shown. It is the name of the actual Java class file.

A URL to access a stored procedure through the gateway might look like one of these:

```
http://webserver/pls/dadname/procedurename
http://webserver/pls/dadname/schemaname.procedurename
http://webserver:portnumber/pls/dadname/procedurename
http://webserver/pls/dadname/procedurename?param1=value1&param2=value2
```

The procedure names in these URLs specify PL/SQL procedures. They can use either a stateful or a stateless execution model, depending on how you configure the DAD, as explained in the following section.

> **See Also:**
>
> - *Using mod_plsql* in the documentation for the Oracle HTTP Server for information about the DAD configuration parameters.
>
> - *Oracle9i Java Tools Reference* for the syntax of the session shell commands.

Writing Stateful PL/SQL Stored Procedures

Typically, when you run a PL/SQL stored procedure over the Web, its state goes away when the procedure ends. This state information includes the values of any package variables it accesses, its transaction state, and any rows it inserted into temporary tables.

You might want to change this behavior when several procedures are called in sequence, for example during a registration procedure that uses several different HTML forms. Instead of passing the information from one procedure to another using CGI-style parameters, you can store it in package variables until the entire process is complete. You can do a single commit or rollback when the registration succeeds or fails.

You can preserve this state information across calls to PL/SQL stored procedures by following these steps:

1. Publish the embedded PL/SQL gateway servlet through the Oracle Servlet Engine, as previously described. This only needs to be done once.

2. Set the `stateful` attribute of the DAD to `Yes`. By default, its value is `No`. This only needs to be done once, and remains in effect for all packages and stored procedures called through this DAD. You can also set this attribute at the global level, so that all new DADs inherit the same setting.

3. Create a package containing some variables, if you need storage for data to be preserved across calls.

4. Write one or more PL/SQL stored procedures that access the package variables, perform different parts of a single transaction, and generally take advantage of stateful execution.

5. When all the data is ready, explicitly commit if the operation is successful, or rollback if the operation fails. There is no implicit commit when the procedure ends. When an exception is raised, there is an implicit rollback to the state at the beginning of the current procedure call, but the transaction remains open so a commit or rollback is still needed at the end.

To explicitly delete the package state information, you can call `DBMS_SESSION.RESET_PACKAGE`. This technique lets you get the performance benefits of stateful procedures while keeping the default behavior for state information.

Configuring Database Access Descriptors from an Application

When you configure a Web server to run Oracle stored procedures, you typically use a browser interface to set up the database access descriptor (DAD). To automate this operation, you can configure the DAD by calling procedures in the package DBMS_EPGC. The following example shows how to set or change the DAD configuration from an application. The next section describes each procedure in package DBMS_EPGC.

```
--
-- A sample procedure that configures an embedded gateway
-- instance for the given port number.
--
CREATE OR REPLACE PROCEDURE sample1_init_cfg(port IN PLS_INTEGER) IS
BEGIN

   --
   -- reset instance (port) configuration.
   --
   dbms_epgc.drop_instance(port);
   dbms_epgc.create_instance(port);


   --
   -- set global attributes for the embedded PL/SQL Gateway instance.
   --
   dbms_epgc.set_global_attribute(port, 'defaultdad', 'HR');
   dbms_epgc.set_global_attribute(port, 'adminPath', '/admin_/');
   dbms_epgc.set_global_attribute(port, 'stateful', 'Yes');



   --
   -- create a database access descriptor (DAD) called APPS
   --
   dbms_epgc.create_dad(port, 'APPS');
   dbms_epgc.set_dad_attribute(port, 'APPS', 'default_page', 'APPS.pkg.home');
   dbms_epgc.set_dad_attribute(port, 'APPS', 'document_table', 'APPS.doc_tab');
   dbms_epgc.set_dad_attribute(port, 'APPS', 'document_path', 'docs');
   dbms_epgc.set_dad_attribute(port, 'APPS', 'upload_as_blob', 'jpeg, gif,
txt');
   dbms_epgc.set_dad_attribute(port, 'APPS', 'document_proc',
                               'APPS.doc_pkg.process_download');
   -- override global setting for stateful attribute
   dbms_epgc.set_dad_attribute(port, 'APPS', 'stateful', 'No');
```

```
      --
      -- create a database access descriptor (DAD) called HR.
      --
      dbms_epgc.create_dad(port, 'HR');
      --
      dbms_epgc.set_dad_attribute(port, 'HR', 'username', 'scott');
      dbms_epgc.set_dad_attribute(port, 'HR', 'password', 'tiger');
      dbms_epgc.set_dad_attribute(port, 'HR', 'default_page', 'HR.hello');
      dbms_epgc.set_dad_attribute(port, 'HR', 'document_table', 'wpg_new_doctab');
      dbms_epgc.set_dad_attribute(port, 'HR', 'document_path', 'docs');
      dbms_epgc.set_dad_attribute(port, 'HR', 'upload_as_blob', 'txt');
      dbms_epgc.set_dad_attribute(port, 'HR', 'upload_as_long_raw', 'sql');
      dbms_epgc.set_dad_attribute(port, 'HR', 'document_proc',
                                  'HR.docpkg.process_download');


      --
      -- Commit the changes.
      --
      COMMIT;

END;
/
show errors;


--
-- Configure the embedded gateway for port 8080.
--
EXECUTE sample1_init_cfg(8080);
```

If you have worked with DADs before, you might be familiar with the syntax of the configuration files used by WebDB and OAS. You can import such information in a single operation, as demonstrated by the following program:

```
-- A sample procedure that configures an embedded gateway
-- using the import method.
--
CREATE OR REPLACE PROCEDURE sample2_init_cfg(port IN PLS_INTEGER) IS
   string VARCHAR2(2000);
BEGIN

   string := '
[PLSQL_GATEWAY]
adminpath = /admin_/
defaultdad = HR
stateful = yes
```

```
[DAD_APPS]
DEFAULT_PAGE=APPS.pkg.home
DOCUMENT_PATH=docs
DOCUMENT_PROC=APPS.doc_pkg.process_download
DOCUMENT_TABLE=APPS.doc_tab
STATEFUL=no
UPLOAD_AS_BLOB=jpeg, gif, txt
[DAD_HR]
DEFAULT_PAGE=HR.hello
DOCUMENT_PATH=docs
DOCUMENT_PROC=HR.docpkg.process_download
DOCUMENT_TABLE=wpg_new_doctab
USERNAME=scott
PASSWORD=tiger
UPLOAD_AS_BLOB=txt
UPLOAD_AS_LONG_RAW=sql
';

   dbms_epgc.drop_instance(port);
   dbms_epgc.create_instance(port);

   dbms_epgc.import(port, string);
   --
   -- Commit the changes.
   --
   COMMIT;

END;
/
show errors;

--
-- Configure the embedded gateway for port 8080.
--
EXECUTE sample2_init_cfg(8080);
```

See also the documentation about `mod_plsql` in the Oracle HTTP Server documentation for descriptions of the configuration parameters. Some of the caching and connection pooling parameters do not apply when the stored procedures are accessed outside of `mod_plsql`.

Package DBMS_EPGC

This package lets you configure database access descriptors (DADs) for the Oracle Servlet Engine.

The embedded PL/SQL gateway runs as a plug-in in the Oracle Servlet Engine embedded in the Oracle database. There is typically only one entry point to the Oracle Servlet Engine, istening for HTTP requests on the Oracle Net port to which `mod_ose` connects.

Because the configuration information is stored in the database rather than on a middle tier, it does not work with DADs from the Oracle HTTP Server. You can exchange configuration information with DADs on a middle tier using the `IMPORT/EXPORT` procedures in this package.

Security Model

Although all users have execute privileges on this package, the package performs its own security checking by maintaining a private list of administrative users: only these users can call the methods of this package. `SYS` and `SYSTEM` are always administrative users by default. The `GRANT_ADMIN` and `REVOKE_ADMIN` procedures control the embedded gateway administration privileges for other database users.

Transactional Behavior

All operations run in the caller's transactional context. The caller must explicitly commit after calling any update operations such as import, set, or drop.

To execute configuration operations in a separate transaction context, wrap the calls to this package in an autonomous PL/SQL block.

Types

The procedures in this package use the following type for passing parameters:

```
TYPE varchar2_table IS TABLE OF VARCHAR2(4000) INDEX BY BINARY_INTEGER;
```

Exceptions

The procedures in this package can raise the following exceptions:

```
config_error EXCEPTION;
PRAGMA EXCEPTION_INIT(config_error, -20000);
config_error_num CONSTANT PLS_INTEGER := -20000;
```

```
user_already_exists EXCEPTION;
PRAGMA EXCEPTION_INIT(user_already_exists, -20001);
user_already_exists_num CONSTANT PLS_INTEGER := -20001;

invalid_port EXCEPTION;
PRAGMA EXCEPTION_INIT(invalid_port, -20002);
invalid_port_num PLS_INTEGER := -20002;

invalid_username EXCEPTION;
PRAGMA EXCEPTION_INIT(invalid_username, -20003);
invalid_username_num PLS_INTEGER := -20003;

not_an_admin EXCEPTION;
PRAGMA EXCEPTION_INIT(not_an_admin, -20004);
not_an_admin_num PLS_INTEGER := -20004;

privilege_error EXCEPTION;
PRAGMA EXCEPTION_INIT(privilege_error, -20005);
privilege_error_num PLS_INTEGER := -20005;

dad_not_found EXCEPTION;
PRAGMA EXCEPTION_INIT(dad_not_found, -20006);
dad_not_found_num PLS_INTEGER := -20006;

invalid_dad_attribute EXCEPTION;
PRAGMA EXCEPTION_INIT(invalid_dad_attribute, -20007);
invalid_dad_attribute_num PLS_INTEGER := -20007;

invalid_global_attribute EXCEPTION;
PRAGMA EXCEPTION_INIT(invalid_global_attribute, -20008);
invalid_global_attribute_num PLS_INTEGER := -20008;

instance_already_exists EXCEPTION;
PRAGMA EXCEPTION_INIT(instance_already_exists, -20009);
instance_already_exists_num PLS_INTEGER := -20009;
```

Summary of Subprograms

CREATE_INSTANCE Procedure

Creates a gateway instance identified by a port number. You have to do this call before configuring attributes and privileges for the instance.

If the instance (port) is already in use, this operation results in an error.

The bulk configuration procedures (IMPORT and EXPORT) can be used without explicitly creating the instance.

The calling user of this routine automatically gets administrative privileges on this gateway instance.

```
PROCEDURE create_instance(port IN PLS_INTEGER);
```

DROP_INSTANCE Procedure

Drops the configuration information for a gateway instance identified by a port number. In some cases it might be easier to drop and recreate the instance than to modify it.

```
PROCEDURE drop_instance(port IN PLS_INTEGER);
```

DROP_ALL_INSTANCES Procedure

Drops the configuration information for all gateway instances in the database.

The caller of this procedure must either be SYS or have administrative privileges on all gateway instances in the database.

```
PROCEDURE drop_all_instances;
```

GRANT_ADMIN Procedure

The following APIs grant and revoke gateway administration privileges to database users. The SYS and SYSTEM users are always administrative users by default.

Grants gateway administrative privileges to a user.

```
PROCEDURE grant_admin(port IN PLS_INTEGER, username IN VARCHAR2);
```

REVOKE_ADMIN Procedure

Revokes gateway administrative privileges of a user.

```
PROCEDURE revoke_admin(port IN PLS_INTEGER, username IN VARCHAR2);
```

GET_ADMIN_LIST Procedure

Gets the list of gateway administrative users, other than SYS and SYSTEM. If no such users exist, the result is an empty table, with zero elements.

```
PROCEDURE get_admin_list(port  IN          PLS_INTEGER,
                         users OUT NOCOPY  VARCHAR2_TABLE);
```

SET_GLOBAL_ATTRIBUTE Procedure

Sets the value of a global attribute, one that applies to all DADs. If an attribute is already set for a given port number, the old value is overwritten with the new one.

Attribute names are not case-sensitive. Attribute values are sometimes case-sensitive, for example when the values represent UNIX filenames, but values such as Yes and No are not case-sensitive.

```
PROCEDURE set_global_attribute(port      IN PLS_INTEGER,
                               attrname  IN VARCHAR2,
                               attrvalue IN VARCHAR2);
```

GET_GLOBAL_ATTRIBUTE Procedure

Gets the value of a global attribute. Returns NULL if the attribute has not been set. Raises an exception if the attribute is not a valid attribute.

```
FUNCTION get_global_attribute(port      IN PLS_INTEGER,
                              attrname  IN VARCHAR2)
RETURN VARCHAR2;
```

DELETE_GLOBAL_ATTRIBUTE Procedure

Deletes a global attribute.

```
PROCEDURE delete_global_attribute(port      IN PLS_INTEGER,
                                  attrname  IN VARCHAR2);
```

GET_ALL_GLOBAL_ATTRIBUTES Procedure

Get all global attributes/values for an embedded gateway instance. The output is two index-by tables, one with the attribute names, and the other with the corresponding attribute values. If the gateway instance has no global attributes set, the output arrays are empty.

```
PROCEDURE get_all_global_attributes(port           IN  PLS_INTEGER,
                                    attrnamearray  OUT NOCOPY VARCHAR2_TABLE,
                                    attrvaluearray OUT NOCOPY VARCHAR2_TABLE);
```

CREATE_DAD Procedure

Creates a new DAD, with no attributes set. The DAD name is not case-sensitive. If a DAD with this name already exists, the old information is deleted.

```
PROCEDURE create_dad(port IN PLS_INTEGER, dadname IN VARCHAR2);
```

DROP_DAD Procedure

Drops a DAD from the gateway configuration.

```
PROCEDURE drop_dad(port IN PLS_INTEGER, dadname IN VARCHAR2);
```

SET_DAD_ATTRIBUTE Procedure

Sets an attribute for a DAD (Database Access Descriptor). It creates the DAD if it does not already exist. Any old value of the attribute is overwritten.

DAD names and DAD attribute names are not case sensitive. DAD attribute values might be case-sensitive depending upon the attribute.

```
PROCEDURE set_dad_attribute(port      IN PLS_INTEGER,
                            dadname   IN VARCHAR2,
                            attrname  IN VARCHAR2,
                            attrvalue IN VARCHAR2);
```

Example

```
set_dad_attribute(8080, 'myApp', 'default_page', 'myApp.home');
set_dad_attribute(8080, 'myApp', 'document_path', 'docs');
```

GET_DAD_ATTRIBUTE Procedure

Gets the value of a DAD attribute. Raises an error if DAD does not exist, or if the attribute is not a valid attribute. Returns NULL if the attribute is not set.

```
function get_dad_attribute(port      IN PLS_INTEGER,
                           dadname   IN VARCHAR2,
                           attrname  IN VARCHAR2) return VARCHAR2;
```

DELETE_DAD_ATTRIBUTE Procedure

Deletes a DAD attribute.

```
PROCEDURE delete_dad_attribute(port      IN PLS_INTEGER,
                              dadname   IN VARCHAR2,
                              attrname  IN VARCHAR2);
```

GET_DAD_LIST Procedure

Gets the list of all DADs for an embedded gateway instance. If no DADs exist, the result is an empty table, with zero elements.

```
PROCEDURE get_dad_list(port IN PLS_INTEGER,
                      dadarray OUT NOCOPY VARCHAR2_TABLE);
```

GET_ALL_DAD_ATTRIBUTES Procedure

Get all attributes of a DAD. The output is two index-by tables, one with the attribute names, and the other with the corresponding attribute values. If the DAD has no attributes set, the output arrays are empty.

```
PROCEDURE get_all_dad_attributes(port          IN  PLS_INTEGER,
                                dadname        IN  VARCHAR2,
                                attrnamearray  OUT NOCOPY VARCHAR2_TABLE,
                                attrvaluearray OUT NOCOPY VARCHAR2_TABLE);
```

IMPORT Procedure

The following procedures let you bulk load the configuration information for an embedded PL/SQL gateway. The input can be in any of the following forms:

- A VARCHAR2 (with a maximum length of 32 KB)

- An index-by table of VARCHAR2 variables

- A CLOB

The syntax of the configuration information must be the same as that used by the Oracle HTTP Server in the Internet Application Server. The easiest way to create it is to export it from an existing DAD.

```
PROCEDURE import(port IN PLS_INTEGER,
```

```
                        cfg  IN VARCHAR2);

PROCEDURE import(port IN PLS_INTEGER,
                 cfg  IN DBMS_EPGC.VARCHAR2_TABLE);

PROCEDURE import(port IN PLS_INTEGER,
                 cfg  IN CLOB);
```

EXPORT Procedure

The following procedures export the configuration information of an embedded
PL/SQL gateway to a flattened form so that it can be used with the Oracle HTTP
Server in *i*AS. The output can be any of the following:

- VARCHAR2 (with a maximum length of 32 KB)

- index-by table of VARCHAR2 variables

- CLOB

```
PROCEDURE export(port   IN  PLS_INTEGER,
                 cfg    OUT NOCOPY VARCHAR2);

PROCEDURE export(port IN  PLS_INTEGER,
                 cfg  OUT NOCOPY dbms_epgc.VARCHAR2_TABLE);

PROCEDURE export(port   IN  PLS_INTEGER,
                 cfg    OUT NOCOPY CLOB);
```

A

Abbreviations and Acronyms

This appendix lists some of the most common acronyms that you find in the areas of computer networking, distributed object development, and Java. In cases where an acronym refers to a product or a concept that is associated with a specific group, company, or product, the group, company, or product is indicated in brackets following the acronym expansion. For example: CORBA ... [OMG].

This acronym list is intended as a helpful guide. It is extensive, and should include every acronym in this book, but there are now so many computer-related acronyms that any list such as this must be incomplete.

3GL	third generation language
4GL	fourth generation language
ACID	atomicity, consistency, isolation, durability
ACL	access control list
ADT	abstract datatype
AJP	Apache JServ protocol
AFC	application foundation classes [Microsoft]
ANSI	American National Standards Institute
API	application program interface
AQ	advanced queuing [Oracle8]
ASCII	American standard code for information interchange
ASP	active server page(s) [Microsoft]
AWT	abstract windowing toolkit [Java]

BDK	beans developer kit [Java]
BLOB	binary large object
BOA	basic object adapter [CORBA]
BSD	Berkeley system distribution [UNIX]
C/S	client/server
CGI	common gateway interface
CICS	customer information control system [IBM]
CLI	call level interface [SAG]
CLOB	character large object
COM	common object model [Microsoft]
COM+	common object model, extended [Microsoft]
CORBA	common object request broker architecture [OMG]
CSS	cascading style sheet(s)
DB	database
DBA	database administrator, database administration
DBMS	database management system
DCE	distributed computing environment [OSF]
DCOM	distributed common object model [Microsoft]
DDCF	distributed document component facility
DDE	dynamic data exchange [Microsoft]
DDL	data definition language [SQL]
DHTML	dynamic HTML
DLL	dynamic link library [Microsoft]
DLM	distributed lock manager [Oracle8]
DML	data manipulation language [SQL]
DNS	domain name server, domain naming system
DOM	document object model
DOS	disk operating system
DSOM	distributed system object model [IBM]

DSS	decision support system
DTP	distributed transaction processing
EBCDIC	extended binary-coded decimal interchange code [IBM]
EJB	Enterprise JavaBean
ERP	enterprise resource planning
ESIOP	environment-specific inter-orb protocol
FTP	file transfer protocol
GB	gigabyte
GIF	graphics interchange format
GIOP	general inter-orb protocol
GUI	graphical user interface
GUID	globally-unique identifier
HTML	hypertext markup language
HTTP	hypertext transfer protocol
IDE	integrated development environment, interactive development environment
IDL	interface definition language
IEEE	Institute of Electrical and Electronics Engineers
IIOP	internet inter-ORB protocol
IIS	internet information server [Microsoft]
IP	internet protocol
IPC	interprocess communication
IS	information services
ISAM	indexed sequential access method
ISAPI	Internet server API [Microsoft]
ISO	international standards organization
ISP	internet service provider
ISQL	interactive SQL [Interbase]
ISV	independent software vendor

IT	information technology
J2EE	Java 2 Enterprise Edition [Sun]
JAR	Java archive (on analogy with tar, q.v.)
JAAS	Java Authentication and Activation Service (Sun)
JCK	Java compatibility kit [Sun]
JDBC	"Java database connectivity"
JDK	Java developer kit
JFC	Java foundation classes
JIT	just in time
JLS	Java language specification
JMS	Java messaging service
JNDI	Java naming and directory interface
JNI	Java native interface
JOB	Java objects for business [Sun]
JPEG	joint photographic experts group
JRMP	Java remote message protocol
JSP	JavaServer Pages [Sun] (sometimes used for Java stored procedure [Oracle], but this is deprecated)
JTA	Java transaction API
JTS	Java transaction service
JWS	Java Web Server [Sun, obsolete product]
KB	kilobyte
LAN	local area network
LDAP	lightweight directory access protocol
LDIF	LDPA data interchange format
LOB	large object
MB	megabyte
MIME	multi-purpose internet mail extensions

MIS	management information services
MOM	message-oriented middleware
MPEG	motion picture experts group
MTS	multi-threaded server [Oracle]
MTS	Microsoft Transaction Server [Microsoft]
NCLOB	national character large object
NIC	network information center [internet]
NIS	network information service [Sun]
NNTP	net news transfer protocol
NSAPI	Netscape server application programming interface
NSP	network service provider
NT	New Technology [Microsoft]
OCI	Oracle call interface
OCX	OLE common control [Microsoft]
ODBC	open database connectivity [Microsoft]
ODBMS	object database management system
ODL	object definition language [Microsoft]
ODMG	Object Database Management Group
OEM	original equipment manufacturer
OID	object identifier
OLE	object linking and embedding
OLTP	on line transaction processing
OMA	object management architecture [OMG]
OMG	Object Management Group
OO	object-oriented, object orientation
OODBMS	object-oriented database management system
OQL	object query language
ORB	object request broker
ORDBMS	object-relational database management system

OS	operating system
OSF	Open System Foundation
OSI	open systems interconnect
OSQL	object SQL
OTM	object transaction monitor
OTS	object transaction service
OWS	Oracle Web Server (obsolescent)
PAM	pluggable authentication module
PB	petabyte
PDF	portable document format [Adobe]
PGP	pretty good privacy
PL/SQL	procedural language/SQL [Oracle]
PNG	portable network graphics
POA	portable object adapter [CORBA]
RAM	random access memory
RAS	remote access service [Microsoft]
RCS	revision control system
RDBMS	relational database management system
RFC	request for comments
RFP	request for proposal
RMI	remote method invocation [Sun]
ROM	read only memory
RPC	remote procedure call
RTF	rich text file
SAF	server application function [Netscape]
SAG	SQL Access Group
SCSI	small computer system interface
SDK	software developer kit
SET	secure electronic transaction

SGML	standard generalized markup language
SID	system identifier [Oracle]
SLAPD	standalone LDAP daemon
SMP	symmetric multiprocessing
SMTP	simple mail transfer protocol
SOAP	simple object access protool [Microsoft]
SPI	service provider interface
SQL	structured query language
SQLJ	SQL for Java
SRAM	static (or synchronous) random access memory
SSL	secure socket layer
TB	terabyte
TCPS	TCP for SSL
TCP/IP	transmission control protocol/internet protocol
TM	transaction monitor
TP	transaction processing
TPC	Transaction Processing Council
TPCW	TPC Web benchmark
TPF	transaction processing facility
TPM	transaction processing monitor
UCS	universal character set [ISO 10646]
UDP	user datagram protocol
UI	user interface
UML	unified modeling language [Rational]
URL	universal resource locator
VAR	value-added reseller
VB	Visual Basic [Microsoft]
VRML	virtual reality modeling language
W3G	WWW Group

WAI	Web application interface [Netscape]
WAN	wide area network
WIPS	Web interactions per second [TPCW]
WWW	World Wide Web [W3G]
XA	extended architecture [X/Open]
XML	extensible markup language [W3G]
XSL	extensible stylesheet [W3G]
XSLT	extensible stylesheet language transformations
jdb	Java debugger [Sun]
tar	tape archive, tape archiver [UNIX]
tps	transactions per second

Index

H

Hidden Form Fields,   1-4
host name,   2-36
hotload
   WAR file subelement,   8-28
HRRoot
   sample Web service,   2-14
HTML,   1-3
HTML pages,   8-2, 8-3
HTTP
   client,   1-11, 2-15, 8-8
   presentation,   2-15
   protocol,   1-2, 1-3
   request
      DELETE,   7-13
      GET,   7-13, 7-19, 8-31
      HEAD,   7-13
      OPTIONS,   7-13
      POST,   1-5, 7-13, 8-31
      PUT,   7-13
      TRACE,   7-13
      types to be protected,   7-13
   request types,   7-13
   requests
      stateless,   2-17
   security,   7-2, 7-3, 7-5, 7-12
   security realm,   8-30
   service,   2-5, 2-14, 5-11
   services,   2-19
   session object,   2-5
httpd,   5-3
httpds.conf,   5-8, 5-22
   Apache configuration file,   5-4
httpdsctl,   5-4
HTTPS,   1-5
HttpServletRequest
   getContextPath() method,   2-37
   getPathInfo() method,   2-37
   getServletPath() method,   2-37
HttpSession object,   1-4

I

IfModule
   Apache directive,   5-5
Include
   Apache directive,   5-8
INIT.ORA,   3-3
intranet,   2-20
IP address,   2-20

J

J2EE,   2-5
JAR,   8-6
Java archive. See JAR
Java Naming and Directory Interface. See JNDI
Java Stored Procedure,   2-8
Java Virtual Machine. See JVM
JAVA$HTTP$REALM$GROUP$,   7-10
JAVA$HTTP$REALM$MAPPING$,   7-13
JAVA$HTTP$REALM$POLICY$
   permissions table,   7-14
JAVA$HTTP$REALM$PRINCIPAL$,   7-10
JavaBeans,   1-6, 8-2, 8-4
java.io.FilePermission,   2-12
JavaServer Pages. See JSPs
JDBC,   1-4
   server side internal driver for,   1-7
JNDI,   1-5, 2-8
   binding,   2-8, 2-9
   context objects in,   2-8
   namespace,   2-14, 2-33, 7-3, 7-14
   permissions in namespace,   7-3
   protection,   7-2
   realm type,   7-8
   reference objects in,   2-8
JServ,   1-2
jserver-loader
   WAR file subelement,   8-29
JSP,   1-6, 2-16, 8-3, 8-9, 8-36, 8-39, 8-41
jsp-info
   WAR file element,   8-28
JVM,   1-8, 8-1