

Oracle8*i*

Oracle Servlet Engine User's Guide

Release 3 (8.1.7)

July 2000

Part No. A83720-01

Oracle Servlet Engine User's Guide, Release 3 (8.1.7)

Part No. A83720-01

Copyright © 1996, 1999, 2000, Oracle Corporation. All rights reserved.

Primary Author: Susan Kraft

Secondary Author: John Russell

Contributors: Ellen Barnes, Jose Fernandez, Hal Hildebrand, Sunil Kunisetty, Angela Long, Jasen Minton, Brian Wright, Ronald Decker, Kannan Muthukkaruppan

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and JDeveloper™, Net8™, Oracle Objects™, Oracle8 i™, Oracle8™, Oracle7™, Oracle Lite™, PL/SQL™, Pro*C™, SQL*Net®, and SQL*Plus® are trademarks or registered trademarks of Oracle Corporation. All other company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Contents

1 Introduction

Serving Web Applications	1-2
Session Memory and Scalability.....	1-2
OSE in the Database	1-4
Database Sessions	1-4
Web Services	1-5
Web Domains.....	1-6
Single-Domain	1-8
Multi-Domain	1-9
Servlet Contexts	1-11
Servlets	1-12
Java Naming and Directory Interface.....	1-13
Permission and Schema	1-13
Summary of Containment Hierarchy	1-14
Security	1-14

2 The Servlet Basics with OSE

JNDI	2-1
Session Shell	2-3
Getting and Setting Object Parameters	2-3
Virtual Paths	2-4
Property Groups	2-4
Deploy a Servlet	2-5
Servlet Classes Published in a Servlet Context	2-6
Published Servlets	2-6
Serving an HTTP Request	2-8

Finding the Web Domain	2-8
Single-Domain Web Service:.....	2-8
Multi-Domain Web Services:	2-8
Finding the Servlet Context.....	2-9
Mapping the Virtual Path to a Servlet	2-9
Trying the Default Servlet - Static Content.....	2-10

3 JNDI and the Session Shell

About JNDI	3-2
JNDI Permissions	3-2
Invoking the Session Shell	3-3
Directory Navigation and Management	3-3
Permissions and Ownership	3-3
Overview of OSE Session Shell Commands	3-4
Service Configuration.....	3-4
Web Domain Configuration.....	3-4
Security Management	3-4
Servlet Context Management.....	3-5
Servlet Management.....	3-5
Export Commands.....	3-6

4 Architecture

HTTP Requests	4-2
URLs	4-2
Life Cycle of the HTTP Request	4-2
Endpoints	4-3
Finding the Web Domain	4-3
Finding the Servlet Context.....	4-4
Finding the Servlet.....	4-4
Finding for the Default Servlet	4-5
Database Sessions	4-6
User Identity of Database Sessions	4-8
Database Sessions	4-8
Database Session Creation, Termination, and Time-Outs.....	4-8
Database Session Time-Out.....	4-8

Automatic Termination of Database Sessions.....	4-9
HTTP Session Creation, Termination, and Time-Outs	4-9

5 Apache Module for Oracle OSE

Requirements for OSE with Apache	5-2
Overview of mod_ose in Apache for OSE	5-2
Configuring mod_ose in Apache for OSE	5-4
Connection Descriptors and Syntax of tnsnames.ora	5-5
Configuring Apache for OSE Applications	5-7
Stateful and Stateless Applications in Apache	5-7
Stateful and Stateless Handlers in Apache	5-7
Extracting Configuration Information for Apache	5-9
Topology of a Site Using mod_ose	5-11
mod_ose Installation for Pre-Installed Apache Systems	5-12
Apache Configuration.....	5-13
Specification of OSE Service	5-13
Specification of Dynamic Requests.....	5-13
Forwarding a URL	5-14

6 OSE Server Configuration

Setting Up OSE	6-2
Create Services	6-2
Create Domains	6-2
Contexts Group.....	6-2
MIME Group	6-3
Create Servlet Contexts	6-3
MIME Group	6-3
Add Servlets	6-4

7 Developer Tools and Procedures

Web Services	7-2
Virtual Host and IP Domains in the Web Service.....	7-2
Changing Ports of the Web Service.....	7-3
Managing Servlet Contexts	7-4

Creating or Change a Servlet Context	7-4
Deleting a Servlet Context	7-5
What Is in a Servlet Context?	7-6
config Object	7-6
doc_root Object	7-6
named_servlets Subdirectory.....	7-7
defaultServlet Object.....	7-7
policy Directory	7-7
httpSecurity	7-7
Servlet Context Group Parameters	7-8
context.params	7-8
context.mime	7-9
context.servlets.....	7-9
context.error.uris	7-9
Managing Published Servlets	7-10
Servlet Classes Published in a Servlet Context.....	7-10
Writing a Servlet	7-12
Writing the Servlet Code	7-12
Compiling the Servlet	7-12
Loading the Servlet Code	7-13
Publishing the Servlet	7-13
Accessing the Servlet.....	7-13

8 Security HTTP Administration

Overview	8-1
Prerequisites to Web Server Security	8-2
Authentication and Authorization	8-3
Declaring Principals	8-3
Groups.....	8-4
Users	8-4
Realms	8-4
DBUSER Considerations	8-5
The Tools	8-6
The Mechanics.....	8-6
Realms	8-6

Principals	8-7
Details.....	8-8
Protecting A Resource	8-8
The Mechanics.....	8-9
More Details	8-10
Declaring Permissions	8-10
The Mechanics.....	8-10
Declaring A Security Servlet	8-12
The Mechanics.....	8-12
Trouble Shooting	8-13

A Writing PL/SQL Servlets

Overview of PL/SQL Servlets	A-1
Configuring mod_ose to Run PL/SQL Servlets.....	A-2
Writing Stateful PL/SQL Stored Procedures	A-3
Configuring Database Access Descriptors from an Application	A-4
Package DBMS_EPGC	A-7
Summary of Subprograms	A-9
CREATE_INSTANCE Procedure.....	A-9
DROP_INSTANCE Procedure	A-9
DROP_ALL_INSTANCES Procedure	A-9
GRANT_ADMIN Procedure	A-9
REVOKE_ADMIN Procedure	A-10
GET_ADMIN_LIST Procedure.....	A-10
SET_GLOBAL_ATTRIBUTE Procedure.....	A-10
GET_GLOBAL_ATTRIBUTE Procedure.....	A-10
DELETE_GLOBAL_ATTRIBUTE Procedure	A-10
GET_ALL_GLOBAL_ATTRIBUTES Procedure	A-11
CREATE_DAD Procedure	A-11
DROP_DAD Procedure.....	A-11
SET_DAD_ATTRIBUTE Procedure.....	A-11
GET_DAD_ATTRIBUTE Procedure	A-12
DELETE_DAD_ATTRIBUTE Procedure.....	A-12
GET_DAD_LIST Procedure	A-12
GET_ALL_DAD_ATTRIBUTES Procedure.....	A-12

IMPORT Procedure.....	A-13
EXPORT Procedure.....	A-13

B Examples

Index

Send Us Your Comments

Oracle Servlet Engine User's Guide, Release 3 (8.1.7)

Part No. A83720-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

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

- Electronic mail — jpgcomnt@us.oracle.com
- FAX - 650-506-7225. Attn: Java Platform Group, Information Development Manager
- Postal service:

Oracle Corporation
Information Development Manager
500 Oracle Parkway, Mailstop 4op978
Redwood Shores, CA 94065
USA

Please indicate if you would like a reply.

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

Who Should Read This Book

This book has been written for the following audiences:

- **Management**—You may have purchased Oracle8*i* for reasons other than Java development within the database. However, if you want to know more about Oracle8*i* Java features, see *Oracle8*i* Java Developers Guide* for a management perspective.
- **Non-Java Developers**—Oracle database programming consists of PL/SQL and other non-Java programming. For experienced PL/SQL developers who are not familiar with Java, a brief overview of Java and object-oriented concepts is discussed in the first part of *Oracle8*i* Java Developers Guide*. For more detailed information on Java, see "[Java Information Resources](#)" at the end of this Preface.
- **Java Developers**—Pure Java developers are used to a Java environment that follows the Sun Microsystem specification. However, when Java is combined in the database, both Java and database concepts merge. Thus, the Java environment within Oracle8*i* is expanded to include database concerns. The bulk of this book discusses the differences you need to understand to run Java in the database. The following outlines the two viewpoints that arise from this merge:
 - * **Java environment**—Note that Oracle8*i* delivers an implementation that compiles with Java—any 100% pure Java code will work. Oracle8*i* JServer affects your Java development in the way that you manage your classes and the environment in which your classes exist. For example, the classes must be loaded into the database. In addition, there is a clearer separation of client and server in the Oracle8*i* model.

-
- * Database environment—You need to be aware of database concepts for managing your Java objects. This book gives you a comprehensive view of how the two well-defined realms—the Oracle8i database and the Java environment—fit together. For example, when deciding on your security policies, you must consider both database security and Java security for a comprehensive security policy.

If you are not familiar with JavaServer Pages, see "[Oracle8i JavaServer Pages Developer's Guide](#)".

Java Information Resources

The [Oracle Java Tools Reference](#) lists and describes all the commands used in managing the Oracle Servlet Engine.

The following table lists the sources of current information discussed in the Java programming documentation suite:

Location	Description
http://www.oracle.com/java	The latest offerings, updates, and news for Java within the Oracle8i database. This site contains Frequently Asked Questions (FAQs), updated JDBC drivers, SQLJ reference implementations, and white papers that detail Java application development. In addition, you can download try-and-buy Java tools 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 tutorials, book recommendations, and the Java Developer's Kit (JDK). The JDK is located at http://java.sun.com/products
http://java.sun.com/docs/books/jls http://java.sun.com/docs/books/vmspec	The Oracle8i Java Server (JServer) is based on the Java Language specification (JLS) and the Java virtual machine (JVM) specification.
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 bookstore 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 "[Oracle8i JavaServer Pages Developer's Guide](#)".

Introduction

The Oracle Servlet Engine (OSE) works as a specialized Web server, designed as a scalable *servlet* server inside the Oracle8i database.

The servlet classes are loaded into Oracle8i with a `loadjava` command and published in a namespace inside the database. A *servlet runner* handles HTTP requests, instantiates published servlets in sessions, and invokes servlet methods.

The following sections contains descriptions and diagrams of the following topics:

- [Serving Web Applications](#)
- [Web Services](#)
- [Summary of Containment Hierarchy](#)
- [Security](#)

You can use one of the create commands to define *domains*, *contexts*, and *servlets*. All contents are served by servlets. The servlet location is defined by the client in the URL.

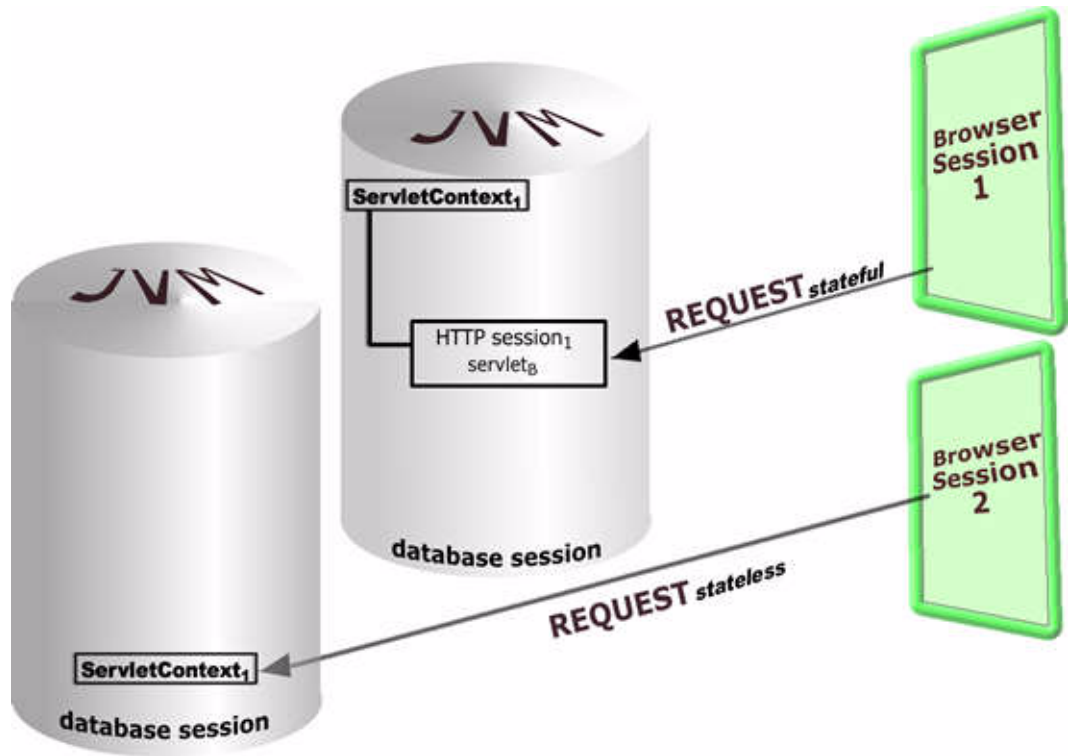
Serving Web Applications

A Web page is defined as an HTML file containing only static HTML descriptions of hyperlinks, and graphic and formatted text display. A Web *application* provides users with a more interactive experience than a static page can deliver. When HTML content is extended by servlets, creating dynamic content, that Web page becomes a Web application.

Session Memory and Scalability

The OSE is designed for virtually unlimited linear scalability. because each browser session is matched with a single virtual Java virtual machine (JVM), there can be multiple virtual JVMs (see [Figure 1-1, "Browser Sessions Matched With Virtual JVMs"](#)). The only limit on database platforms supporting the virtual JVM sessions is the amount of hardware present at any given time. See the [Oracle8iJava Developers Guide](#) for detailed information.

Figure 1-1 Browser Sessions Matched With Virtual JVMs



Although OSE serves static pages and runs CGI scripts, it is intended to be deployed as a servlet runner behind a standard Web server, such as Apache, Netscape, or IIS. See [Chapter 5, "Apache Module for Oracle OSE"](#).

One of the differences between OSE and other servlet engines is how the database session used is designed. Browser-sessions set up servlets sessions. The database session contains all the HTTP session *objects* created for that particular browser session. By employing database session memory, objects are not dropped after the request (connection) ends. However, objects are dropped when the database session ends.

OSE in the Database

OSE is a built-in Web server running inside Oracle8i supporting Web applications. Each session (virtual Java VM) executes the following items:

- JavaServer Pages (JSP)
- Servlets
- JavaStored Procedures

Database Sessions

The sessions terminate either explicitly or on a time-out configured by the administrator. For more information, see [Chapter 6, "OSE Server Configuration"](#).

The OSE unique attributes are:

- All servlets are activated in a browser-session sponsored database session.
- *Each database session has its own virtual JVM.* There are multiple virtual JVMs—one for every browser session.
- The OSE uses one virtual JVM per session, *isolating the sessions* and their static variables from each browser session (rather than creating a new thread with every browser session).
- *The database session contains all the HTTP session objects* created for that particular browser session.
- *An HTTPsession is activated per stateful servlet* context upon the servlet's request.
- An Oracle module, `mod_ose`, provides a layer between the Oracle HTTP server (Apache) and the Oracle8i database holding the Oracle servlet server (OSE). `mod_ose` defines ports, servers, and web domains by *setting communication variables*.

Web Services

In the hierarchical arrangement of this environment, OSE serves as an execution context for one or more *Web services*. You must specify a service root in the namespace when you create a Web service. The service root is the top level that contains the domain information for the entire Web service.

When you configure a Web service in OSE, you create a Web service, its end points, Web domains, and servlet contexts. A Web service is associated with one or more network end points. HTTP clients, who connect to a Web service end point, get contents from one of the associated Web domains. You must also configure all the attributes so that information in the namespace configuration has paths and pointers defined to handle and execute the requests.

OSE supports two types of Web service configurations:

- single-domain
- multi-domain

A single-domain Web service is sufficient for most cases. Configure the Web service to listen on a single port (8080 is used in all of our examples) and send any request received at the end point to the unique domain.

To support HTTPS, associate an additional end point to the single-domain Web service. This end point listens to a different port (9090 is used in all our examples), and you can configure it for SSL connections.

The multi-domain Web service is used for more advanced configurations, such as:

- In the *virtual hosts* configuration, one server with multiple Internet domain name system (DNS) names corresponds to multiple Web domains.
- In the *multiple IP address* configuration, one server with multiple network interface cards (NICs) corresponds to multiple Web domains.

In the multi-domain scenario, the network uses either the IP addressing, or the host name, or both to establish the request connection and to route the request to the correct domain.

Web Domains

A *Web domain* contains servlet contexts or, in the case of IP and virtual hosted, multi-domained Web services, another Web domain.

The Web domain for HTTP requests is identified by the address part of the URL. The root of the Web domain is located in the service root of the containing Web service.

Note: You can change the configuration of your Web domain structure from the default with tools that are described in the [Oracle Java Tools Reference](#).

The configuration parameters of each domain are in the `config` object in the Web domain's directory tree. Graphical representations of these configurations are shown in [Figure 1-2, "Single Domain Namespace Model Structure"](#) and [Figure 1-3, "Multi-Domain, Multi-Homed Example with Virtual Hosts In the Structure"](#). Notice the `config` object at the top level of the structure.

The following list describes the Web domain hierarchy:

- A Web domain has an owner corresponding to the schema namespace directory owner.
- For the single-domain case, *service root* is the location for both the Web domain and the root of the Web service. (See [Figure 1-2, "Single Domain Namespace Model Structure"](#) on page 1-8.)
- For the IP host domain case, the Web domain name is the host name located in the *service root*. (See [Figure 1-3](#) on page 1-8.)
- For the virtual host domain case, the Web domain name is the virtual host name, located in the *service root*. (See [Figure 1-3](#) on page 1-8.)
- A `/contexts` subdirectory, located in the domain root, contains the servlet contexts directories.
- The published servlets are under the `/named_servlets` directory.
- Parallel to the `/named_servlets` directory are configuration parameters in the namespace objects, `config`, `httpSecurity`, `policy`, `defaultServlet`, and `doc_root`.
- The `doc_root` object is a pointer (soft link) to static contents on the client.

The directory structure describing the namespace is a model you can use to explore and manipulate the elements that make up the Web domain. Each file and directory are actually objects with properties in the Oracle8i databases, rather than files with contents. However, for most of this discussion, a directory and file model will suffice while using the tools to create a Web domain.

Single-Domain

When a URL, such as `http://cavist.com:8080/cellar/welcome.html`, is sent to a *single-domain*, the request accesses the Web service listening on port 8080, on the host named `cavist.com`.

Example 1-1 Single-Domain: Port Relationship and Domain Root in URL

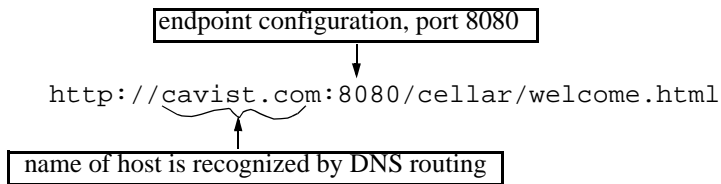
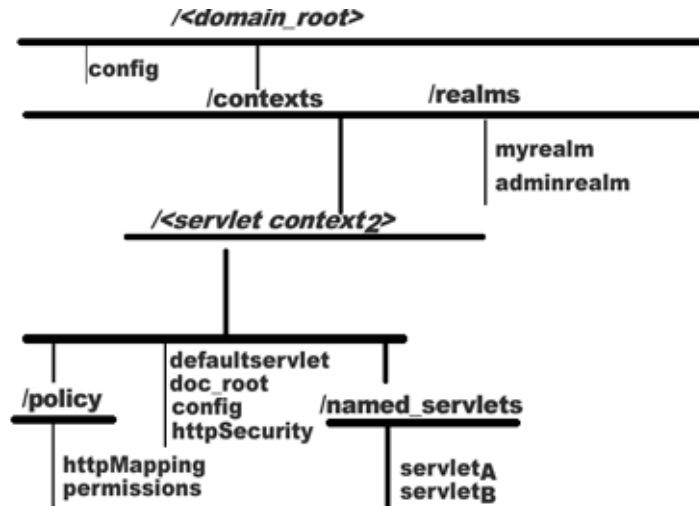


Figure 1-2 Single Domain Namespace Model Structure



Multi-Domain

When you set-up a *multi-domain* Web service, the Web domain name is dependent on the configuration type of the service root. An example of a multi-domain, multi-homed configuration (more than one IP address on a machine) combined with multiple virtual hosts is shown in [Figure 1-3](#).

For example, a virtual host, `cavist.com`, in a domain defined by the IP address, `10.1.1.20`, can have the identical structure naming pattern as another IP address and virtual host pairing, such as `10.1.1.30` hosting `jones.com`, sharing the same *service root*. The names are defined in a path so that the lower branches are uniquely defined by the domain names.

Example 1-2 Multiple-Domain: Port Relationship and service root in URL

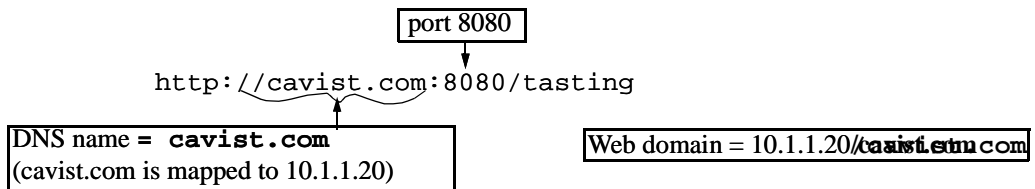
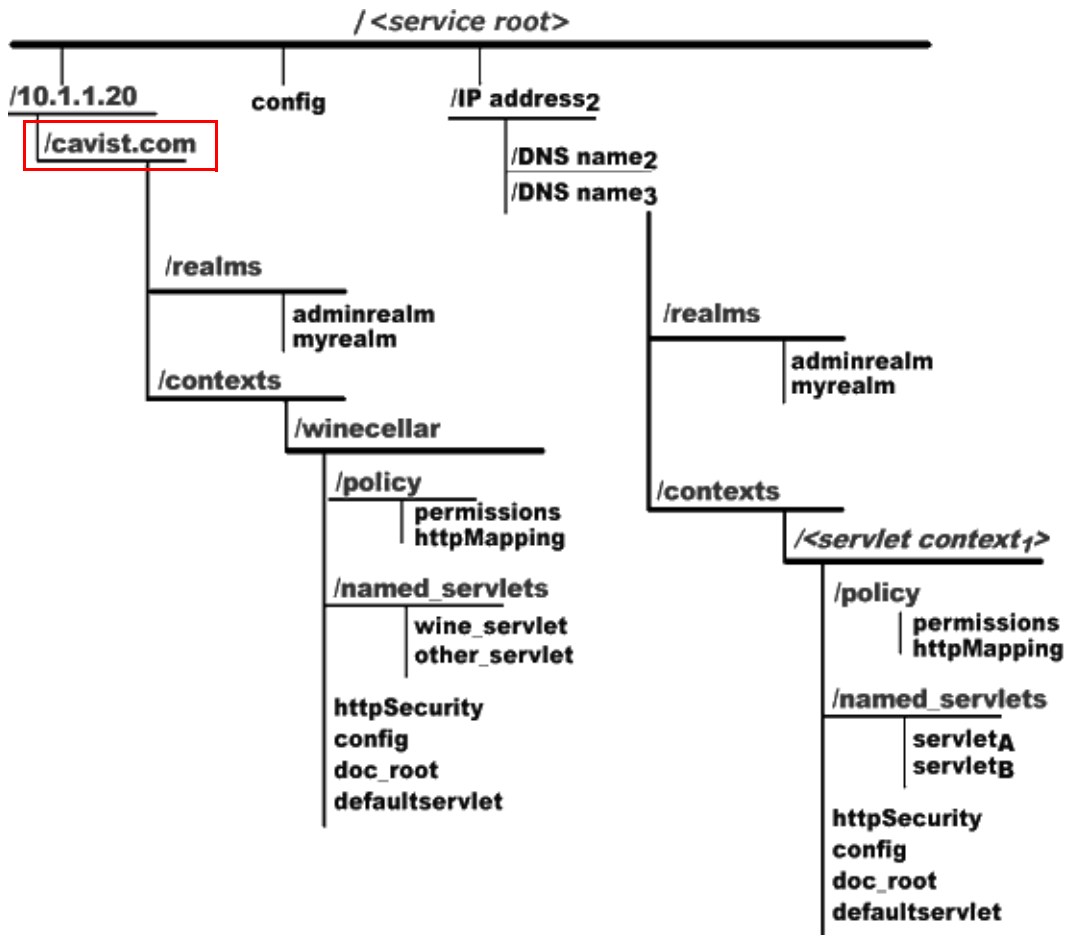


Figure 1-3 Multi-Domain, Multi-Homed Example with Virtual Hosts In the Structure



Servlet Contexts

Think of a *servlet context* as an application loaded into OSE. It is a set of servlets, configuration parameters, JSPs, and pointers to static contents on the file system that are all accessible below the same virtual path. The servlet context is usually identifiable as the first segment of a URL's path.

A servlet context configuration describes how the Web server behaves when serving its contents (security, timeouts, MIME types, mapping of virtual paths extensions to servlets, statefulness). OSE supports nested servlet contexts that can inherit configuration properties from their parents.

In practice:

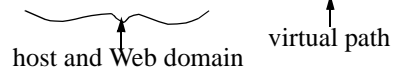
- The servlet context parent directory, `/contexts`, is a namespace subdirectory of its domain root directory.
- The `winecellar` servlet context is stored in the `<domain root>/contexts/winecellar` directory.
- The servlet context directory contains its configuration parameters, a pointer to the static document root, and a subdirectory, `named_servlets`, containing servlets.

The configuration information of a servlet context is a namespace object, `config`, in the servlet context directory. Look at [Figure 1-2, "Single Domain Namespace Model Structure"](#) and notice the `config` object listed under the `<servlet context1>` directory.

If the `config` object contains the entry `/cellar` as mapped to `/winecellar`, then the URL `http://cavist.com:8080/cellar/welcome.html` would access the servlet context, `winecellar`, in the Web domain, `cavist.com`.

Example 1–3 Mapping Virtual Paths to Servlet Contexts

`http://cavist.com:8080/cellar/welcome.html`


host and Web domain virtual path

`<Web domain>/winecellar/welcome.html`


Servlet Context

`/cellar is mapped to /winecellar`

Servlets

A *Servlet* is a Java class. Load servlet classes and any related support classes, into the database with the `loadjava` command. Publish the servlet into a servlet context with the session shell.

Publishing a servlet:

- creates a named namespace object in the `named_servlets` subdirectory of the servlet context directory
- associates a virtual path with the servlet, as contained in the `config` object of their servlet context to be used when matching against a URI

Example 1–4 Servlets Associated with the HTTP Virtual Path

A servlet accessible as `http://cavist.com:8080/cellar/winefinder` could be published as a namespace object named `service root/contexts/winecellar/named_servlets/winefinderservlet` with the virtual path mapping entry `/winefinder=winefinderservlet` in `service root/contexts/winecellar/config`.

Java Naming and Directory Interface

Earlier, we said the servlets are published in a namespace and that the directory structure was a model. This namespace is the Sun Java Naming and Directory Interface (JNDI). The OSE's *JNDI* implementation uses SQL tables to store the contents of a JNDI accessible namespace.

Access the JNDI with the session shell command-line tool. Use these commands to explore and manipulate the namespace.

Using the session shell you can navigate the namespace as if it were a file system to change directories and list the contents (see [Chapter 3, "JNDI and the Session Shell"](#) for tool definitions and examples).

The contents are organized into hierarchical containment relationships of security, mapping, servlets and default objects. Servlet contexts correspond to applications deployed in the Web server. Servlet contexts, which are mapped to the address part of an URL, are grouped in Web domains.

Example 1-5 URL Shows a Client Accessing Contents

```
http://<Web domain:port>/<servlet-context>/<path>
```

We can translate this example URL into a real URL and name such as this:

```
http://cavist.com:8080/winecellar/welcome.html
```

Web domains and servlet contexts have owners (Oracle schema) with full administration rights. The owner of the Web domain can administer the Web domain, create servlet contexts, and give access to other schemas. The owner of a servlet context can publish contents in it and tune its configuration parameters.

Permission and Schema

The ownership and permissions are similar in construct to the UNIX environment, in that there are read, write, and execute abilities defined with session shell commands. Web domain schemas are the only valid users in the JNDI namespace. The users are given permissions and ownership in their domains by administrators.

Summary of Containment Hierarchy

The containment hierarchy of the OSE is defined as:

Web Service(s) > Web Domain(s) > Servlet Context(s) > Servlets

Security

OSE supports *authentication* and *access control* as required by the Servlet 2.2 Specification.

In defining your security, you must define a valid set of users. You can:

- use the database users as its population
- define your own realm principals

Principal declarations are held in the `realms` directory of a Web service. The following are properties of realms:

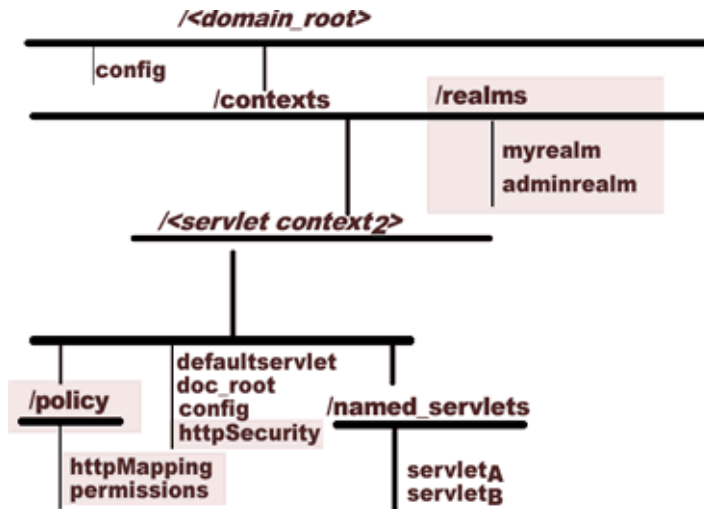
- Realm definitions are within the scope of a service.
- A realm is made up of *users* and *groups*.
- All `servlet contexts` within a service can use the same realm definitions.

The `policy` directory is automatically created when you set permissions and create URL security mappings.

See [Figure 1-4, "Security Components in the JNDI Namespace"](#), for the general relationship of `policy` to a servlet context.

[Figure 1-4](#) shows the relationship of the `realms` and `policy` directories with respect to the rest of the domain in the JNDI namespace. `realms` is at the top level of the domain, whereas `policy` is a sub-directory of `servlet context`.

Figure 1-4 Security Components in the JNDI Namespace



Versions OSE supports servlets 2.1 and JSPs 1.0.

The Servlet Basics with OSE

This chapter contains a series of operations designed to be helpful during your first time using the servlet engine. The topics we cover are the following:

- [JNDI](#)
- [Session Shell](#)
- [Deploy a Servlet](#)
- [Serving an HTTP Request](#)

JNDI

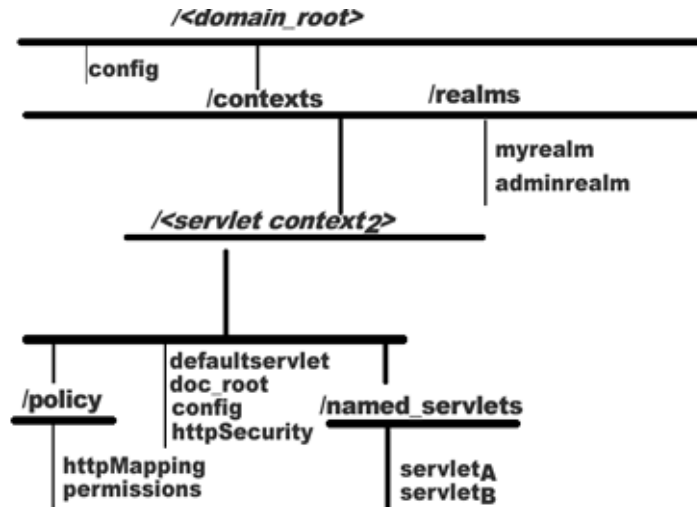
The JNDI namespace is an API storing information and contents of the servers, and provides naming and directory functionality to applications written in the Java programming language. JNDI is independent of specific naming or directory service implementations, enabling Java applications to access different naming and directory services with a common API. Various naming and directory service providers are plugged in seamlessly behind the common API, which allows Java applications to coexist with legacy applications and systems, such as a file system.

A full JNDI server, inside Oracle8i, is accessible with JNDI APIs from inside and outside of Oracle8i. When managing the Web server, you can interactively see and change the contents and properties of the namespace with session shell tool. See the [Oracle Java Tools Reference](#) for a complete discussion of the session shell commands and syntax.

The namespace is hierarchical, with two types of entries, *objects* and *directories*.

- Objects are used to store Java references.
- Directories contain other directories or objects. Paths to JNDI entries are indicated with the regular UNIX notation `/dir/dir/dir/leaf`.

Figure 2–1 JNDI OSE Namespace Structure Showing `dir/dir/dir/leaf`



Session Shell

The session shell commands, shown in this section, are used in the most commonly performed tasks in the OSE management. In the following sections, you will see a simple set of commands used to create and manipulate objects in the JNDI.

Note: If a multiple line entry is invoked by using a double-quote, the session shell prompts for the next entry with a right-angle bracket, ">", until another double-quote is entered.

Another method for multiple line entries is to use a backslash "\" as a continuation indicator.

There are session shell commands for use with the namespace. There are different styles of commands which are characterized as follows:

- a *UNIX-like command set* for exploring and manipulating the namespace
- *commands* to get and set properties of JNDI objects
- *specialized commands* to create new objects (for example, the servlet contexts and their configuration parameters, and servlets).

See [Chapter 3, "JNDI and the Session Shell"](#) for an overview of these commands. Read [Oracle Java Tools Reference](#) for a complete description of these commands.

Getting and Setting Object Parameters

Use the `getproperties` command to show all parameters of a JNDI object and the `setproperties` command to change the parameters. You can also change the parameters, along with their entire group parameters, using the `setgroup` command.

The syntax for the `setparameters` and `getparameters` are:

```
getproperties <object>
setproperties <object> <properties>
```

In `getproperties`, the properties are printed as a Java properties file. In `setproperties` the properties need to be passed in with the same syntax. You will likely have more than one property. Enter a double-quote before the first property, and use the carriage return after each entry. The shell prompts with an angle bracket

(>) for more data or an end of entry. Use double-quotes after entering the last property, marking the last entry (i.e., `>foo=bar"`).

Note: The `setproperties` command clears all of the properties of the object before setting the specified properties.

Example 2-1 Change the doc_root Path

```
$ getproperties doc_root
FSContextURL=/usr/local/oracle/jis/public_html
$ setproperties doc_root FSContextURL=/usr/local/winecellar
$ getproperties doc_root
FSContextURL=/usr/local/winecellar
```

Note: When entering the new path, specify the whole path.

For more information on `doc_root`, see "[doc_root Object](#)", on page 7-6.

Virtual Paths

Virtual paths can be associated with a servlet when publishing it to the servlet context sub-directory, `named_servlets`.

Given that an *URL* is defined by:

protocol + servername + URI, then the structure is: `http://<servername>/<URI>`.

The *URI*, shown as part of the URL, is defined as the *request URI* comprised of the **ContextPath + ServletPath + PathInfo**

The first portion of the virtual path mapping to a servlet context is called the servlet context's *virtual path*; the second portion mapping into a particular servlet is called the Servlet's virtual path. Any left over portion in the URI is given as `PATH_INFO` to the servlet.

Property Groups

Defines a group of parameters within a JNDI entry.

Deploy a Servlet

In the previous sections you read about, and perhaps tested, examples of JNDI objects with these procedures:

1. Create a JNDI object.
2. Display properties of the `doc_root` object, with the `getproperties` command.

This section explains how to publish and run a servlet using the following commands:

- `loadjava`
- `publishservlet`
- `setproperties`

Servlet Classes Published in a Servlet Context

Servlet classes are loaded into Oracle8i with `loadjava` and published in a servlet context with the session shell commands. Publishing a Servlet creates a JNDI object in the `named_servlets` subdirectory of the servlet contexts.

This JNDI object lists the servlet classname and its initialization parameters. Servlets are associated with an HTTP virtual path as described in their JNDI servlet context `config` object.

Use `loadjava` to load the servlet class and any support classes into the database. (See [Oracle Java Tools Reference](#) for a detailed description of this tool.)

Published Servlets

Published servlets are JNDI objects of class

`SYS:oracle.aurora.mts.ServletActivation` under the `named_servlets` directory of a servlet context. To be accessible from an HTTP client, servlets must be associated with a virtual path, or a wild-card name, in their servlet context `config`.

```
publishservlet [-virtualpath <path>] [-stateless] [-reuse] [-properties props] \  
contextName <servletName> [className]
```

`-virtualpath` option and *path (optional)*: virtual path to associate with this servlet for invocation

`-stateless` flag (*optional*): tells OSE the servlet is stateless. When this is set, the servlet has no access to the `HTTPSession`.

contextName: the name assigned to this `context servlet` directory.

servletName: the name assigned to this servlet in the `named_servlets` directory.

className: the name of the class implementing the `HttpServlet` interface.

This command publishes a servlet by name in the context and associates a virtual path with the named servlet.

Example 2-2 Publish a Servlet with publishServlet Command

```

          servlet path          contextname
          |                     |
          v                     v
$ publishServlet -virtualpath /tastings /webdomains/contexts/winecellar \
  tastingServlet SCOTT:winemasters.tasting.Tasting
          ^                     ^
          |                     |
    servletName          className

```

Verify the servlet is published, enter:

```

$ ls /webdomains/contexts/winecellar/named_servlets
tastingServlet

```

Verify the virtual path mapping

```

$ getgroup /webdomains/contexts/winecellar/config context.servlets
/errors/internal=internalError
/tastings=tastingServlet

```

Example 2-3 Modify a Published Servlet Properties with setproperties Command

You can add additional properties with the session shell commands then access those new properties from the servlet code with the setproperties.

```

$ getproperties invoker
servlet.class=SCOTT:winemasters.tasting.Tasting

$ setproperties invoker "servlet.class=SCOTT:winemasters.tasting.Tasting
>details=high
>style=parker"

$ getproperties invoker
servlet.class=SCOTT:winemasters.tasting.Tasting
details=high
style=parker

```

Serving an HTTP Request

This section describes the algorithm OSE uses to find the right servlet.

Servlets handle all HTTP requests. The request deployment begins when a client sends an HTTP request to the server. OSE extracts the virtual path from the request.

When a service is invoked, the `config` object uses the virtual path to map the servlet to the request.

The following four sections describe how HTTP requests are serviced.

Finding the Web Domain

The Web domain that handles the requests is dependent on the Web service that is configured for OSE.

Single-Domain Web Service:

The Web domain handling the request is the single Web domain associated with the service.

Multi-Domain Web Services:

If you configured more advanced Web Services, then the domain used depends on the service and the URL employed to access OSE. Refer to the [Multi-Domain](#) discussion in [Chapter 1, "Introduction"](#) for more information.

Finding the Servlet Context

OSE searches in the contexts properties group of the Web domain `config` object. This set of properties is a list of mappings for virtual paths to Context names.

OSE matches the virtual path of the request against each entry in the Contexts properties. The longest match indicates which servlet context will serve the request.

Example 2–4 HTTP Requests Finding the Servlet Context

If the URL is:

```
http://cavist.com:8080/cellar/bordeaux.wine
```

and the virtual path from the Web domain `config` object maps the virtual path to the servlet context:

```
/cellar=/winecellar
```

Then the servlet context, `cavist/contexts/winecellar`, is used.

If a servlet context match is not found, the request is served by the context:

```
/cavist/contexts/default
```

Mapping the Virtual Path to a Servlet

When the servlet is mapped to the URI, the portion of the URI path that was mapped to the servlet context is ignored. The remaining portion is matched with entries in the `context.servlets` properties of the servlet context `config` object. The longest match indicates which servlet will handle the request.

Entries in the `context.servlets` properties can be paths or wild-card names according to the Servlet 2.2 Specification. Partial paths have priority over wild-card names. Exact matches have priority over both partial paths and wild-card names.

Example 2–5 HTTP Requests Finding the Servlet

The virtual path `/cellar/bordeaux.wine` is served by the context `/cavist/contexts/winecellar`.

To find the name of the servlet, `bordeaux.wine` is matched against the virtual paths mapping, defined by the servlet context `config` object.

Within the servlet context, if a servlet match is not found, the request is served by the default servlet.

Trying the Default Servlet - Static Content

If no match for the requested URL is found, 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.

Example 2-6 Request Served by defaultServlet

Suppose the `winecellar/doc_root` object points to the file system directory, `/usr/local/coolplace/html`

Therefore, the contents of the `doc_root` object parameters are `FSContextURL=/usr/local/coolplace/html`

The URL is `http://cavist.com:8080/cellar/labels/lafitte.gif`

The virtual path, `/cellar` is mapped to a servlet context, `/winecellar`. If the servlet context did not have any matches for `/labels/lafitte.gif` (or a partial match, such as `/labels/*`), then the default servlet is used.

The default servlet:

1. starts and receives the `pathInfo, /labels/lafitte.gif`
2. matches the `pathInfo` below `doc_root`.
3. retrieves `/usr/local/coolplace/html/labels/lafitte.gif`

This match must be *full* matches. If no default servlet is found, OSE generates an internal error code.

JNDI and the Session Shell

This chapter contains tool definitions and examples. The topics we cover are the following:

- [About JNDI](#)
- [JNDI Permissions](#)
- [Invoking the Session Shell](#)
- [Overview of OSE Session Shell Commands](#)

A full JNDI server, inside Oracle8*i*, is accessible with JNDI session shell commands from inside and outside of Oracle8*i*. You can interactively manipulate the contents of the namespace with the session shell.

About JNDI

JNDI stores information and contents of the different Java servers— OSE, CORBA and EJB— running in Oracle8i.

The namespace is hierarchical with two types of entries, directories and objects.

Directories contain other directories or objects. Paths to JNDI entries are indicated with the regular UNIX notation `/dir/dir/dir/leaf`.

Objects are used to store Java References. A reference is an instance of the `javax.naming.Reference` class. The JNDI server gets the Reference components (Class name, Class factory name and parameters) and uses this information to instantiate the Java object. The session shell provides commands to store new references in the JNDI namespace and to manipulate their parameters. Additionally, the session shell provides a set of commands to navigate the namespace, remove entries, and change their permissions.

See figures in [Chapter 1, "Introduction"](#), for depictions of the JNDI structure.

JNDI Permissions

All contents in the JNDI server are secure. You must have the correct access rights to view and modify contents when accessing the JNDI server as a user. Similar to the UNIX file system, JNDI supports three type of access rights: **read**, **write**, and **execute**. Access can be granted to databases users (schema) or database roles. This can be done with one of the shells or programmatically. Setting **read**, **write**, or **execute** JNDI permissions is similar to setting file and directory permissions with UNIX.

The JNDI server also uses the ownership concept, where each entry is owned by a database schema. This is similar to the UNIX file system.

Invoking the Session Shell

All commands execute on the server by remote access and must specify the communication transport in use to contact the server. The following communication options are available: JDBC, HTTP and IIOP (SESS_IIOP).

To access the session shell commands on a client/server, type:

```
sess_sh -s transportURL -u[ser] username[/passwd] [-p[assword] passwd] [otherargs... ]
```

-s *service* transportURL: specifies the transport to use for communicating with the server, in the form of a URL descriptor.

The following URLs are supported:

jdbc:oracle: <i>type:spec</i>	A JDBC URL that specifies how to connect to the database using JDBC.
http:// <i>host:port</i>	An HTTP URL indicating the host and port to use to connect to the administrative webserver pre-installed in the database.
sess_iiop:// <i>host:port[:sid]</i>	A SESS_IIOP URL indicating the host, port and SID for the GIOP listener on the server.

-u *username*: The login name for the database session.

-p *passwd*: The password during login.

-command "*cmd...*": A command to execute on the server

See the [Oracle Java Tools Reference](#) for complete details regarding the shell tool and environment.

Directory Navigation and Management

The session shell supports the standard UNIX commands for navigation and managing directories, such as `cd`, `pwd`, `ls`, `mkdir`, and `rm`.

Permissions and Ownership

Each JNDI entry has permissions set. The JNDI server supports three types of permissions: READ, WRITE and EXECUTE. Permissions can be granted to or removed from databases users or groups with the shell `chmod` command.

Overview of OSE Session Shell Commands

The session shell provides a set of specialized commands to manage the Web server and publish servlets. See [Oracle Java Tools Reference](#) for syntax requirements. The uses of each command, for manipulating the OSE JNDI namespace, are briefly described here.

Service Configuration

This set of commands creates new services.

`createservice` —Manipulates parameters required by the ServicePresentation code, without defining the endpoints (TCP ports) for service.

`addendpoint` —Adds a new endpoint and performs dynamic registration of the endpoint with the Listener storing it in the dynamic registration tables.

`rmendpoint` —Removes a specific endpoint from a service and the dynamically registered ports from the Listener.

`destroyservice` —Removes the service and all its endpoints including their dynamic registration. The `-all` flag erases the entire JNDI tree (from the service root level).

`createwebservice` —Manipulates parameters required by the ServicePresentation code and initializes Web specific configurations.

Web Domain Configuration

This set of commands sets the location of servlet contexts. As with all JNDI entries, each location has an administrator/owner

`createwebdomain` —Creates a Web domain administered by the current schema, where servlets execute as that schema. In addition it defines an initial servlet context, `default`, and `doc_root`.

`destroywebdomain` —Removes the Web domain and all associated servlet contexts.

Security Management

This set of commands sets the security for your domain and servlets specifying the realm and the authentication method to be used by the security class.

`realm` —Lists all the realm commands.

`realm list -w <Web service root>` —Lists all realms declared for a service.

`realm map -s <ServletContextPath> [-(add | remove) <path> -scheme <auth>:<realm>]` —Defines/lists(/declares not to be) protected paths within a servlet context.

`realm echo [0 | 1]` —Turns echo off or on.

`realm publish -w <Web service root> [-(add | remove) <realmName> [-type (RDBMS | DBUSER | JNDI)]]` Creates/publishes/deletes an realm.

`realm user -d <domainContextPath> -realm <realmName> [-(add | remove) <userName> [-p <user password>]]` —Creates/deletes a user.

`realm group -d <domainContextPath> -realm <realmName> [-(add | remove) <groupName> [-p <group password>]]` —Creates/deletes a group.

`realm parent -d <domainContextPath> -realm <realmName> [-group <groupName> [-(add | remove) <principalName>]] [-query <principalName>]` —Adds/lists/removes a principal to a group.

`realm perm -d <domainContextPath> -realm <realmName> -s <ServletContextPath> -name <principalName> [-path <path> (+ | -) <permList>]` —Declares, clears, or lists a granted or denied permission on the specified path for a user for valid HTTP methods ([Declaring Permissions in Chapter 8, "Security HTTP Administration"](#)).

Servlet Context Management

This set of commands manipulates the Context.

`createcontext` —Creates a context on the corresponding virtualpath of the domain.

`destroycontext` —Removes servlet context information and all the servlets from that domain.

`adderrorpage` —Defines which URL reports errors for this context for an error code.

`rmerrorpage` —Remove the error page associated with the corresponding error code.

Servlet Management

This set of commands handles the publishing and unpublishing of servlets.

`publishservlet` —Publishes a servlet by name in the Context. It also can associate a virtual path with the named servlet. Servlets published in a servlet context declared as stateless, are not allowed access to the HTTPSession object.

`unpublishservlet` —Removes the servlet from the servlet context, as well as any existing virtualpath for the servlet in the mapping table.

Export Commands

This set of commands provides the means for extracting the structure of a Web domain and can generate the corresponding configuration file. Use this command-generated file for `mod_ose` or other proxies.

`exportwebdomain`—The `export` utility can be used in one or two stages when generating a configuration file:

1. In XML format, generate the structure of a Web domain or contexts within a domain.
2. (*optionally*) Apply transformations to the XML structure, producing a configuration file for a specific Web server (for example: `apache`, `iis`).

Refer to [Chapter 5, "Apache Module for Oracle OSE"](#) for a detailed explanation regarding exporting the Web domain.

Architecture

This chapter describes how OSE handles requests for single-domain and multi-domain Web services. You can follow the logic OSE uses to find the right servlet. The topics we cover are the following:

- [HTTP Requests](#)
- [Life Cycle of the HTTP Request](#)
- [Database Sessions](#)

HTTP Requests

When the first HTTP request is received from a browser, a session is created for the client. Any following requests are routed to the session. The session tracking mechanisms are cookies or URL rewrites.

URLs

The URL is composed of the protocol, the server, and the Uniform Resource Identifier (URI), as such:

```
<protocol>://<server>/<URI>
```

The URI is specified by a client in URL request. The URI contains information OSE uses to match against declared virtual paths. The mappings in `config` objects are used to determine which servlet to employ for a given request. If a servlet cannot be located, the default servlet is employed.

Life Cycle of the HTTP Request

As an HTTP request life-cycle example, we use the URL `http://cavist.com:8080/cellar/bordeaux.wine` in the multi-domain depicted in [Figure 1-3](#).

The following steps show the progress of this cycle:

1. The browser sends an HTTP *request*.
2. The request is received on port 8080.
3. A database session is created and OSE is given the request.
4. OSE uses a service to listen on port 8080. (Refer to [Chapter 6, "OSE Server Configuration"](#) for instruction on how to set up the OSE.)
5. The request is examined by the web service and looks at the IP address, mapping it to the IP domain.
6. The DNS name (`cavist.com`) is mapped to the JNDI namespace directory (`cavist.com`).
7. The `10.1.1.20/cavist.com/config` object is used to match the URI into a servlet context (the URI virtual path `cellar` maps to the JNDI namespace entry, `winecellar`). See [Finding the Servlet Context](#) on page 4-4.

8. The servlet context `config` object is used to match the remaining portion of the URI (`bordeaux.wine`) to the correct servlet,
`cavist.com/contexts/winecellar/named_servlets/wine_servlet`
9. The servlet, `wine_servlet`, services the request,
`http://cavist.com:8080/cellar/bordeaux.wine`
10. session *terminates*. (see [Database Session Time-Out](#) on page 4-8.)

Endpoints

Endpoints are the ports which receive the information. Net8 controls the load balancing and connectivity. In order to receive incoming requests, the listener must be configured with an endpoint for each presentation type, using the IP address of the host and a valid port number to be assigned to the listener.

Finding the Web Domain

The Web domain handling requests directly depends on the containing Web service configuration of the OSE.

When a Web service takes the request as a:

- single-domain, there is only one Web domain associated with the service. By default it is contained in the *service root*.
- multi-domain, the Web domain is tied to the IP address associated with the DNS name of the URL.

Hint: For a review of multi-domain set ups, see [Example 1-2](#) as an example followed by a multi-domain directory tree [figure](#).

Finding the Servlet Context

OSE searches in the domain `config` object for the set of properties that maps the virtual paths to servlet context names. OSE then matches the URI of the request against each entry in the `config` mapping properties. The longest match indicates which servlet context will serve the request. If the requested servlet context is not found, the default servlet context is used. The default servlet context is stored in, `/<domain root>/contexts/default`.

Example 4-1 HTTP Requests

If the URI is `/cellar/bordeaux.wine`

and the contents of the `contexts` group from the `config` object are:

```
/cellar=/winecellar
/test=/test
```

then the partial URI match `/cellar` is served by the *context*, `/contexts/winecellar`

Finding the Servlet

After the servlet context is identified, the unmatched portion of the URI is matched against the virtual paths in the servlet context `config` object. OSE ignores the portion in the virtual path corresponding to the servlet context. The remaining portion is matched against entries in the `context.servlets` properties of the servlet context. The best match indicates which servlet handles the request.

Note: Entries in the `context.servlets` properties can be paths or wild-names (see the Servlet 2.2 Specification). Partial paths have priority over wild-names. Exact matches have priority over both partial paths and wild-names.

Example 4-2 Request Served by Servlet

Continuing with the scenario from the previous example, 4-1, the named *servlet*, `tastingServlet`, is published in `winecellar/named_servlets`.

In this case, `bordeaux.wine` is mapped by the servlet context to a *servlet* entry named `tastingServlet`.

Finding for the Default Servlet

If no match for the requested URL is found, OSE looks for a servlet named `defaultServlet`, the servlet context directory. When the *default servlet* is found, it processes the request.

When a Web domain is created, OSE installs a default servlet in the domain servlet context directory. There is always a default servlet in the Web service, but you can provide a virtual mapping of `/` to function as your new default servlet.

If the unused portion of the URI does not match or partially match the virtual paths published in a servlet context, then this same URI portion is used by the default servlet. The default servlet looks in the `doc_root` for an exact match. If a match exists, the page is served. If no exact match is found, OSE generates an error code.

Mappings in the servlet context `config` object that map error codes to error pages. For example, an error code 404 is mapped to the URI named `/system/errors/404.htm`. This results in the default servlet serving that error file from the name and location in the `doc_root`.

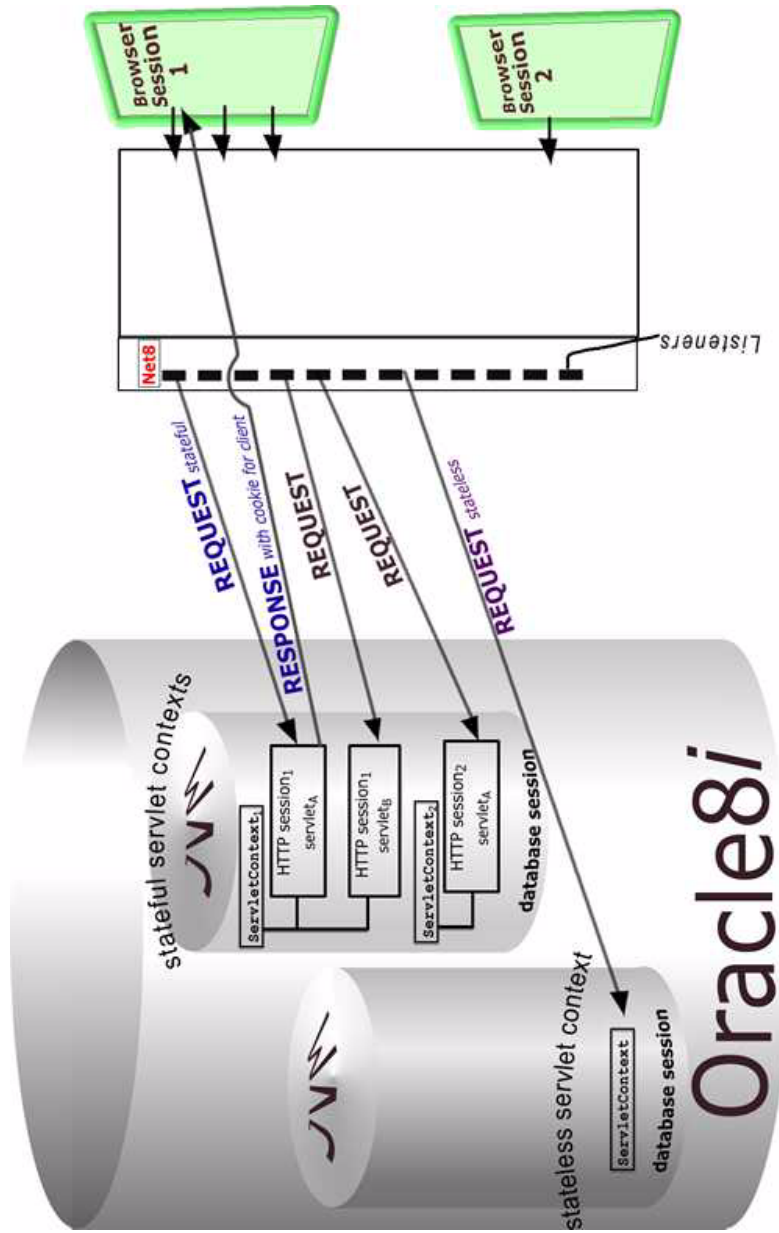
Example 4-3 Request Served by the Default Servlet

Continuing from the previous example, [4-2](#), if no match for the servlet is found, the request is served by `service root/contexts/winecellar/defaultServlet`. The default servlet is employed, and the results are sent back to the client

Database Sessions

The nature of a database session is one of the differences between OSE and other servlet engines.

Figure 4-1 Single Virtual JVM per Browser Session, Multiple HTTP Session in Database Sessions



User Identity of Database Sessions

By default, the database session is authenticated as the owner of the Web domain. All SQL access, through JDBC or SQLJ, is performed in the schema of the Web domain owner.

If you deploy a set of servlets in a domain owned by SCOTT, they access tables in SCOTT's schema, unless another schema name is prepended to the table name. For example, SYS:HTTP\$LOG. In this example, the schema SCOTT needs access rights to the other schema's table.

You can use *definer's rights* code to change this behavior. Code that executes through classes granted definer's rights, executes as SYS.

Database Sessions

HTTP clients can combine several HTTP session objects in a single database session. The HTTP session is set up for each servlet. 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, as shown in [Figure 4-1](#).

Two time-outs are associated with Web server sessions inside Oracle8i:

- the time-out of the database session itself
- the time-out of each HTTP session created in the database session

Database Session Creation, Termination, and Time-Outs

The first time an HTTP client connects to an instance of Oracle8i running OSE, a session is created inside the database. *The session is a regular database session* that runs its own virtual JVM.

Database Session Time-Out

The database session time-out causes all contained HTTP sessions to terminate. This causes all its Java state information to be discarded and all its non-committed SQL states to be rolled back.

Think of it as the *termination* of the virtual JVM executing the servlets.

The database session time-out is set by the Web service parameter, `service.globalTimeout`. The Web service object controls the connection. You

can set this parameter when you create the service, or configure it later (see [Example 4-4](#)).

Example 4-4 Create Service and Set Global Time-out

```
createservice [ -http | -iiop ] [-service className ] [-properties propGroups ]  
-root location [ -globalTimeout secs ] service
```

The time-out begins to count after an HTTP request returns. If another request comes in before the time-out expires, it will start counting again using the same value after their first request terminates.

When a database session has timed out, a browser cannot connect to it anymore, even if the browser has kept the session cookie. Instead, it is routed to a new session.

Automatic Termination of Database Sessions

A database session will automatically be terminated if there are no more HTTP session objects left. An HTTP session time-out can cause a database session to terminate.

HTTP Session Creation, Termination, and Time-Outs

An HTTP session is created when a servlet in a stateful servlet context invokes the method `getSession(true)`.

Time-out occurs when the preset value is met with no further activity on the connection. This time-out value is defined in units of seconds and is stored in the `config` object. If the database session ends, the HTTP sessions must end as well.

More than one HTTP session object can exist per database session. Only one database session exists per client, as shown in [Figure 4-1](#).

A stateful session is useful for a dialogue situation between the client and the servlet. A cookie is sent to the client as a method to keep the client information readily available to the server. A stateful session is useful during sessions when the client is engaged in a step-wise set of information exchanges, when the information in the next step is predicated by the information sent in the previous step. If the client does not support cookies, OSE uses URL rewrites.

Apache Module for Oracle OSE

This chapter contains instructions on how to configure and use OSE, a servlet server, with Apache, an HTTP Web server. You will see how this arrangement delegates the dynamic content through `mod_ose` to the OSE running in Oracle8i.

The topics we cover are the following:

- [Requirements for OSE with Apache](#)
- [Overview of `mod_ose` in Apache for OSE](#)
- [Configuring `mod_ose` in Apache for OSE](#)
- [Configuring Apache for OSE Applications](#)
- [Topology of a Site Using `mod_ose`](#)
- [`mod_ose` Installation for Pre-Installed Apache Systems](#)
- [Forwarding a URL](#)

Requirements for OSE with Apache

`mod_ose` requires the following Apache capabilities:

- Oracle HTTP server (powered by Apache)
- `mod_so` must be installed in Apache (or other support for Dynamic Shared Objects)
- `mod_mime` must be installed in Apache
- machine executing Apache must be configured as an Oracle client-side configuration
- Oracle client-side libraries must be available on `LD_LIBRARY_PATH` or equivalent

Overview of `mod_ose` in Apache for OSE

`mod_ose` is a module in the Oracle HTTP server (Apache), which serves as a conduit from Apache to OSE. In conjunction with `mod_ose`, OSE is the servlet engine for Apache, where only static pages are served by Apache and the dynamic pages are served by OSE.

You can create multi-node, in-tandem configurations. With these types of configurations, you can make a more scalable service than you can with simple stand-alone OSE or Apache components.

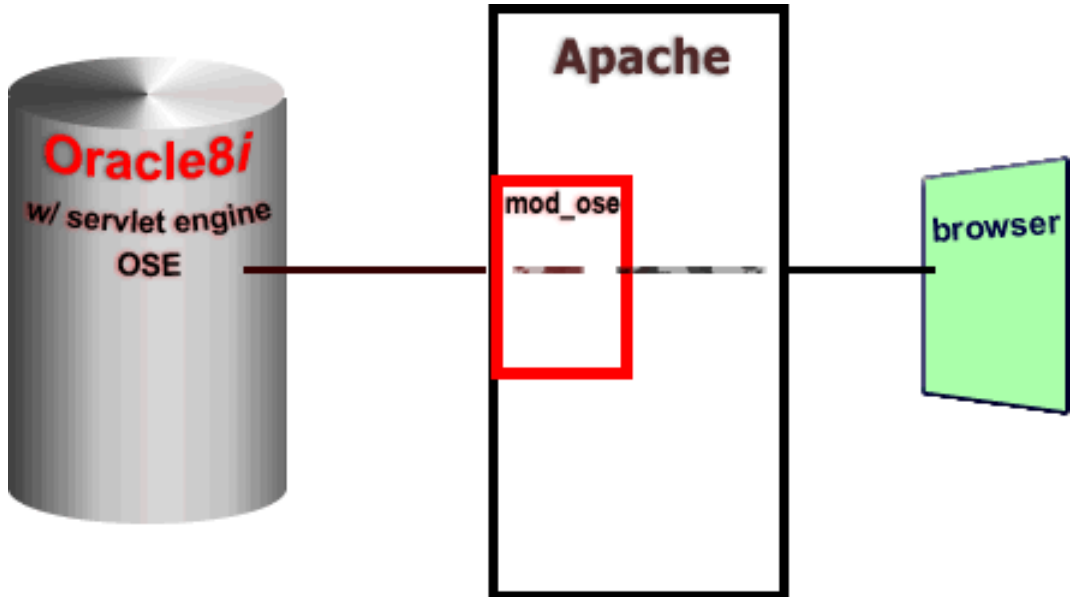
Specifically, with a multi-node arrangement there is:

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

`mod_ose` works in a network in conjunction with Net8. The [Oracle8 Net8 Administrator's Guide](#) describes, in detail load balancing, using Oracle Listener, and the fail-over configuration.

Before determining how many and which type of server your network requires, you must understand the basic configuration of the two servers, Web and servlet. [Figure 5-1](#) depicts a simple representation of the base network arrangement, in this order: browser, Apache, `mod_ose` inside Apache, Net8 (represented by the connection between `mod_ose` and the database), the database, and OSE.

Figure 5-1 Simple Topology of the Apache Web Server and Oracle's Servlet Engine (OSE), Using mod_ose



Configuring mod_ose in Apache for OSE

This section introduces mod_ose configuration steps needed to use Apache as the HTTP server and OSE as the servlet server.

The following steps define the database connection descriptors for use with mod_ose and set the Apache configuration so that the requests are sent to the appropriate database.

1. Name the Oracle Net8 connection descriptor in `tnsnames.ora` as a pointer to the OSE service (see [Example 5-1](#)).
2. Generate a Web domain configuration file using the session shell command, `exportwebdomain` (see [Example 5-2](#)).
3. Include this generated configuration file in the Apache configuration file, `httpd.conf` (see [Example 5-3](#)).
4. **If Apache was *previously installed***, define which connection to use (see [Example 5-4](#)).

Connection Descriptors and Syntax of tnsnames.ora

mod_ose uses the same configuration mechanism for finding connection descriptors that other Oracle clients, such as OCI clients, do.

Depending on the sqlnet.ora configuration, connections are defined by either:

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

or

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

Entries for the connection used by Apache must be in tnsnames.ora and be defined by the following lines:

```
<your_defined_http_connection_name> = (DESCRIPTION=
    listenerAddress
    (CONNECT_DATA=
        serviceSpec
        presentationSpec
    )
)
```

where the syntax is defined as,

```
listenerAddress := (PORT=...) (ADDRESS=(HOST=...) (PROTOCOL=...))...
serviceSpec := (SERVICE_NAME=...) [ (INSTANCE_NAME=...) ]
presentationSpec := (PRESENTATION=http://<JndiServiceName>)
```

Table 5–1 tnsnames.ora Connection Information Key Words

Key Words	Definitions
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 <i>Oracle8 Net8 Administrator's Guide</i> for details on how to set this parameters.

Table 5–1 tnsnames.ora Connection Information Key Words (Cont.)

Key Words	Definitions
serviceSpec	<p>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.</p> <p>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.</p> <p>single database instance: specify INSTANCE_NAME Indicates an specific instance within the service group should be used.</p>
presentationSpec	<p>indicates which HTTP service should be used for this connection.</p> <p><JndiServiceName> is a place holder for the name of a service in the JNDI namespace (for example, /service/ServiceName) that understands HTTP and has a Net8 end-point associated with it. See the session shell command, createwebservice.</p>

Connection(s) are named in the tnsnames.ora file, as shown in the following [example](#).

Example 5–1 tnsnames.ora Defines Entry, inst1_http, as the Apache Connection in This Scenario

```
inst1_http = (DESCRIPTION=
  (ADDRESS=(PORT=5521)(host=dlsun1609)(PROTOCOL=tcp))
  (CONNECT_DATA=
    (SERVICE_NAME=<WebServerName>)
    (PRESENTATION=http://admin)
  )
)
```

Note: <WebServerName> is the place-holder for your Web server name, such as foo.us.oracle.com.

Configuring Apache for OSE Applications

OSE was design for stateful Web applications. For these types of applications, each browser client is assigned a database session that keeps track of all stateful applications the user is executing in the domain. By default, all applications are considered stateful.

Stateful and Stateless Applications in Apache

You can configure Apache to use a stateless connection to OSE. There are certain applications for which a database session does not keep state information, with respect to the client. For these requests, each Apache process has a ***semi-persistent*** connection to a session used to process stateless connections to any client. Because each Apache worker handles only one request at a time, the stateless connection is used sequentially by different incoming clients. Stateless servlet contexts and stateless servlets must be indicated as such when published to the JNDI namespace.

The commands `ServletContext` entry and `Servlet` (see [Oracle Java Tools Reference](#) for each entry) in the session shell tool have a **`-stateless`** option that you can use to declare which application contexts or particular servlets are stateless. This information, used in conjunction with the `Export` command, simplifies management of the configuration.

Several Oracle APPS use the stateless model. It is advantageous in such cases to declare them as stateless when installed in OSE.

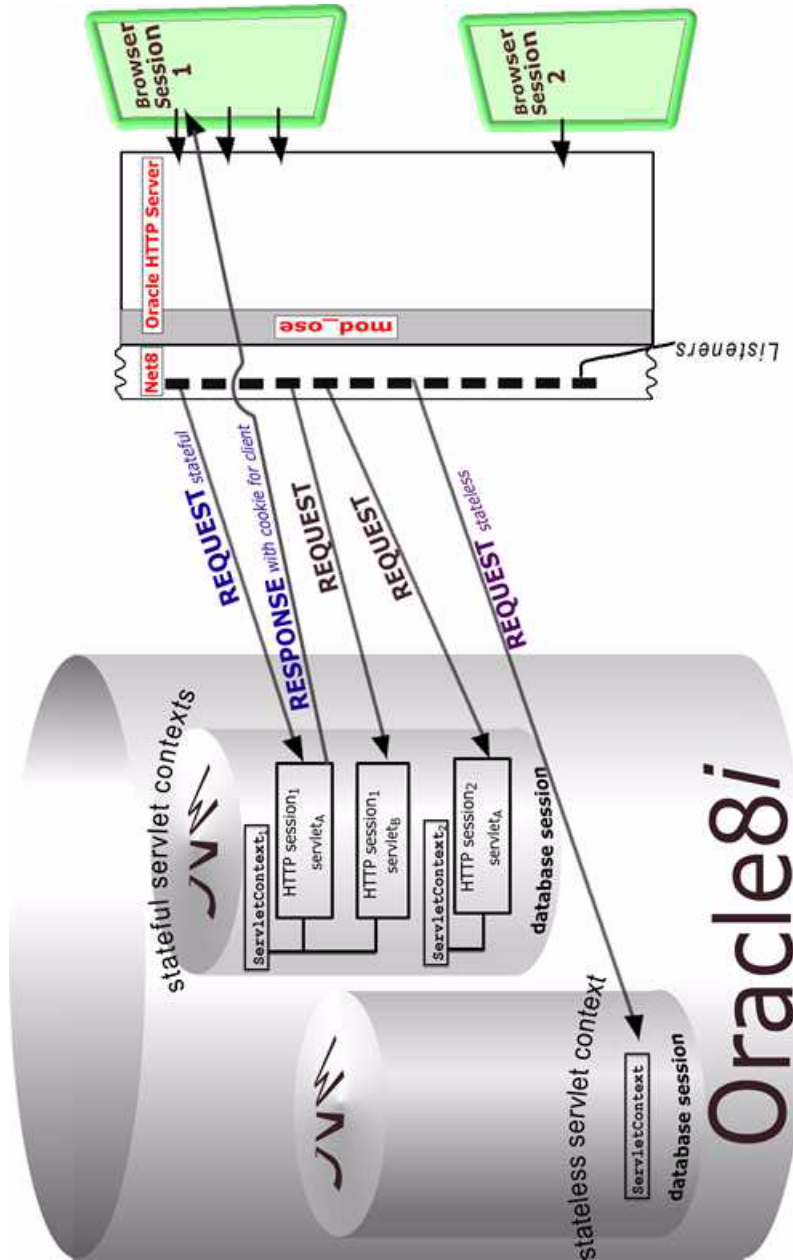
Stateful and Stateless Handlers in Apache

We specify whether a request uses the stateful or the stateless connection by specifying the particular variant of the ***handler*** to use:

```
SetHandler aurora-stateless-server
SetHandler aurora-statefull-server
SetHandler aurora-server
```

- The first handler (specifying the stateless connection) considers a request served through this channel attempting to start an `HttpSession` as an error.
- The second handler (specifying the stateful connection) allows an `HTTPSession` and requires Apache to define a separate session to serve the requests.
- The third handler uses the default mode (stateful).

Figure 5-2 Stateful Vs. Stateless Sessions



Extracting Configuration Information for Apache

A unique place specifying the configuration of Web applications running on OSE is the JNDI name space in JServer. A tool, `exportwebdomain`, extracts the information about the applications installed in a Web domain and generates the corresponding `Apache.conf` file. Include this file in the Apache main configuration file (see [Example 5-2, "Export the Web Domain Structure to an Apache CFG File"](#)).

1. Use the `exportwebdomain` command to generate the structure of a `webdomain` in a configuration file for *mod_ose*. The export utility works in two stages:
 - Generates in XML format the structure of a `webdomain` or `contexts` within a domain.
 - Applies transformations to the XML, producing configuration files specific to `mod_ose`. Generate a configuration file at the shell level.

```
sess_sh -s <ose-url> -u <user> -p <passwd>
```

```
exportwebdomain -format Apache -netservice <name in tnsname.ora> <Web domain> & > <file.conf>
```

The command connects to the server and generates the `webdomain` information in the format required by Apache and saves it into *<file.conf>*.

The parameters are defined as:

<Web domain> - The JNDI location of the Web domain to export (defaulted to *service root* for the single-domain).

<file.conf> - The name of the file to store the generated configuration, in this case, `Apache.conf`.

The options are defined as:

format type: produces output in a defined format. Use the text string `Apache` or `apache` to generate the configuration file for `mod_ose`.

See the `export` command, in the [Oracle Java Tools Reference](#), for details of all the options available. In particular, you can select to export particular contexts, or for whether `doc_root` (static pages) should be requested to OSE or will be served locally.

Example 5-2 Export the Web Domain Structure to an Apache CFG File

1. Enter:

```
exportwebdomain -format apache -netSERVICE inst1_http /webdomains & >
/apache/config/webdomains.cfg
```

The output file `webdomains.cfg` holds the Web domain configuration.

2. Include this generated file, shown in the following [example](#), at the bottom of the Apache configuration file, `httpd.conf`.

Example 5-3 *exportwebdomain Command Output Results, webdomains.cfg*

```
# Apache configuration
# Domain: /webdomains
#
<IfModule mod_ose.c>

AuroraService inst1_http
#
# Context for VPATH /context-test/
#

<Location /context-test/ >
AddHandler aurora-server snoop
</Location>

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

<Location /context-test/errors/internal >
SetHandler aurora-server
</Location>

<Location /context-test/examples/counter >
SetHandler aurora-server
</Location>

<Location /context-test/examples/snoop >
SetHandler aurora-server
</Location>

<Location /context-test/servlet/* >
SetHandler aurora-server
</Location>

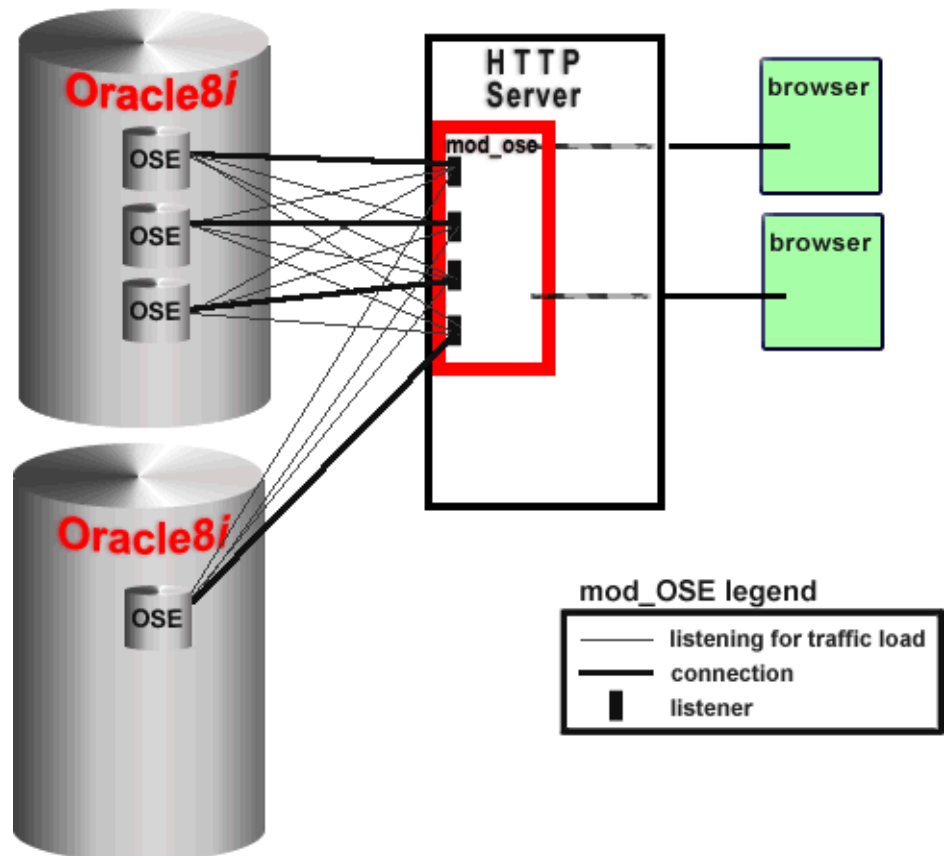
</IfModule>
#
# End of configuration
```

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.

Figure 5-3 Listeners and Load Balancing Through mod_ose, Using Net8

A Load Balancing Topology Technique



mod_ose Installation for Pre-Installed Apache Systems

If Apache is already installed on your system, you must set the configure as required by OSE. If Apache is installed by Oracle with the OSE, the installation is complete.

From the `bin` directory of your Apache installation, execute the command:

```
apxs -i -a -n mod_ose $APACHE_HOME/Apache/libexec/libjipa8i.so
```

This command copies the module to your Apache installation and makes it available by adding the corresponding `LoadModule` directive to `httpd.config`. The next time Apache is restarted, the new module will be available.

Note: The installer should make the library available in `$APACHE_HOME/Apache/libexec`.

`libjip.so`, on which `libjipa8i.so` depends, is copied to `$ORACLE_HOME/lib` by the installer.

Apache Configuration

The Apache configuration is divided in two parts:

- [Specification of OSE Service](#) (for requests)
- [Specification of Dynamic Requests](#) (URLs)

Specification of OSE Service

In this step, we configure which connection to use when communicating with OSE in the database. You can configure the OSE service in Apache only at the virtual host level. Paths in a virtual host can be routed to only one OSE server.

Use the `mod_ose` command `AuroraService`, to specify which connection description should be used to contact OSE. The syntax is:

```
<IfModule mod_ose.c>
    AuroraService connection-name
</IfModule>
```

The following configuration [example](#) define which connection to use, pointing to the `inst1_http` connection, as defined in [Example 5-1](#):

Example 5-4 *Defining the Connection Description Contacting OSE*

```
<IfModule mod_ose.c>
    AuroraService inst1_http
</IfModule>
```

If the virtual host does not define its own `AuroraService` configuration, then the default Host configuration will be used.

Specification of Dynamic Requests

The `AddHandler` and `SetHandler` commands, defined as part of `mod_mime`, specify which requests are sent to Aurora. (See "[Requirements for OSE with Apache](#)" on page 5-2, regarding `mod_mime`.)

- `AddHandler` specifies file extensions
- `SetHandler` specifies the address segment sent to OSE

Example 5-5 *AddHandler Specifies File Extensions Served by OSE*

```
AddHandler aurora-server .jsp .snoop
```

This sends stateful requests for a file with a `.jsp` *or* `.snoop` extension to OSE.

Example 5–6 SetHandler Specifies All Connection Requests, Parsing the Address Segment Sent to OSE

```
SetHandler aurora-server
```

This command must be used in conjunction with the *Location directive*, parsing the components of the virtual path served by OSE:

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

The virtual path of the request used in Apache are forwarded as *is* to the OSE. The OSE generates the passed segment of the Apache configuration string from the JNDI namespace information.

Forwarding a URL

When an HTTP request comes through Apache to `mod_ose`, the initial path information, `http://<web-domain:port>` is discarded. The remaining path is sent to OSE. After the servlet is executed, the result is returned to the client. In general, the request path initiates Apache routing a request through `mod_ose` to the servlet server.

Example 5–7 Apache Routing a Request Through `mod_ose` to OSE

If 3000 is the port on which Apache is listening and the following directive is included in the `httpd.conf` file:

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

The client asks for `http://apacheserver:3000/MyContexts/Counter`. `mod_ose` forwards the request as, `/MyContexts/Counter`, to OSE. If `/Counter` is found as a published servlet in the `/MyContexts` servlet context directory, then OSE sends a response. If not found, the default servlet returns Error 404 (Not Found) by default. You can change this behavior with the servlet management tools.

The same logic applies when processing static files.

Example 5–8 Apache and mod_ose Processing Static Files

If the client asks for, `http://apacheserver:3000/MyContexts/index.html`

Then the `/MyContexts/index.html` file must exist under the root directory, as specified in the URL., showing how you can send a path and a file name and the entire path is read.

Note: [Example 5–8](#) highlights a specific request operation detail for those who use `mod_jserv`.

OSE Server Configuration

This chapter uses examples and discussion to describe the OSE administration techniques and tools. The following topics are in this chapter:

- Setting Up OSE
- Create Services
- Create Domains
- Create Servlet Contexts
- Add Servlets

Setting Up OSE

OSE works through services that are configured to listen to end-points. In a service there are one or more domains. Domains contain one or more servlet contexts. Servlet contexts represent the Web application layer, containing servlets and various support entries.

The domain is configured to map virtual paths to servlet contexts. Servlet contexts are configured to map virtual paths to servlets.

Note: (Refer to the [Oracle Java Tools Reference](#) for detailed information on the commands used in the following sections.)

Create Services

Define the basic level of a service, the service name, property groups, and the root location, with the `createwebservice` command. If you are creating a multi-domain service, this command also defines the IP address and virtual host name. Look at any configured service structure with the `getproperties` command.

```
getproperties /service/<service_name>
```

Use the `addendpoint` command to add a new endpoint dynamically with an existing database listener. You can also create a new endpoint statically.

Create Domains

Create a Web domain with the `createwebdomain` command. A new Web domain is owned by the current schema that executed the command. Servlets contained in this domain, are executed as the domain owner. Each Web domain is initialized with a servlet context, `/default`.

Contexts Group

You can see the results of the configured domain structure by typing:

```
getproperties <domainroot>/config
```

The *context* group lists the mappings between virtual paths and contexts that handle HTTP requests. It is a list of name-to-value pairs.

- The *name* part is a virtual path.
- The *value* part is the name of a context, relative to the contexts subdirectory of the Web domain.

This is a list of virtual paths mapped to the proper servlet contexts which service HTTP requests.

MIME Group

The *MIME* group lists the extension to MIME type mappings that the Web domain supports.

Create Servlet Contexts

Create servlet contexts with the `createcontext` command. A servlet context is contained in `contexts` directory within the domain root. You can configure the servlet context to support either stateless or stateful servlets.

You can see the results of the configured servlet context structure by typing:

```
getproperties <domainroot>/contexts/<servlet_context>/config
```

You can see the virtual mapping in the servlet context config object with:

```
getproperties <domainroot>/config
```

MIME Group

The *MIME group* lists the extension to MIME type mappings that the Web server supports.

Add Servlets

Publish a servlet by name in the servlet context with the `publishservlet` command. This command can also associate a virtual path with the named servlet. You can see the results of the configured service structure by typing:

```
getproperties <domainroot>/contexts/<servlet_context>/named_servlets/<servletA>
```

You can see the virtual mapping in the servlet context config object with:

```
getproperties <domainroot>/contexts/<servlet_context>/config
```

Developer Tools and Procedures

This chapter uses examples and discussion to describe the OSE developer techniques and tools. The following topics are in this chapter:

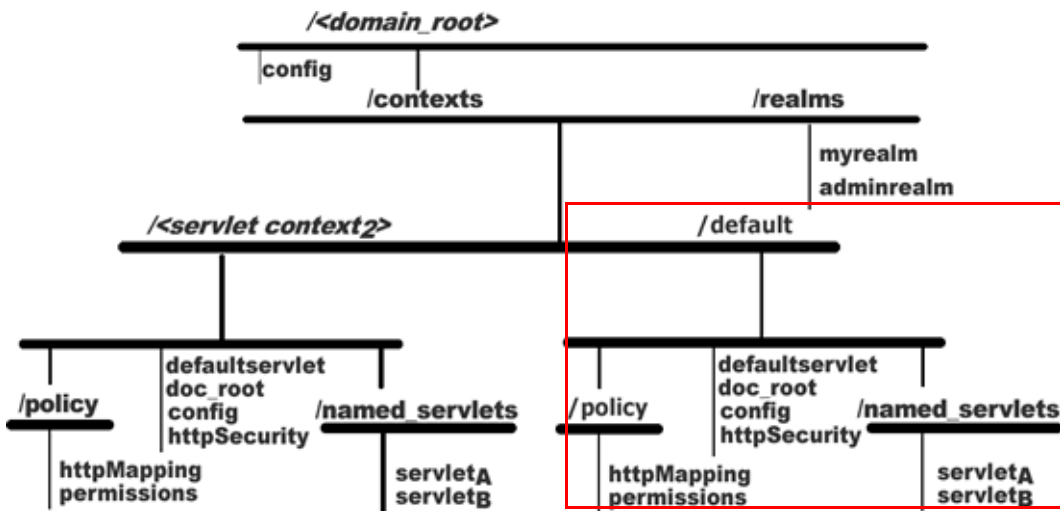
- Web Services
- Managing Servlet Contexts
- What Is in a Servlet Context?
- Managing Published Servlets
- Writing a Servlet

Web Services

A Web service contains one or more domains, depending on the configuration of the service. A newly created Web domain contains one servlet context, `default`. It serves contents for all requests that do not map to another servlet context. You can create specific servlets and servlet contexts for your own applications (see [Figure 7-1, "JNDI default Servlet Context in the JNDI Namespace"](#)).

JNDI namespace directory structure in parallel to another servlet context. The same structure requirements apply to both branches of the JNDI namespace tree. See [Figure 1-2](#) and [Figure 1-3](#) for visual examples of single and multiple domains.

Figure 7-1 JNDI default Servlet Context in the JNDI Namespace



Use the `createwebservice` command to define properties of the service. The [Oracle Java Tools Reference](#) has a detailed description of this command.

Virtual Host and IP Domains in the Web Service

In a single domain case, you do not have either an IP or a virtual host domain. The domain root is the service root. If you have a service configured to be a multi-domain service, as either a virtual host or an IP address, the domain will be rooted in the service root. If the service has both types of domains, IP address and a virtual host, the IP address is rooted in the service root, and the virtual host is

contained in the IP address domain (see [Figure 1-3, "Multi-Domain, Multi-Homed Example with Virtual Hosts In the Structure"](#)).

These domains are created by the `createwedomain` command. See the [Oracle Java Tools Reference](#) for syntax and argument details.

Changing Ports of the Web Service

Ports are associated with the corresponding presentation either statically or by using the dynamic registration commands. To support HTTPS, associate an additional SSL endpoint to the Web service.

Use the `addendpoint` command to configure the ports that are to be associated with the service (see [Oracle Java Tools Reference](#) for syntax and arguments).

Managing Servlet Contexts

This section describes how to create, change, and delete a servlet context.

Creating or Change a Servlet Context

Create new servlet contexts with the session shell `createcontext` command.

- You must have your username set with "write permission" in the containing domain's `contexts` directory to create a servlet context.
- You must have your username set with "write permission" in the containing domain's `config` object.
- The argument passed in as the virtual path gets entered into the domain's `config` object, making the new servlet context accessible to HTTP clients.
- Servlets execute using the identity of the creator of the context.

`createcontext` defines and sets up *some or all* the following properties:

- defines the virtual path associated with this context
- initializes context with a list of properties
- (*optional*) defines the location of the static pages
- (*optional*) destroys and replaces previously existing named context
- (*optional*) if flagged stateless, the context is stateless— servlets are not allowed to obtain the `HTTPSession` object

In [Example 7-1](#), "Creating a Servlet Context Named, winecellar", the servlet context handles all HTTP requests beginning with, `http://<host-name>:8080/cellar`

Example 7-1 Creating a Servlet Context Named, winecellar

1. Start the session shell tool.
2. Create a new servlet context.

Enter:

```
$createcontext -virtualpath /cellar /webdomains winecellar
                                virtualPath domainName servlet context
```

3. List the context.

Enter:

```
$ ls /webdomains/contexts
./          ../          default/    winecellar/
```

4. The context is also listed in the Web service virtual path mappings. Enter:

```
$ getgroup /webdomains/config contexts
/cellar=winecellar
```

Deleting a Servlet Context

Remove a servlet context and all servlets in the context with the session shell tool `destroycontext` command. The mapping in the domain `config` object is removed as well.

What Is in a Servlet Context?

A servlet context encapsulates the notion of a web application. A servlet context contains several entries that represent its configuration and contents. You can see the entries with the session shell tool.

Example 7–2 Servlet Context Entries of winecellar Representing Configuration and Content

1. Start the session shell.
2. `$ cd /webdomains/contexts/winecellar`
3. `$ ls`

```
./
../
config
doc_root
named_servlets/
defaultServlet
policy
httpSecurity
```

config Object

The `config` entry has a set of properties controlling the behavior of the servlet context. You can see and modify the `config` object with the session shell commands that manage object properties. See [Oracle Java Tools Reference](#) for more information.

doc_root Object

The `doc_root` object is a JNDI object representing an instance of class `SYS:oracle.aurora.namespace.filesystem.FSContextImpl` that serves as a link to the file system. The `doc_root` object is where the static content is located. It has one property known as `FSContextURL`. The file system path of the static contents of the servlet context, is stored in the `FSContextURL` property.

named_servlets Subdirectory

The `named_servlets` subdirectory contains the servlets published in the servlet context. Each published servlet is a JNDI object representing an instance of class `SYS:oracle.aurora.mts.ServletActivation`.

defaultServlet Object

If there is an entry in the servlet context named, `defaultServlet`, that servlet is used when no other servlet matches the request. If there is no entry in the servlet context for a default servlet, one is provided by the containing service.

policy Directory

The `policy` directory contains security configuration entries. See details in [Chapter 8, "Security HTTP Administration"](#).

httpSecurity

The security servlet, `httpSecurity`, functions as the first filter for all requests. This servlet handles security by raising an exception or not raising an exception and allowing the request processing to proceed. See details in [Chapter 8, "Security HTTP Administration"](#).

Servlet Context Group Parameters

This section contains the full list of groups and properties that a servlet context `config` object supports.

The group, `context.properties`, lists a set of properties that control the servlet context.

Example 7-3 Viewing and Modifying config Properties

```
$ getproperties config
--group--=context.properties
context.browse.dirs=true
context.welcome.names=index.html
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
...

$ addgroupentry config context.properties context.browse.dirs false
$ getgroup config context.properties
context.browse.dirs=false
context.welcome.names=index.html
context.accept.charset=ISO-8859-1
context.accept.language=en
context.default.languages=*
context.default.charsets=*
```

context.params

The `context.params` group is a list of arbitrary name/value pairs that are accessible at runtime by the servlet context `getAttribute` method.

Example 7-4 Servlet Context getAttribute Method

If you have the following parameters:

```
$ getgroup config context.properties
```

```
details=high
```

You can access the details parameter in a servlet as follows:

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
    ... {
    ...
    String details = getServletContext().getAttribute ("details");
    if (details.equals ("high")) {
        ...
    }
}
```

context.mime

The group `context.mime` lists the MIME types that the servlet context supports. This group has the same format as the MIME types group in the Web domain configuration.

context.servlets

When you publish servlets, you associate them with virtual paths, and you can see these virtual paths with the session shell `getgroup` command. See the section, ["JNDI default Servlet Context in the JNDI Namespace"](#) for more information.

The group `context.servlets` lists the virtual path mapping for the published servlets. It contains a list of name/value pairs, where the name part is the virtual path, and the value part is the name of a servlet to handle the virtual path. The servlet name is relative to the `named_servlets` directory of the servlet context. The name portion of the pair contains either a virtual path or a wild-card name.

context.error.uris

The group `context.error.uris` associates HTTP error codes with URIs within the servlet context that are used to report errors to the HTTP clients. Often, these URIs are handled by the default servlet and are relative to the `doc_root` parameter of the servlet context.

```
$ getgroup config 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
```

Managing Published Servlets

A servlet is published once it has a servlet activation entry in the `named_servlets` directory and the virtual path that is mapped in the servlet context `config` object.

This section discusses how to manage the servlets published in the OSE.

Servlet Classes Published in a Servlet Context

The session shell `publishservlet` and `unpublishservlet` commands modify the `named_servlets` directory and the servlet context `config` object. The servlets are created (published) or removed with their associated paths and names.

Published servlets are JNDI objects of class `SYS:oracle.aurora.mts.ServletActivation`. To be accessible from an HTTP client, servlets must be associated with a virtual path or a star-name, in their servlet context.

Example 7-5 Publishing a Servlet

```
$ publishservlet -virtualpath /tastings \  
  /webdomains/contexts/winecellar tastingServlet \  
  SCOTT:winemasters.tasting.Tasting
```

Note: The backslash is a line continuation aide. It is shell tool command line convention to assist with very long entries.

Example 7-6 Verifying the New Servlet and Virtual Path Mappings

```
# Verify the servlet is here  
$ getproperties /webdomains/contexts/winecellar/named_servlets/tastingServlet  
  servlet.class=SCOTT:winemasters.tasting.Tasting  
  
# Verify the virtual path mapping  
$ getgroup /webdomains/contexts/winecellar/config context.servlets  
  /errors/internal=internalError  
  /tastings=tastingServlet
```

A newly published servlet has only one property, the `servlet.class`. The `servlet.class` specifies the full name of the servlet class: the schema and the fully qualified path within the schema. If no schema is specified, the current schema is used.

You can add additional properties with session shell commands and access additional properties from the servlet code with the `getInitParameter()` method call.

Example 7-7 Adding Properties to the Servlet

```
$ setproperties invoker \  
"servlet.class=SCOTT:winemasters.tasting.Tasting  
details=high  
style=parker"
```

```
$ getproperties invoker  
servlet.class=SCOTT:winemasters.tasting.Tasting  
details=high  
style=parker
```

You can access the servlet properties with the following code:

```
public void doGet (HttpServletRequest request, HttpServletResponse response)  
    {  
    ...  
    String details = getServletConfig().getInitParameter ("details");  
    if (details.equals ("high")) {  
    ...  
    }  
    }
```

Writing a Servlet

Once you have configured OSE, you can begin writing and deploying servlets and JSPs.

Note: A prerequisite for this section is understanding the "Endpoints" section in chapter "Architecture".

A good reference book on servlets, such as the "Java Servlet Programming" (O'Reilly and Associates), is needed to cover the basics of building servlets.

We support the Servlet 2.2 specification.

The following steps describe the method of writing and publishing a servlet:

1. Write the servlet code.
2. Compile the servlet.
3. Load the servlet class in the database.
4. Publish the servlet.
5. Open the servlet from a Web browser.

Writing the Servlet Code

A servlet book will give a full spectrum of examples and tutorials. Please see a servlet book for assistance in writing the code.

Compiling the Servlet

To compile the servlet, you must have the following jar(s) in your CLASSPATH:

1. The jar containing the http classes:
`$(ORACLE_HOME)/jis/lib/servlet.jar`
2. If your servlet accesses the database, you also must have the JDBC classes:
`- %(ORACLE_HOME)/jdbc/lib/classes12.zip`
`- % javac -classpath .:$(ORACLE_HOME)/jis/lib/servlet.jar HelloWorld.java`

Loading the Servlet Code

Load the servlet with the `loadjava` command. See the documentation on `loadjava` in the "JServer Concepts Manual".

```
% loadjava -u scott/tiger -r HelloWorld.class
```

Publishing the Servlet

Here we publish in the default context. We are mapping the servlet to the virtualpath `/hello`.

```
$ publishservlet -virtualpath /hello /webdomains/contexts/default helloServlet SCOTT:HelloWorld
```

If you change the servlet, you must reload it in the database, but it is not necessary to republish the servlet in the Web server.

Accessing the Servlet

You can now see the servlet in a Web browser, using the URL:
`http://<host>:8080/hello`

Security HTTP Administration

Overview

This chapter contains a series of operations designed to be helpful in setting up the HTTP security for the servlet engine. The topics we cover are the following:

- Prerequisites to Web Server Security
- Authentication and Authorization
- Declaring Principals
- The Tools
- Protecting A Resource
- Declaring Permissions
- Declaring A Security Servlet
- Trouble Shooting

Prerequisites to Web Server Security

Before you define and program the security settings in the OSE, you must be proficient at setting up and manipulating the JNDI structure. Verify your knowledge against the following checklist:

- Single-Domain on page 1-8
- Multiple-Domain on page 1-9
- HTTP Virtual Path on page 2-4
- Mapping Virtual Paths to Servlet Contexts on page 2-9
- HTTP Requests on page 4-2
- Request Served by Servlet on page 2-5
- Request Served by the Default Servlet on page 4-5
- Create Service on page 7-2
- Create a Context on page 7-4

If you are unsure of any of the above topics, review the diagrams and examples describing how the OSE works with the JNDI namespace and how Web browsers interact with the OSE. A link follows each topic to a specific discussion. A complete example list is located in [Appendix B, "Examples"](#).

After you learn the basic security set up in this chapter, you can safely change the configuration of your Web service security from the default set up.

Authentication and Authorization

In HTTP security, access to a protected resource is composed of two parts: *authentication* of valid credentials and *authorization* of the user. Authentication is the validation of submitted credentials establishing that a user is known and validated by the system. Authorization is the determination of whether an authorized user is allowed to perform the requested action.

There are four stages to declare when establishing security measures:

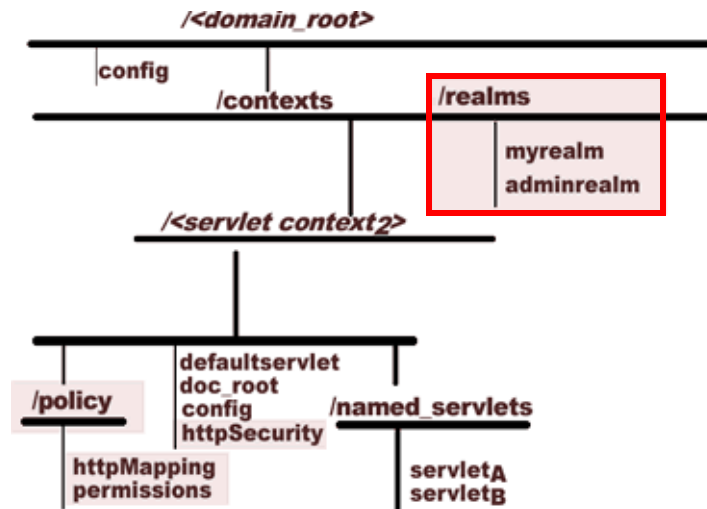
1. The principals of a service.
2. Resources as being protected and how they are to be protected.
3. Permissions of principals within the servlet context.
4. A security servlet in the `root` of the servlet context.

Use these steps to ensure the correct base information is in use defining HTTP security for your Web resources. Without any one of these steps, security will either be non-existent or will deny access to protected resources.

Declaring Principals

A principal is a generic term for either a user or a group. The realm is an object in the service, which contains the declared principals. [Figure 8-1](#) shows the position of the realm object at the top level of the Web service, at the same level as the `config` object for the service.

Figure 8–1 JNDI Structure Depicting /realms in the Domain Root



Groups

Groups contain other principals (users or other groups). Individual members of a group inherit the permission of the group object.

Users

Users are single objects. Unlike a group, there are no subsets of other principals belonging to a user.

Realms

Each realm defines a separate set of principals. The realm and its implementation are core to all of HTTP Security. There can be multiple realms within a service. The realm is the source of:

- the valid set of principals
- the types of principals that are handed to the server

Because the realm is the source of all principals, it also plays a key role in:

- what types of credentials are to be used to identify principals
- aiding the principals in managing the credentials themselves or it can defer to whatever entity that does have them
- establishing the relationships among all principals within it.

By default, there are three implementations of realms, named in HTTP Security:

- JNDI— stores all information in JNDI entries in the namespace
- DBUSER— defers to local user definitions within the database itself
- RDBMS— stores all principals and their relationships in more regular database tables

These names are just shortcuts. When instantiating the realm, use the appropriate name when declaring the realm class in the JNDI namespace.

DBUSER Considerations

The DBUSER realm is a principal definition derived from the users and roles defined within the database. There are several implications.

- No principal management is allowed through any security commands. Principal creation, deletion, and role membership is managed by the database, not security tools.
- Because all instances utilize the same source for principal definition, all instances will essentially be equivalent.
- When referring to principals, no case translations are performed to the form of the name, as presented to the Java system. This means when the database entity was created, unless the case was explicitly expressed, the entire name is uppercase. For example, SYS and PUBLIC are in uppercase, but if a user were created in lower case, such as:

```
create user "joe" identified by welcome;
```

then the username would be joe, in lowercase.

Note: The upper case/lower case distinction is important when supplying usernames and passwords from the browser.

The Tools

You set the security levels with the `realm` command, using the different flags and options. You can see the complete definition of all valid realm uses in the [Oracle Java Tools Reference](#). To see a list of the different choices, execute the `realm` command from the session shell.

Example 8-1 Executing The Realm Command Listing Valid Security Commands

```
$ realm
usage:
realm
realm list -d <webServiceRoot>
realm echo [0|1]
realm secure -s <servletContextPath>
realm map -s <servletContextPath> [- add|remove <path>] [-scheme <auth>:<realm>]
realm publish -d <webServiceRoot> [- add|remove <realmName>] [-type RDBMS | DBUSER | JNDI]
realm user -d <webServiceRoot> -realm <realmName> [- add|remove <userName> [-p <user password>]]
realm group -d <webServiceRoot> -realm <realmName> [- add|remove <groupName> [-p <group password>]]
realm parent -d <webServiceRoot> -realm <realmName> [-group <groupName> [- add|remove <principalName>]]
[-query <principalName>]
realm perm -d <webServiceRoot> -realm <realmName> -s <servletContextPath> -name <principalName> [-path <path>]
(+|-) <permList>
```

The Mechanics

Underscoring its central role, `realm` is the start of all security commands in the shell. The following sections depict example commands creating and managing realms from the shell.

Realms

To create a RDBMS realm, type:

```
realm publish -w /myService -add testRealm -type RDBMS
```

For JNDI and DBUSER, use those titles as the type argument.

To remove a realm, type:

```
realm publish -w /myService -remove testRealm
```

Realm declarations reside in the JNDI namespace. Deploying customized realms, once written, requires only slight customization of the namespace entry.

To publish a custom realm, type:

```
realm publish -w /myService -add testRealm -classname foo.bar.MyRealm
```


Principals

To create a user, type:

```
realm user -w /myService -realm testRealm1 -add user1 -p  
upswd1
```

To create a group, type:

```
realm group -w /myService -realm testRealm1 -add group1 -p gpswd1
```

In either of the above commands, if the password is left blank, the principal name is used instead.

To delete a user, type:

```
realm user -w /myService -realm testRealm1 -remove user1
```

To delete a group, type:

```
realm group -w /myService -realm testRealm1 -remove group1
```

To list users of a realm, type:

```
realm user -w /myService -realm testRealm1
```

To list groups of a realm, type:

```
realm group -w /myService -realm testRealm1
```

To add a principal to a group, type:

```
realm parent -w /myService -realm testRealm -group group1 -add user1
```

To remove a principal from a group, type:

```
realm parent -w /myService -realm testRealm -group group1 -remove user1
```

To list principals within a group, type:

```
realm parent -w /myService -realm testRealm -group group1
```

To query which groups a principal is a member of, type:

```
realm parent -w /myService -realm testRealm -q user1
```

(All realms do not support this query option.)

Note: Not all realms support editing principals. For example, DBUSER realms do not support any principal manipulation.

Details

If a service has any realms declared, they are located in a `realms` sub-context of the service. If it is a JNDI realm, there is additional sub-contexts within the `realms` context that contain its principal declarations.

If `/realms` is removed, all realm definitions are removed along with it. However any external resources (such as table entries) would remain. Using the session shell `realm` tool is much safer for efficient realm management.

Removing subcontexts of `realms` can affect any JNDI type realms. The RDBMS realm is defined to use the following tables:

- `JAVA$HTTP$REALM$PRINCIPALS`— contains all principals and an encoded form of their password
- `JAVA$HTTP$REALM$GROUPS`— contains principal/group relationships

Protecting A Resource

OSE HTTP security resource protection is local to the servlet context. To declare a resource protected, two pieces of information must be supplied, embodied in a *protection scheme*. A scheme is of the form:

```
<authType>:<realmName>
```

There are two valid authentication types:

- **Basic**— typical base64 encoding, not very secure.
- **Digest**— both parties keep the password to themselves and pass highly encrypted codes. The codes are embedded with many situation-specific values, such as time stamp, URL being requested, a secret key, and IP of the requester.

Although Digest is far more secure than Basic, not all browsers support it. From looking at typical installations, IE5 supports it; Netscape 4.7 does not.

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 /myService/contexts/myContext -a /system/* -scheme <NONE>
realm map -s /myService/myService/contexts/myContext -a /* -scheme
basic:testRealm1
```

The Mechanics

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.

Protecting paths to resources with protection schemes:

```
realm map -s /myService/contexts/myContext -a /doc/index.html -scheme
basic:testRealm1
realm map -s /myService/contexts/myContext -a /doc -scheme basic:testRealm2
realm map -s /myService/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
```

To remove the protection of a path, type:

```
realm map -s /myService/contexts/myContext -r /doc/index.html
```

To list all protected paths within a servlet context, type:

```
realm map -s /myService/contexts/myContext
```

To explicitly declare a path not to be protected, type:

```
realm map -s /myService/contexts/myContext -a /system/*
-scheme <NONE>
```

To list all protected paths within a servlet context, type:

```
realm map -s /myService/contexts/myContext
```

More Details

The JNDI entry for protection mappings is located in a subdirectory, *policy*, of the servlet context. Within that sub-context is an entry, `httpMapping`, which creates the object responsible for handling the security servlet protection mapping. By default, this object is used as an index into the `JAVA$HTTP$REALM$MAPPING$` table. The HTTP realm mapping table contains all the mapped paths. Simple JNDI entry manipulation can introduce a customized version of `HttpProtectionMapping`.

Declaring Permissions

Permissions are the most involved of all HTTP Security declarations because they tie domain-scoped entities with servlet context-scoped entities and reside in the servlet context themselves.

A permission declaration consists of several pieces:

1. service
2. realm within specified service
3. servlet context within specified service
4. principal within specified realm
5. path to which the permission is to apply
6. whether the permission is being granted or denied
7. HTTP actions being assigned

Given all the pieces that are being tied into one permission declaration, it is easy to see why these are the most complicated declarations.

Of those pieces, only the HTTP actions have not been talked about yet. HTTP security permissions concern only valid HTTP request methods: GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS.

The Mechanics

To declare a granted permission on `/foo/index.html` for `user1` for GET and POST, type:

```
realm perm -w /myService -realm testRealm1 -s
/myService/contexts/myContext -n user1 -u /foo/index.html +
get,post
```

To declare a denied permission on `/foo/*` for `user1` for `PUT` and `DELETE`, type:

```
realm perm -w /myService -realm testRealm1 -s  
/myService/contexts/myContext -n user1 -u /foo/* - put,delete
```

To clear granted permissions on `/foo/index.html` for `user1`, type:

```
realm perm -w /myService -realm testRealm1 -s  
/myService/contexts/myContext -n user1 -u /foo/index.html +
```

To list all permissions for a user, type:

```
realm perm -w /myService -realm testRealm1 -s  
/myService/contexts/myContext -n user1
```

More Details

In the *policy* subcontext of a servlet context, there will be an entry, *config*. This is the entry used to create the object responsible for all permission declaration checks.

Again, 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 main responsibility 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.

The Mechanics

To create a security servlet, type:

```
realm secure -s /myService/contexts/myContext
```

Note: The servlet is not published in `named_servlets` but within the servlet context directory itself.

Trouble Shooting

There are several layers of suspected problems to eliminate when debugging HTTP Security. This minimal checklist can help you start your trouble shooting quest.

- ❑ Check spelling (for instance, realm names, user names, or URI specifications).
- ❑ If using a DBUSER realm, check the case considerations.
- ❑ 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. This can be done 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, as 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 only be accessible to `user1` in `myRealm` using Basic authentication then there has to be the following:
 1. a realm named `myRealm` is within the domain
 2. the realm has to contain a user named, `user1`, with a known password
 3. a mapping of `/doc/index.html` or some more general path to a protection scheme `basic:myRealm` within the servlet context
 4. a security servlet declared for the servlet context
 5. a permission granting GET rights to the user `user1` for `/doc/index.html` (or a more general path)

Writing PL/SQL Servlets

This chapter describes new and changed features for release 8.1.7. 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 (iAS), 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 iAS. This module is recommended 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.

See Also: ■ For detailed information about running PL/SQL procedures over the web, see *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/initpls.sql` to load the embedded PL/SQL gateway servlet into the database server.

From the system command line, use the `sess_sh` 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:

- PL/SQL stored procedures can be accessed 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.
- *Oracle8i Java Tools Reference* for the syntax of the `sess_sh` command.

Writing Stateful PL/SQL Stored Procedures

Typically, when a PL/SQL stored procedure is run 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=wpq_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 *Using mod_plsql* book for the Oracle HTTP Server 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 can be multiple instances of the Oracle Servlet Engine, each listening for HTTP requests on a unique port. With each port, you can associate an instance of the embedded PL/SQL gateway. The port number in the the HTTP request determines which instance of the embedded PL/SQL gateway (with its associated configuration) is used.

Each instance of the embedded PL/SQL gateway is independently configurable. You can use the `DBMS_EPGC` package to set and get this configuration information.

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 user 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, you can 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. This call must be done 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 iAS. 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 iAS. 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);
```


B

Examples

This appendix contains links to all the examples in the Oracle Servlet Engine User's Guide.

Single-Domain: Port Relationship and Domain *Root* in URL 1-8

Multiple-Domain: Port Relationship and *service root* in URL 1-9

Mapping Virtual Paths to Servlet Contexts 1-12

Servlets Associated with the HTTP Virtual Path 1-12

URL Shows a Client Accessing Contents 1-13

Change the *doc_root* Path 2-4

Publish a Servlet with *publishservlet* Command 2-7

Modify a Published Servlet Properties with *setproperties* Command 2-7

HTTP Requests Finding the Servlet Context 2-9

HTTP Requests Finding the Servlet 2-9

Request Served by *defaultservlet* 2-10

HTTP Requests 4-4

Request Served by Servlet 4-4

Request Served by the Default Servlet 4-5

Create Service and Set Global Time-out 4-9

tnsnames.ora Defines Entry, *inst1_http*, as the Apache Connection in This Scenario 5-6

Export the Web Domain Structure to an Apache CFG File 5-9

exportwebdomain Command Output Results, webdomains.cfg 5-10

Defining the Connection Description Contacting OSE 5-13

AddHandler Specifies File Extensions Served by OSE 5-13

SetHandler Specifies All Connection Requests, Parsing the Address Segment Sent to OSE 5-14

Apache Routing a Request Through mod_ose to OSE 5-14

Apache and mod_ose Processing Static Files 5-15

Creating a Servlet Context Named, *winecellar* 7-5

Servlet Context Entries of *winecellar* Representing Configuration and Content 7-6

Viewing and Modifying config Properties 7-8

Servlet Context getAttribute Method 7-8

Publishing a Servlet 7-10

Verifying the New Servlet and Virtual Path Mappings 7-10

Adding Properties to the Servlet 7-11

Executing The Realm Command Listing Valid Security Commands 8-6

Symbols

":continuation indicator, 2-3
:continuation indicator, 2-3
>:continuation indicator, 2-3

Numerics

404.htm, 4-5

A

access control, 1-14
algorithm
 finding a servlet, 4-1
Apache configuration
 using tnsnames.ora, 5-5
Authentication, 8-3
authentication, 1-14
Authorization, 8-3

C

Commands
 modify entries, 3-4
configuration of your Web domain defaults, 1-6
connection name file
 tnsnames.ora, 5-5
context.params, 7-8
context.properties, 7-8
Contexts Group, 6-2
cookie, 4-2
 and time-outs, 4-9
 stateful session, 4-9

Creating new entries, 3-4
customization
 policy/map object, 8-10
 realms, 8-6

D

database, 4-3, 4-6
database access descriptors
 configuring from an application, A-3
database session timeout
 service.globalTimeout, 4-8
Database Session Timeouts
 Timeouts, 4-9
DBMS_EPGC package, A-7
 CONFIG_ERROR exception, A-8
 CREATE_DAD procedure, A-11
 CREATE_INSTANCE procedure, A-9
 DAD_NOT_FOUND exception, A-8
 DELETE_DAD_ATTRIBUTE procedure, A-12
 DELETE_GLOBAL_ATTRIBUTE
 procedure, A-10
 DROP_ALL_INSTANCES procedure, A-9
 DROP_DAD procedure, A-11
 DROP_INSTANCE procedure, A-9
 EXPORT procedure, A-13
 GET_ADMIN_LIST procedure, A-10
 GET_ALL_DAD_ATTRIBUTES
 procedure, A-12
 GET_ALL_GLOBAL_ATTRIBUTES
 procedure, A-11
 GET_DAD_ATTRIBUTE procedure, A-12
 GET_DAD_LIST procedure, A-12
 GET_GLOBAL_ATTRIBUTE procedure, A-10

- GRANT_ADMIN procedure, A-9
- IMPORT procedure, A-13
- INSTANCE_ALREADY_EXISTS exception, A-8
- INVALID_DATA_ATTRIBUTE exception, A-8
- INVALID_GLOBAL_ATTRIBUTE exception, A-8
- INVALID_PORT exception, A-8
- INVALID_USERNAME exception, A-8
- NOT_AN_ADMIN exception, A-8
- PRIVILEGE_ERROR exception, A-8
- REVOKE_ADMIN procedure, A-10
- SET_DAD_ATTRIBUTE procedure, A-11
- SET_GLOBAL_ATTRIBUTE procedure, A-10
- USER_ALREADY_EXISTS exception, A-8
- VARCHAR2_TABLE type, A-8
- Declaring Permissions, 8-10
- Default Servlet, 2-10, 4-5
- default set up
 - security change, 8-2
 - Web domain, 1-6
- defaultervlet, 2-10
 - error message, 4-5
- doc_root object, 7-6
- Document Root, 4-3

E

- embedded PL/SQL gateway, A-2
- Endpoints, 1-5
- Entries
 - bind command, 3-4
 - modify, 3-4
 - new, 3-4
- error code
 - 404, 4-5
- error message
 - posted to client, 4-5
- Example
 - Adding Properties to the Servlet, 7-11
 - Change the doc_root Path, 2-4
 - Create Service and Set Global Time-out, 4-9
 - Executing The Realm Command Listing Valid Security Commands, 8-6
 - Export the Web Domain Structure to an Apache CFG File, 5-9

- exportwebdomain Command Output Results, webdomains.cfg, 5-10
- getgroup, 7-8, 7-9
- getproperties config, 7-8
- HTTP Requests, 4-4
- HTTP requests
 - finding the servlet context, 2-9
- HTTP Requests Finding the Servlet, 2-9
- Mapping Virtual Paths to Servlet Contexts, 1-12
- Modify a Published Servlet Properties with setproperties Command, 2-7
- Multiple-Domain- Port Relationship and service root, 1-9
- Publish a Servlet with publishervlet Command, 2-7
- Request Served by defaultervlet, 2-10
- Request Served by Servlet, 4-4
- Request Served by the Default Servlet, 4-5
- Servlets Associated with the HTTP Virtual Path, 1-12
- Single-Domain- Port Relationship and Domain Root in URL, 1-8
- tnsnames.ora Defines Entry, inst1_http, as the Apache Connectio, 5-6
- URL Shows a Client Accessing Contents, 1-13
- Viewing and Modifying config Properties, 7-8
- Export Commands, 3-6

F

- fail-over configurations, 5-2, 5-5
- Forwarding a URL, 5-14

G

- get properties
 - ls, 2-7
- getSession(true), 4-8
- Group context.error.uris, 7-9
- Group context.mime, 7-9
- Group context.params, 7-8
- Group context.properties, 7-8
- Group context.servlets, 7-9
- Groups
 - property, 2-4

H

HTTP Requests

servlets, 2-8

HTTP requests

find the right servlet, 4-1

HTTP security, 8-1

authentication and authorization, 8-3

Declaring A Security Servlet, 8-12

Declaring Permissions, 8-10

Mechanics of Creating a Security Servlet, 8-12

Mechanics of protecting a resource, 8-9

The Mechanics of Permissions, 8-10

Trouble Shooting, 8-13

HTTP session

getSession(true), 4-8

http_sh

getproperties

add properties to the servlet, 2-7

publishservlet -virtualpath, 2-7

setproperties, 2-3, 2-7

httpSecurity, 7-7

I

internet newsgroups, xii

Introduction

containment hierarchy, 1-14

time-out, 1-4

invoke session shell

transportURL, 3-3

J

Java

documentation, xii

introduction, xi

Java Namespace and Directory Interface, 1-13

Java properties file, 2-3

Java References, 2-2

JDBC

web information, xii

JDK

web location, xii

JLS

web information, xii

JNDI

invoking the session shell, 3-3

navigation, 3-3

permissions, 3-2

JNDI and the Session Shell, 3-1

JNDI object

virtual path, 1-12

JServer

definition, xii

JVM

Web information, xii

L

Listeners and load balancing, 5-11

load balancing mod_ose, 5-5

Load the Servlet class in the database, 2-5

loadjava, 1-12

M

Mapping

of virtual paths, 2-9

the URI to the Virtual Path, 4-4

the Virtual Path to a Servlet, 2-9

mod_ose

fail-over configuration, 5-2

fail-over configurations, 5-5

tnsnames.ora, 5-5

Multi-Domain Web Service

requests, 2-8, 4-3

O

object of class

SYS:oracle.aurora.mts.ServletActivation, 7-7,
7-10

SYS:oracle.aurora.namespace.filesystem.FSConte
xtImpl, 7-6

Oracle schema, 1-13

Ownership, 1-13, 3-3

P

Permissions, 1-13

- Ownership, 3-3
- Schema, 1-13
- PL/SQL servlets, A-1
- Property groups, 2-4
- Protecting A Resource, 8-8
- Publish the Servlet
 - run a servlet, 2-5

R

Requests

- finding the servlet, 4-1
- Multi-Domain web service, 2-8
- Multi-Domain WebService, 4-3
- Servlet Context, 4-4
 - , 2-9
- servlets, 2-8
- Single-Domain WebService, 2-8, 4-3
- URI, 2-4

routing a request

- through mod_ose, 5-14

S

Schemas

- Web domain, 1-13
- Security, 1-14, 8-3
 - Trouble Shooting, 8-13
- security customization, 8-6, 8-10
- security defaults, 8-2
- Security HTTP, 8-1
- security servle, 7-7
- Service Configuration, 3-4
- Serving an HTTP Request, 2-8

Servlet

- default, 2-10

servlet

- default, 4-5
- finding, 4-1
- finding it, 4-1
- HTTP requests, 2-8

Servlet Context, 1-13

- config, 7-6
- creating new, 7-4
- inside, 7-6

- Management, 3-5
 - to Handle Requests, 2-9, 4-4
- Servlet Contexts, 1-11
- Servlet Management, 3-5
- servlet property
 - servlet.class, 7-10
- servlets
 - written in PL/SQL, A-1
- Session
 - created for client, 4-2
- session shell, 3-3
 - cd, 7-6
 - createcontext command, 7-4
 - createcontext -virtualpath, 7-4, 7-5
 - destroycontext command, 7-5
 - getgroup, 2-7, 7-10
 - getproperties, 2-3, 7-8, 7-11
 - getproperties
 - add properties to the servlet, 2-7
 - ls, 7-5, 7-6
 - publishservlet -virtualpath, 7-10
 - setproperties, 7-11
 - tools, 2-1
- setgroup, 2-3
- Single-Domain WebService
 - requests, 2-8, 4-3
- SQLJ
 - documentation, xii
- stateful procedures
 - written in PL/SQL, A-3
- Static Content
 - default servlet, 2-10

T

The Servlet Basics with OSE

- session shell tools, 2-1

timeout

- database session, 4-8
- HTTP session, 4-8

tnsnames.ora

- mod_ose, 5-5

transportURL

- invoke session shell, 3-3

Trouble Shooting

security, 8-13

U

URL, 2-4

V

Versions, 1-15

virtual host

multiple domains, 1-9

Virtual Path, 7-5

mapping, 2-9

virtual path, 2-4

example, 4-4

JNDI object, 1-12

OSE matches, 2-9

W

Web Domain

Configuration, 3-4

discussion, 1-6

finding to handle requests, 2-8, 4-3, 4-6

Web Service

virtual path, 7-5

Web Services, 1-5

endpoints, 1-5

multi homed, 1-5

simple, 1-5

Web sites with more Java info, xii

