Oracle® Fusion Middleware

Administering Oracle HTTP Server

12*c* (12.1.2)

E37888-13

July 2015

This document describes how to configure and use Oracle HTTP Server as a framework for hosting static pages, dynamic pages, and applications over the Web.

ORACLE

Oracle Fusion Middleware Administering Oracle HTTP Server, 12*c* (12.1.2)

E37888-13

Primary Author: Tom Pfaeffle

Contributor: Jeff Trawick,Leonard Bottleman, Ken Vincent, Maria Choudhary, Edwin Spear

Contents

2 Understanding Oracle HTTP Server Modules

3 Understanding Oracle HTTP Server Management Tools

Part II Managing Oracle HTTP Server

4 Working with Oracle HTTP Server

C Troubleshooting Oracle HTTP Server

D Configuration Files

E Property Files

F Configuring mod_security

G OHS Module Directives

Glossary

Index

Preface

This guide describes how to manage Oracle HTTP Server, including how to start and stop Oracle HTTP Server, how to manage network components, configure listening ports, and extend basic functionality using modules.

Audience

Administering Oracle HTTP Server is intended for application server administrators, security managers, and managers of databases used by application servers. This documentation is based on the assumption that readers are already familiar with Apache HTTP Server.

Unless otherwise mentioned, the information in this document is applicable when Oracle HTTP Server is installed with Oracle WebLogic Server and Oracle Fusion Middleware Control. It is assumed that readers are familiar with the key concepts of Oracle Fusion Middleware as described in the *Oracle Fusion Middleware Concepts Guide* and the *Administering Oracle Fusion Middleware*.

For information about installing Oracle HTTP Server in standalone mode, see *Installing and Configuring Oracle HTTP Server*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware 11*g* Release 1 (11.1.1) documentation set:

- *Understanding Oracle Fusion Middleware*
- *Administering Oracle Fusion Middleware*
- *High Availability Guide*

- Apache documentation included in this library

> **Note:** Readers using this guide in PDF or hard copy formats will be unable to access third-party documentation, which Oracle provides in HTML format only. To access the third-party documentation referenced in this guide, use the HTML version of this guide and click the hyperlinks.

Conventions

The following text conventions are used in this document:

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

What's New in Oracle HTTP Server 12*c* 12.1.2

The following topics introduce the new and changed features of Oracle HTTP Server and other significant changes that are described in this guide, and provides pointers to additional information. This document is the new edition of the formerly titled *Administrator's Guide for Oracle HTTP Server*.

New and Changed Features in 12*c* (12.1.2)

This section contains the following information:

New Features in 12*c* (12.1.2)

This section describes new features in this version of Oracle HTTP Server. These features include:

12*c* (12.1.2) Introduces the WebLogic Management Framework

This version of Oracle HTTP Server introduces the WebLogic Management Framework, a set of tools that leverage Oracle WebLogic 12*c* (12.1.2) interfaces to provide a simple, consistent and distributed framework for managing Oracle. For more information on the WebLogic Management Framework, see "What is the WebLogic Management Framework?" in *Understanding Oracle Fusion Middleware*.

The following changes are a result of the new framework:

- Configuration is a postinstallation task that begins with creating a domain, primarily by using Configuration Wizard. For more information, see *Installing and Configuring Oracle HTTP Server*.
- Support for remote management of OHS instances cannot be added after creation. The necessary domain type (WebLogic Server or standalone) should be chosen before installation (see Section 1.4, "Domain Types"). This is different from Oracle HTTP Server 11g where you could register an Oracle instance with a WebLogic domain at a later time to manage it by using the non-J2EE management tool.
- Configuration files for instances that are part of a WebLogic Server domain are maintained on the administration server node, not on the managed server.

- Changes made to configuration files on the managed server are not preserved when updates are made on the administration server, for example, by using Fusion Middleware Control.
- Command support for managing Oracle HTTP Server is provided primarily within WLST, instead of from the operating system shell. Existing WLST commands and new commands added in this release are applicable to Oracle HTTP Server (see Section 3.6, "Using the WebLogic Scripting Tool").
- Server-specific configuration previously maintained in opmn.xml is now configured in ohs.plugins.nodemanager.properties within the Oracle HTTP Server configuration directory.
- When starting or stopping Oracle HTTP Server, console output is now written to the Node Manager log instead a special console log file.
- Server configuration directories no longer include product code, such as Apache HTTP Server documentation, FastCGI programming libraries, or icon files used by content generated by Oracle HTTP Server. This code resides only in the product directory.
- The administration port, previously referred to as the Proxy MBean or Admin Port, is now used whether or not the instance is managed as part of a WebLogic Server domain. The port should now be limited to the loopback interface. In the previous release, the administration server would connect to the port.
- The Oracle HTTP Server MBeans, which might be visible in Fusion Middleware Control or WLST, are provided for the use of Oracle management tools. The interfaces are not supported for other use and are subject to change without notice.

OHS 12.1.2 Supports FIPS 140

Oracle HTTP Server 12.1.2 now complies with the Federal Information Processing Standard publication 140 (FIPS 140). Although the modules used in this version of Oracle HTTP Server are still undergoing their FIPS 140 validation, it uses a version of the underlying SSL libraries that has gone through formal FIPS certification.

As part of Oracle HTTP Server's FIPS 140 compliance, the mod_ossl plug-in now includes the SSLFIPS directive. This directive enables FIPS from Oracle HTTP Server configuration files by toggling the SSL library FIPS_mode flag on or off. SSLFIPS must be set in the global server context and cannot be configured with conflicting settings (for example, `SSLFIPS on` followed by `SSLFIPS off` or similar). The mode applies to all SSL library operations.

For more information on SSLFIPS, see Section G.2.6, "SSLFIPS".

Note: FIPS is available only on the UNIX/Linux platform. It is not available on the Windows platform

---

Search Capability on mod_wl_ohs Configuration Page

When configuring mod_wl_ohs by using Fusion Middleware Control, you can see a list of clusters or servers available to the selected Oracle HTTP Server instance by clicking the Search icon:

Selecting this tool displays a selection dialog box, from which you can select the cluster or server you want to use.

AutoFill Capability on mod_wl_ohs Configuration Page

You can now easily add valid WebLogic Server and endpoint locations for a specified Base URL to the Locations table on the mod_wl_ohs Configuration screen by clicking the AutoFill button. Data for any location of the same name will be updated and any new locations will be added to the table.

Significant Updates in 12*c* (12.1.2)

This section describes features that have been significantly updated from earlier versions of Oracle HTTP Server. These updates include:

- WLS Plug-in Logs Are Now Part of the Web Server Logs
- sqlnet.ora NZ Trace Logging Mechanism is No Longer Supported
- Privileged Ports on UNIX Have Different Support Implementation

WLS Plug-in Logs Are Now Part of the Web Server Logs

The WebLogic Server plug-in logs are now part of the Oracle HTTP Server error log and are prefixed with `weblogic`: to easily identify them. Hence the directives `WLLogFile` and `Debug` are deprecated. If the configuration still uses any of these directives, the following note will appear in the console log file:

```
The WLLogFile directive is ignored. The web server log file is used instead.
The Debug directive is ignored. The web server log level is used instead.
```

sqlnet.ora NZ Trace Logging Mechanism is No Longer Supported

Oracle HTTP Server no longer supports the sqlnet.ora NZ trace logging mechanism. As of version 12.1.2, you should use the new `SSLNZTraceLogLevel` directive to enable NZ trace logging using ssl.conf file. For more information, see Section G.2.9, "SSLNZTraceLogLevel".

Privileged Ports on UNIX Have Different Support Implementation

Support for listening on privileged ports on UNIX has a different implementation that does not require running any Oracle HTTP Server code as root. The User and Group directives no longer have to be configured.

Features Removed from 12*c* (12.1.2)

The following features were removed from 12.1.2:

- Integration with Oracle Web Cache
- mod_oradav
- mod_osso
- SSO Plug-ins for Third-party Web Servers
- Oracle WebLogic Server Proxy Plug-Ins for Third-party Web Servers
- SSL Protocol Version 2 and Export Ciphers

Integration with Oracle Web Cache

Oracle Web Cache is no longer included in Fusion Middleware 12c. Oracle HTTP Server support for integration with Oracle Web Cache has been removed.

mod_oradav

The mod_oradav module is no longer included with Oracle HTTP Server. Customers who require DAV support in Oracle HTTP Server must use a third-party solution, such as the open source module mod_dav.

mod_osso

The mod_osso module is no longer included with Oracle HTTP Server. Oracle WebGate is the recommended replacement. WebGate is now installed with Oracle HTTP Server.

SSO Plug-ins for Third-party Web Servers

The SSO plug-ins for IIS and iPlanet are no longer included with Oracle HTTP Server. Oracle WebGate is the recommended replacement.

Oracle WebLogic Server Proxy Plug-Ins for Third-party Web Servers

The proxy plug-ins for IIS and iPlanet are no longer included with Oracle HTTP Server. Customers who require proxy support for those web servers can use any proxy support bundled with the web server or use third-party solutions.

SSL Protocol Version 2 and Export Ciphers

Support for SSL Protocol Version 2 and export ciphers has been removed. Their use is no longer recommended for secure communication.

Part I

Understanding Oracle HTTP Server

This part presents introductory and conceptual information about Oracle HTTP Server. It contains the following chapters:

1 Introduction to Oracle HTTP Server

This chapter serves as an introduction to the Oracle HTTP Server (OHS). It describes key features of OHS and its place within the Oracle Fusion Middleware Web Tier and also provides information on the OHS directory structure, the OHS configuration files, and how to obtain OHS support.

Oracle HTTP Server is the web server component for Oracle Fusion Middleware. It provides a listener for Oracle WebLogic Server and the framework for hosting static pages, dynamic pages, and applications over the Web.

This chapter includes the following sections:

1.1 What is Oracle HTTP Server?

Oracle HTTP Server 12*c* (12.1.2) is based on Apache HTTP Server 2.2.22 infrastructure (with critical bug fixes from higher versions) and includes modules developed specifically by Oracle. The features of single sign-on, clustered deployment, and high availability enhance the operation of the Oracle HTTP Server. Oracle HTTP Server has the following components to handle client requests:

- HTTP listener, to handle incoming requests and route them to the appropriate processing utility.
- Modules (mods), to implement and extend the basic functionality of Oracle HTTP Server. Many of the standard Apache HTTP Server modules are included with Oracle HTTP Server. Oracle also includes several modules that are specific to Oracle Fusion Middleware to support integration between Oracle HTTP Server and other Oracle Fusion Middleware components.
- Perl interpreter, a persistent Perl runtime environment embedded in Oracle HTTP Server through mod_perl.

Oracle HTTP Server enables developers to program their site in a variety of languages and technologies, such as:

- Perl (through mod_perl, CGI and FastCGI)
- C and C++ (through CGI and FastCGI)
- PHP, Ruby and Python (through CGI and FastCGI)
- Oracle PL/SQL

Oracle HTTP Server can also be a proxy server, both forward and reverse. A reverse proxy enables content served by different servers to appear as if coming from one server.

Note: For more information about Fusion Middleware concepts, see *Understanding Oracle Fusion Middleware*.

1.2 Oracle HTTP Server 12*c* (12.1.2) Topologies

Oracle HTTP Server leverages the WebLogic Management Framework to provide a simple, consistent and distributed environment for administering Oracle HTTP Server, Oracle WebLogic Server, and the rest of the Fusion Middleware stack. It acts as the HTTP front-end by hosting the static content from within and by leveraging its built-in Oracle WebLogic Server Proxy Plug-In 12.1.2 to route dynamic content requests to WebLogic-managed servers. In such cases, there are multiple ways of implementing Oracle HTTP Server, depending on your requirements. The major implementations, or "topologies," are described in Table 1–1.

Table 1–1 Oracle HTTP Server Topologies

Topology	Description	For More Information
Standard Installation Topology for Oracle HTTP Server in a WebLogic Server Domain	This topology provides enhanced management capabilities through the Fusion Middleware Control and WebLogic Management Framework. A WebLogic Server domain can be scaled out to multiple physical machines and be centrally managed by the administration server. This topology is depicted in Figure 1–1.	See "Standard Installation Topology for Oracle HTTP Server in a WebLogic Server Domain" in *Installing and Configuring Oracle HTTP Server*.

Table 1–1 (Cont.) Oracle HTTP Server Topologies

Topology	Description	For More Information
Standard Installation Topology for Oracle HTTP Server in a Standalone Domain	This topology is similar to an Oracle WebLogic Server Domain topology, but does not provide an administration server or managed servers. It is useful when you do not want your Oracle HTTP Server implementation to front a Fusion Middleware domain and do not need the management functionality provided by Fusion Middleware Control. This topology is depicted in Figure 1–2.	See "Standard Installation Topology for Oracle HTTP Server in a Standalone Domain" in *Installing and Configuring Oracle HTTP Server*.
High availability implementation, with two separate hosts for Oracle HTTP Server on a Web Tier, managed by FMW Control	This topology provides a highly available Oracle Fusion Middleware deployment where each pair of components (Oracle HTTP Server and Web Logic Managed Servers) exist on different host computers. You access the system from the client tier and requests are routed, through a load balancer, to Web servers running Oracle HTTP Servers in the web tier. This topology is depicted in Figure 1–1.	See "Understanding the Oracle Fusion Middleware Standard HA Topology" in the *High Availability Guide*.
Managed Oracle HTTP Server Test Domain	This topology provides a single machine WebLogic Server Domain with an Oracle HTTP Server instance and is geared toward testing. It provides all the administrative capabilities of a full production domain but does not require an external database. The test domain cannot be scaled out to other machines and is not certified to be used in production.	See "createOHSTestDomain()" in the *WLST Command Reference for Infrastructure Components*.

Figure 1–1 Standard Installation Topology for OHS in a WebLogic Server Domain

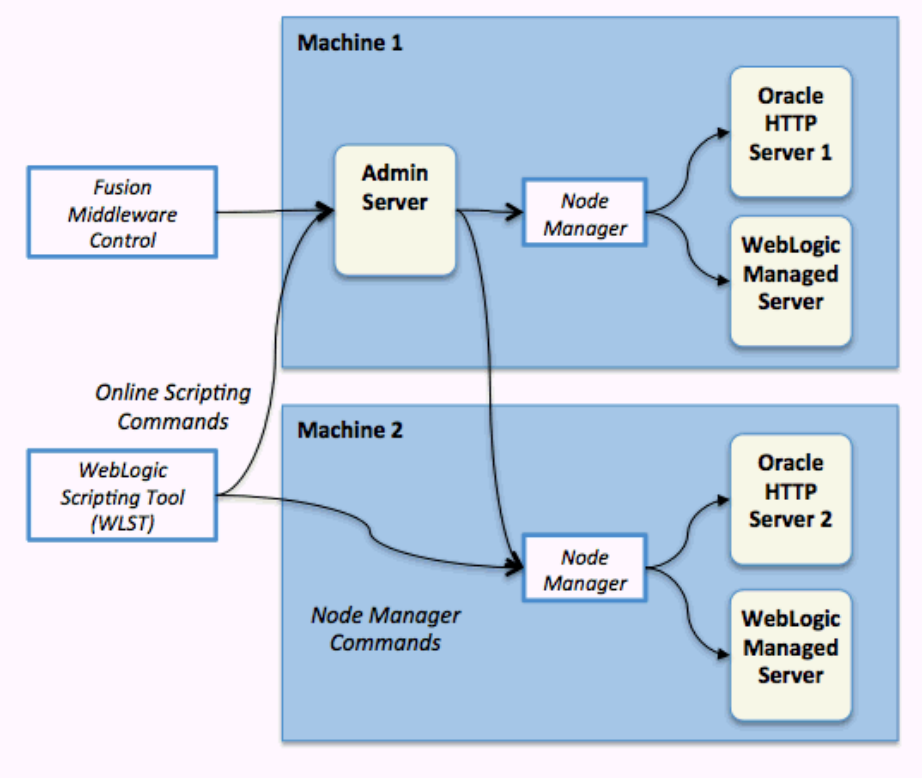
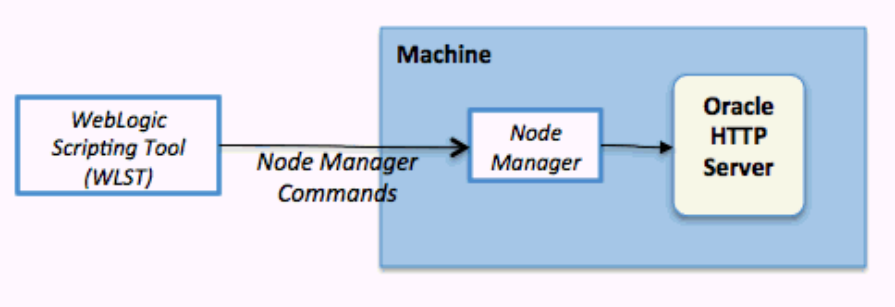


Figure 1–2 Standard Installation Topology for OHS in a Standalone Domain



1.3 Key Features of Oracle HTTP Server

The following sections describe some key features of Oracle HTTP Server:

- Section 1.3.1, "Security: Encryption with Secure Sockets Layer"
- Section 1.3.2, "Security: Single Sign-On with WebGate"
- Section 1.3.3, "URL Rewriting and Proxy Server Capabilities"
- Section 1.3.4, "PL/SQL Server Pages"
- Section 1.3.5, "Server-Side Includes"
- Section 1.3.6, "Perl"
- Section 1.3.7, "Dynamic Scripting Languages"
- Section 1.3.8, "C / C++ (CGI and FastCGI)"

- Section 1.3.9, "Load Balancing"

1.3.1 Security: Encryption with Secure Sockets Layer

Secure Sockets Layer (SSL) is required to run any website securely. Oracle HTTP Server supports SSL encryption based on patented, industry standard, algorithms. SSL works seamlessly with commonly-supported Internet browsers. Security features include the following:

- SSL hardware acceleration support uses dedicated hardware for SSL. Hardware encryption is faster than software encryption.
- Variable security per directory allows individual directories to be protected by different strength encryption.
- Oracle HTTP Server and Oracle WebLogic Server communicate using the HTTP protocol to provide both encryption and authentication. You can also enable HTTP tunneling for the T3 or IIOP protocols to provide non-browser clients access to WebLogic Server services.

See Also: *Securing Applications with Oracle Platform Security Services*

1.3.2 Security: Single Sign-On with WebGate

WebGate enables single sign-on (SSO) for Oracle HTTP Server. WebGate examines incoming requests and determines whether the requested resource is protected, and if so, retrieves the session information for the user. Through WebGate, Oracle HTTP Server becomes an SSO partner application enabled to use SSO to authenticate users, obtain their identity by using Oracle Single Sign-On, and to make user identities available to web applications accessed through Oracle HTTP Server.

See Also: *Securing Applications with Oracle Platform Security Services*

1.3.3 URL Rewriting and Proxy Server Capabilities

Active websites usually update their web pages and directory contents often, and possibly their URLs as well. Oracle HTTP Server makes it easy to accommodate the changes by including an engine that supports URL rewriting so end users do not have to change their bookmarks.

Oracle HTTP Server also supports reverse proxy capabilities, making it easier to make content served by different servers to appear from one single server.

1.3.4 PL/SQL Server Pages

PL/SQL Server Pages are similar in concept to the JavaServer Pages. The mod_plsql module enables PL/SQL to be used as the scripting language within an HTML page. PL/SQL Server Pages get translated into a stored procedure, which then uses the module to send the output to the browser.

1.3.5 Server-Side Includes

Server-Side Includes provide an easy way of adding dynamic or uniform static content across all pages on a site. It is typically used for header and footer information. Oracle HTTP Server supports special directives to enable these only for certain types of files, or for specified virtual hosts.

1.3.6 Perl

Perl is a scripting language often used to provide dynamic content. Perl scripts can either be called as a CGI program, or directly through the mod_perl module. Oracle Fusion Middleware uses Perl version 5.10.

See Also: Section 2.7, "mod_perl"

1.3.7 Dynamic Scripting Languages

Dynamic Scripting languages, for example Ruby, PHP, Python, which capable of being embedded in HTML, making them well-suited for Web development. Their scripts can be executed within Oracle HTTP Server through the built-in CGI or FastCGI modules.

1.3.8 C / C++ (CGI and FastCGI)

CGI programs are commonly used to program Web applications. Oracle HTTP Server enhances the programs by providing a mechanism to keep them active beyond the request lifecycle.

1.3.9 Load Balancing

Oracle HTTP Server includes the mod_wl_ohs module, which routes requests to Oracle WebLogic Server. The mod_wl_ohs module provides the same load balancing functionality as the Oracle WebLogic Server plug-in for Apache HTTP Server (mod_wl).

See Also: "The Dynamic Server List" in *Using Oracle WebLogic Server Proxy Plug-Ins 12.1.2*.

1.4 Domain Types

You can install Oracle HTTP Server either collocated with Oracle WebLogic Server, called a *WebLogic Server Domain* or as a *standalone domain*. You can select which environment you want to use during server configuration. Be aware that certain functionality will not be available to standalone domains.

1.4.1 WebLogic Server Domain

A WebLogic Server Domain is one configured with an administration server and managed servers. A WebLogic Server Domain contains a WebLogic Administration Server, zero or more WebLogic Managed Servers, and zero or more System Component Instances (for example, an Oracle HTTP Server instance). This type of domain provides enhanced management capabilities through the Fusion Middleware Control and WebLogic Management Framework present throughout the system. A WebLogic Server Domain can span multiple physical machines, and it is centrally managed by the administration server. Because of these properties, a WebLogic Server Domain provides the best integration between your System Components and Java EE Components.

WebLogic Server Domains support all WebLogic Management Framework tools.

Because Fusion Middleware Control provides advanced management capabilities, Oracle recommends using WebLogic Server Domain.

1.4.2 Standalone Domain

A standalone domain is a container for system components, such as Oracle HTTP Server. It has a directory structure similar to an Oracle WebLogic Server Domain, but it does not contain an Administration Server or Managed Servers. It can contain one or more instances of system components of the same type, such as Oracle HTTP Server, or a mix of system component types.

For standalone domains, the WebLogic Management Framework supports these tools:

- Node Manager
- The WebLogic Scripting Tool (WLST) commands, including:
  - `nmStart()`, `nmStop()`, `nmSoftRestart()`, and `nmKill()` that start and stop Oracle HTTP Server instance.
  - `nmConnect()` to connect to the node manager
  - `nmLog()` to get the node manager log information

  For a complete list of supported WLST Node Manager commands, see "Node Manager Commands" in "*WLST Command Reference for WebLogic Server*".

  Note: If you have a remote Oracle HTTP Server in a managed mode and another in standalone with the remote administration mode enabled, you can use WLST to perform management tasks such as SSL configuration. A vanilla Oracle HTTP Server in a standalone domain can be used only as a WebLogic Server Node Manager and for Oracle HTTP Server start/stop purposes. You can also do this by using a command-line script.

- Config Wizard
- Pack/Unpack

Generally, you would use a standalone domain when you do not want your Oracle HTTP Server implementation to front an Fusion Middleware domain and do not need the management functionality provided by Fusion Middleware Control. Nor would you use it when you want to keep Oracle HTTP Server in a "demilitarized zone" (DMZ; that is, the zone between the internal and external firewalls) and you do not want to open management ports used by the Node Manager.

1.5 Understanding Oracle HTTP Server Directory Structure

As described in Section 1.4, "Domain Types", Oracle HTTP Server domains can be either WebLogic Server or standalone. When installed, each domain has its own directory structure that contains files necessary to implement the domain type. For a complete file structure topology, see Appendix A "Understanding the Oracle HTTP Server Directory Structures" in *Installing and Configuring Oracle HTTP Server*.

1.6 Understanding Configuration Files

The Oracle HTTP Server configuration is specified through configuration files of several types, notably .conf files, similar to those used in Apache HTTP Server. This section explains the layout of the configuration file directories, mechanisms for editing the files, and more about the files themselves.

1.6.1 Staging and Run-time Configuration Directories

Two configuration directories exist for each Oracle HTTP Server instance:

- Staging directory

  DOMAIN_HOME/config/fmwconfig/components/OHS/*componentName*

- * Run-time directory

  DOMAIN_HOME/config/fmwconfig/components/OHS/instances/*componentName*

Each of the configuration directories will contain the complete OHS configuration -- httpd.conf, admin.conf, auditconfig.xml, etc.

Modifications to the configuration are made in the staging directory. (See Section 1.6.2, "Editing the Configuration") These modifications are automatically propagated to the run-time directory during the following operations:

- Oracle HTTP Server instances which are part of a WebLogic Server Domain

  Modifications are replicated to the run-time directory on the node with the managed OHS instance after changes are activated from within Fusion Middleware Control, or when the administration server initializes and prior changes need to be replicated. If communication with node manager is broken at the time of the action, replication will occur at a later time when communication has been restored.

- Standalone Oracle HTTP Server instances

  Modifications are synchronized with the run-time directory when a start, restart, or stop action is initiated. Some changes might be written to the run-time directory during domain update, but the changes will be finalized during synchronization.

Any modifications to the configuration within the run-time directory will be lost during replication or synchronization.

Note: When a standalone instance is created, the keystores directory containing a demo wallet is created only in the run-time directory.

Before creating the first new wallet for the instance, you must create a keystores directory within the staging directory.

DOMAIN_HOME/config/fmwconfig/components/OHS/*componentName*/keystores

Wallets must then be created within that keystores directory.

1.6.2 Editing the Configuration

For instances that are part of a WebLogic Server Domain, the Oracle HTTP Server configuration is managed by Fusion Middleware Control and the management infrastructure. Direct editing of the configuration in the staging directory is subject to being overwritten after subsequent management operations, including modifying the configuration in Fusion Middleware Control. For such instances, direct editing should only be performed when the administration server is inactive. When the administration server is subsequently started, the results of any manual edits will be replicated to the run-time directory on the node of the managed instance.

For standalone instances, the configuration can be edited directly within the staging directory at any time. The configuration will be activated during start, restart, or stop.

1.6.3 Configuration Files

The default Oracle HTTP Server configuration contains the files described in Appendix D, "Configuration Files".

Additional files can be added to the configuration and included in the top-level .conf file httpd.conf using the `Include` directive. For information on how to use this directive, see the Include directive documentation, at:

`http://httpd.apache.org/docs/2.2/mod/core.html#include)`

The default configuration provides an Include directive which includes all .conf files in the moduleconf/ directory within the configuration.

An Include directive should be added to an existing .conf file, usually httpd.conf, for .conf files which are not stored in the moduleconf/ directory. This may be required if the new .conf file must be included at a different configuration scope, such as within an existing virtual host definition.

1.7 Oracle HTTP Server Support

Oracle provides technical support for the following Oracle HTTP Server features and conditions:

- Modules included in the Oracle distribution. Oracle does not support modules obtained from any other source, including the Apache Software Foundation. Oracle HTTP Server will still be supported when non-Oracle-provided modules are included. If it is suspected that the non-Oracle-provided modules are contributing to reported problems, customers may be requested to reproduce the problems without including those modules.
- Problems that can be reproduced within an Oracle HTTP Server configuration consisting only of supported Oracle HTTP Server modules.
- Use of the included Perl interpreter with the supported Oracle HTTP Server configuration.

2

Understanding Oracle HTTP Server Modules

This chapter provides a high-level description of the Oracle-developed modules, or "plug-ins," used by the Oracle HTTP Server (OHS). It also provides a list of all other Apache- and third party-developed modules for OHS.

Modules (mods) extend the basic functionality of Oracle HTTP Server and support integration between Oracle HTTP Server and other Oracle Fusion Middleware components.

This chapter discusses the modules developed specifically by Oracle for Oracle HTTP Server. It includes the following sections:

2.1 List of Included Modules

This section lists all of the modules bundled with Oracle HTTP Server.

Oracle-developed Modules for Oracle HTTP Server

The following modules have been developed specifically by Oracle for Oracle HTTP Server:

- mod_certheaders
- mod_context
- mod_dms
- mod_odl
- mod_ossl
- mod_plsql

- mod_webgate
- mod_wl_ohs

Apache HTTP Server and Third-party Modules in Oracle HTTP Server

Oracle HTTP Server also includes out-of-the-box the Apache HTTP Server and third-party modules listed in Table 2–1. These modules are *not* developed by Oracle.

Table 2–1 Apache HTTP Server and Third-party Modules in Oracle HTTP Server

Module	For more information, see:
mod_actions	http://httpd.apache.org/docs/2.2/mod/mod_actions.html
mod_alias	http://httpd.apache.org/docs/2.2/mod/mod_alias.html
mod_asis	http://httpd.apache.org/docs/2.2/mod/mod_asis.html
mod_auth_basic	http://httpd.apache.org/docs/2.2/mod/mod_auth_basic.html
mod_authn_alias	http://httpd.apache.org/docs/2.2/mod/mod_authn_alias.html
mod_authn_anon	http://httpd.apache.org/docs/2.2/mod/mod_authn_anon.html
mod_authn_default	http://httpd.apache.org/docs/2.2/mod/mod_authn_default.html
mod_authn_file	http://httpd.apache.org/docs/2.2/mod/mod_authn_file.html
mod_authz_default	http://httpd.apache.org/docs/2.2/mod/mod_authz_default.html
mod_authz_groupfile	http://httpd.apache.org/docs/2.2/mod/mod_authz_groupfile.html
mod_authz_host	http://httpd.apache.org/docs/2.2/mod/mod_authz_host.html
mod_authz_user	http://httpd.apache.org/docs/2.2/mod/mod_authz_user.html
mod_autoindex	http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html
mod_cern_meta	http://httpd.apache.org/docs/2.2/mod/mod_cern_meta.html
mod_cgi	http://httpd.apache.org/docs/2.2/mod/mod_cgi.html
mod_cgid (UNIX only)	http://httpd.apache.org/docs/2.2/mod/mod_cgid.html
mod_deflate	http://httpd.apache.org/docs/2.2/mod/mod_deflate.html
mod_dir	http://httpd.apache.org/docs/2.2/mod/mod_dir.html
mod_dumpio	http://httpd.apache.org/docs/2.2/mod/mod_dumpio.html
mod_env	http://httpd.apache.org/docs/2.2/mod/mod_env.html
mod_expires	http://httpd.apache.org/docs/2.2/mod/mod_expires.html
mod_fastcgi	http://www.fastcgi.com/drupal/node/6
mod_file_cache	http://httpd.apache.org/docs/2.2/mod/mod_file_cache.html
mod_filter	http://httpd.apache.org/docs/2.2/mod/mod_filter.html
mod_headers	http://httpd.apache.org/docs/2.2/mod/mod_headers.html
mod_imagemap	http://httpd.apache.org/docs/2.2/mod/mod_imagemap.html
mod_include	http://httpd.apache.org/docs/2.2/mod/mod_include.html
mod_info	http://httpd.apache.org/docs/2.2/mod/mod_info.html
mod_log_config	http://httpd.apache.org/docs/2.2/mod/mod_log_config.html
mod_logio	http://httpd.apache.org/docs/2.2/mod/mod_logio.html

Table 2–1 (Cont.) Apache HTTP Server and Third-party Modules in Oracle HTTP Server

Module	For more information, see:
mod_mime	http://httpd.apache.org/docs/2.2/mod/mod_mime.html
mod_mime_magic	http://httpd.apache.org/docs/2.2/mod/mod_mime_magic.html
mod_negotiation	http://httpd.apache.org/docs/2.2/mod/mod_negotiation.html
mod_perl	http://perl.apache.org/
mod_proxy	http://httpd.apache.org/docs/2.2/mod/mod_proxy.html
mod_proxy_balancer	http://httpd.apache.org/docs/2.2/mod/mod_proxy_balancer.html
mod_proxy_connect	http://httpd.apache.org/docs/2.2/mod/mod_proxy_connect.html
mod_proxy_ftp	http://httpd.apache.org/docs/2.2/mod/mod_proxy_ftp.html
mod_proxy_http	http://httpd.apache.org/docs/2.2/mod/mod_proxy_http.html
mod_reqtimeout	http://httpd.apache.org/docs/2.2/mod/mod_reqtimeout.html
mod_rewrite	http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html
mod_security	http://www.modsecurity.org/documentation/ Also, for Oracle HTTP Server-specific information regarding mod_security, see Appendix F, "Configuring mod_security".
mod_setenvif	http://httpd.apache.org/docs/2.2/mod/mod_setenvif.html
mod_speling	http://httpd.apache.org/docs/2.2/mod/mod_speling.html
mod_status	http://httpd.apache.org/docs/2.2/mod/mod_status.html
mod_substitute	http://httpd.apache.org/docs/2.2/mod/mod_substitute.html
mod_unique_id	http://httpd.apache.org/docs/2.2/mod/mod_unique_id.html
mod_userdir	http://httpd.apache.org/docs/2.2/mod/mod_userdir.html
mod_usertrack	http://httpd.apache.org/docs/2.2/mod/mod_usertrack.html
mod_vhost_alias	http://httpd.apache.org/docs/2.2/mod/mod_vhost_alias.html

2.2 mod_certheaders

The mod_certheaders module enables reverse proxies that terminate Secure Sockets Layer (SSL) connections in front of Oracle HTTP Server to transfer information regarding the SSL connection, such as SSL client certificate information, to Oracle HTTP Server and the applications running behind Oracle HTTP Server. This information is transferred from the reverse proxy to Oracle HTTP Server using HTTP headers. The information is then transferred from the headers to the standard CGI environment variable. The mod_ossl module or the mod_ssl module populate the variable if the SSL connection is terminated by Oracle HTTP Server.

The mod_certheaders module also enables certain requests to be treated as HTTPS requests even though they are received through HTTP. This is done using the `SimulateHttps` directive.

`SimulateHttps` takes the container it is contained within, such as `<VirtualHost>` or `<Location>`, and treats all requests received for this container as if they were received through HTTPS, regardless of the real protocol used by the request.

See Section G.1, "mod_certheaders" for a list and description of the directives accepted by mod_certheaders.

2.3 mod_context

mod_context creates or propagates **Execution Context ID**s, or ECIDs, for requests handled by Oracle HTTP Server. If an ECID has been created for the request execution flow before it reaches Oracle HTTP Server, mod_context will make the ECID available for logging within Oracle HTTP Server and for propagation to other Fusion Middleware components, such as WebLogic Server. If an ECID has not been created when the request reaches Oracle HTTP Server, mod_context will create one.

mod_context is not configurable. It is enabled by loading it into the server with the LoadModule directive, and disabled by removing or commenting out the LoadModule directive corresponding to this module. It should always be enabled to aid with problem diagnosis.

2.4 mod_dms

mod_dms provides FMW infrastructure access to the OHS Oracle Dynamic Monitoring Service (DMS) data.

2.5 mod_odl

The mod_odl module allows Oracle HTTP Server to access Oracle Diagnostic Logging (ODL). ODL generates log messages in text or XML-formatted logs, in a format which complies with Oracle standards for generating error log messages. Oracle HTTP Server uses ODL by default.

Oracle HTTP Server complies with the Federal Information Processing Standard publication 140 (FIPS 140); it uses a version of the underlying SSL libraries that has gone through formal FIPS certification. As part of Oracle HTTP Server's FIPS 140 compliance, the mod_ossl plug-in now includes the SSLFIPS directive. For more information, see Section G.2.6, "SSLFIPS."

ODL provides the following benefits:

- The capability to limit the total amount of diagnostic information saved. You can set the level of information saved and you can specify the maximum size of the log file and the log file directory.
- When you reach the specified size, older segment files are removed and newer segment files are saved in chronological fashion.
- Components can remain active, and do not need to be shutdown, when older diagnostic logging files are deleted.

You can view log files using Fusion Middleware Control or with WLST commands, or you can download log files to your local client and view them using another tool (for example, a text edit or another file viewing utility).

For more information on using ODL with Oracle HTTP Server, see Chapter 7, "Managing Oracle HTTP Server Logs."

> **See Also:** "Managing Log Files and Diagnostic Data" in *Administering Oracle Fusion Middleware*.

2.6 mod_ossl

mod_ossl, the Oracle Secure Sockets Layer (SSL) implementation in use with the Oracle database, enables strong cryptography for Oracle HTTP Server. It is a plug-in to Oracle HTTP Server that enables the server to use SSL and is very similar to the

OpenSSL module, mod_ssl. mod_ossl supports SSL version 3 and TLS versions 1.0, 1.1. and 1.2, and is based on Certicom and RSA Security technology.

Oracle HTTP Server complies with the Federal Information Processing Standard publication 140 (FIPS 140); it uses a version of the underlying SSL libraries that has gone through formal FIPS certification. As part of Oracle HTTP Server's FIPS 140 compliance, the mod_ossl plug-in now includes the SSLFIPS directive. For more information, see Section G.2.6, "SSLFIPS."

Oracle no longer supports mod_ssl. A tool is provided to enable you to migrate from mod_ssl to mod_ossl, and convert your text certificates to Oracle wallets.

mod_ossl provides:

- Encrypted communication between client and server, using RSA or DES encryption standards.
- Integrity checking of client-server communication using MD5 or SHA checksum algorithms.
- Certificate management with Oracle wallets.
- Authorization of clients with multiple access checks, exactly as performed in mod_ssl.

mod_ossl Directives

See Section G.2 for a list and descriptions of directives accepted by mod_ossl.

> **See Also:** "Configuring SSL for the Web Tier" section of *Administering Oracle Fusion Middleware*.

2.7 mod_perl

The mod_perl module embeds the Perl interpreter into Oracle HTTP Server. This eliminates start-up overhead and enables you to write modules in Perl. Oracle Fusion Middleware uses Perl version 5.10.

The module is disabled, by default. To enable mod_perl, follow the instructions in Section 4.6.4, "Configuring mod_perl".

> **See Also:** mod_perl documentation at `http://perl.apache.org/docs/index.html`

2.7.1 Using mod_perl with a Database

This section provides information for mod_perl users working with databases. It explains how to test a local database connection and set character forms.

2.7.1.1 Using Perl to Access the Database

Perl scripts access databases using the DBI/DBD driver for Oracle. The DBI/DBD driver is part of Oracle Fusion Middleware. It calls Oracle Call Interface (OCI) to access the databases.

Once mod_perl is enabled, DBI must be enabled in the `mod_perl.conf` file to function. To enable DBI, perform the following steps:

Note: The following steps assume you are using Fusion Middleware Control and a managed server. For general information on editing a configuration file, see Section 1.6.2, "Editing the Configuration".

1. Edit the mod_perl.conf file:
   a. In Fusion Middleware Control, navigate to the Oracle HTTP Server Advanced Server Configuration page.
   b. Select the mod_perl.conf file from the menu and click **Go**.
   c. Add the following line to the mod_perl.conf file:

      ```
      PerlModule Apache::DBI
      ```

2. Click **Apply** to save the file.
3. Restart Oracle HTTP Server as described in Section 4.3.4, "Restarting Oracle HTTP Server Instances."

Place the Perl scripts that you want to run in the *DOMAIN_HOME*/config/fmwconfig/components/OHS/instances/*componentName*/cgi-bin.

Example 2–1 Using a Perl Script to Access a Database

```
#!ORACLE_HOME/perl/bin/perl -w
  use DBI;
  my $dataSource = "host=hostname.domain;sid=orclsid;port=1521";
  my $userName = "userid";
  my $password = "password";
  my $dbhandle = DBI->connect("dbi:Oracle:$dataSource", $userName, $password)
    or die "Can't connect to the Oracle Database: $DBI::errstr\n";
  print "Content-type: text/plain\n\n";
  print "Database connection successful.\n";
  ### Now disconnect from the database
  $dbhandle->disconnect
    or warn "Database disconnect failed; $DBI::errstr\n";
  exit;
```

To run the DBI scripts, the URLs would look like the following:

```
http://hostname.domain:port/cgi-bin/scriptname
http://hostname.domain:port/perl/scriptname
```

If a script specifies "`use Apache::DBI`" instead of "`use DBI`", then it can only run from the URL `http://hostname.domain:port/perl/scriptname`.

2.7.1.2 Testing a Database Connection

Example 2–2 shows a sample Perl script for testing a database connection. Replace the instance name, user ID, and password in the `connect` statement with proper values for the target database.

Example 2–2 Sample Perl Script For Testing Connection for Local Seed Database

```
use DBI;
print "Content-type: text/plain\n\n";
$dbh = DBI->connect("dbi:Oracle:instance_name", userid/password, "") ||
            die $DBI::errstr;
$stmt = $dbh->prepare("select * from emp order by empno")|| die $DBI::errstr;
$rc = $stmt->execute() || die $DBI::errstr;
```

```
while (($empno, $name) = $stmt->fetchrow()) {
   print "$empno $name\n";
}
warn $DBI::errstr if $DBI::err;
die "fetch error: " . $DBI::errstr if $DBI::err;
$stmt->finish() || die "can't close cursor";
$dbh->disconnect() || die "can't log off Oracle";
```

2.7.1.3 Using SQL NCHAR Data Types

SQL NCHAR data types (NCHAR, NVARCHAR2 and NCLOB) are reliable Unicode data types. SQL NCHAR data types enable you to store Unicode characters regardless of the database character set. The character set for those data types is specified by the national character set, which is either AL16UTF16 or UTF8.

Example 2–3 shows an example of accessing SQL NCHAR data.

Example 2–3 Sample Script to Access SQL NCHAR Data

```
# declare to use the constants for character forms
use DBD::Oracle qw(:ora_forms);
# connect to the database and get the database handle
$dbh = DBI->connect( ... );

# prepare the statement and get the statement handle
$sth = $dbh->prepare( 'SELECT * FROM TABLE_N WHERE NCOL1 = :nchar1' );

# bind the parameter of a NCHAR type
$sth->bind_param( ':nchar1', $param_1 );
# set the character form to NCHAR
$sth->func( { ':nchar1' => ORA_NCHAR } , 'set_form' );

$sth->execute;
```

As shown in Example 2–3, the `set_form` function is provided as a private function that you can invoke with the standard DBI func method. The set_form function takes an anonymous hash that enables you to set the character form for parameters.

The valid values of character form are either ORA_IMPLICIT or ORA_NCHAR. Setting the character form to ORA_IMPLICIT causes the application's bound data to be converted to the database character set, and ORA_NCHAR to the national character set. The default is ORA_IMPLICIT.

The constants are available as ora_forms in DBD::Oracle.

set_default_form sets the default character form for a database handle. The following example shows its syntax:

```
# specify the default form to be NCHAR
$dbh->func( ORA_NCHAR, 'set_default_form' );
```

This syntax causes the form of all parameters to be ORA_NCHAR, unless otherwise specified with set_form calls. Unlike the set_form function, the set_default_form functions on the database handle, so every statement from the database handle has the form of your choice.

Example 2–4 Sample for set_form

```
# a declaration example for the constants ORA_IMPLICIT and ORA_NCHAR
use DBD::Oracle qw(:ora_forms);
```

```
# set the character form for the placeholder :nchar1 to NCHAR
$sth->func( { ':nchar1' => ORA_NCHAR } , 'set_form' );

# set the character form using the positional index
$sth->func( { 2 => ORA_NCHAR } , 'set_form' );

# set the character form for multiple placeholders at once
$sth->func( { 1 => ORA_NCHAR, 2 => ORA_NCHAR } , 'set_form' );
```

2.8 mod_plsql

The mod_plsql module connects Oracle HTTP Server to an Oracle database, enabling you to create Web applications using Oracle stored procedures.

To access a Web-enabled PL/SQL application, configure a PL/SQL **database access descriptor** (DAD) for the mod_plsql module. A DAD is a set of values that specifies how the module connects to a database server to fulfill an HTTP request. Besides the connection details, a DAD contains important configuration parameters for various operations in the database and for the mod_plsql module in general. Any Web-enabled PL/SQL application which uses the PL/SQL Web ToolKit needs to create a DAD to invoke the application.

This section contains the following topics:

- Section 2.8.1, "Creating a DAD"
- Section 2.8.2, "Configuration Files for mod_plsql"
- Section 2.8.3, "Using Configuration Files and Parameters"
- Section 2.8.4, "Additional Documentation"

mod_plsql Directives

See Section G.3.1 for a list and descriptions of directives accepted by mod_plsql.

2.8.1 Creating a DAD

To create a DAD, perform the following steps:

1. Open the `dads.conf` configuration file.

   For the locations of mod_plsql configuration files, see Table 2–2.

   > **Note:** You can also open and edit the `dads.conf` file by using Oracle Fusion Middleware Control, on the Oracle HTTP Server Advanced Server Configuration page, as described in Section 4.6.6, "Modifying an Oracle HTTP Server Configuration File."

2. Add the following:

   a. The <Location> element, which defines a virtual path used to access the PL/SQL Web Application. This directive groups a set of directives that apply to the named `Location`.

      For example, the following directive defines a virtual path called /myapp that will be used to invoke a PL/SQL Web application through a URL such as `http://host:port/myapp/`.

```
<Location /myapp>
```

Note: Earlier releases of the mod_plsql module were always mounted on a virtual path with a prefix of `/pls`. This restriction is removed in later releases but might still be a restriction imposed by some earlier PL/SQL applications.

b. The `SetHandler` directive, which directs Oracle HTTP Server to enable the mod_plsql module to handle the request for the virtual path defined by the named Location:

```
SetHandler pls_handler
```

c. Additional Oracle HTTP Server directives that are allowed in the context of a `<Location>` directive. Typically, the following directives are used:

```
Order deny,allow
Allow from all
```

d. One or more specific mod_plsql directives. For example:

```
PlsqlDatabaseUsername        scott
PlsqlDatabasePassword        tiger
PlsqlDatabaseConnectString   orcl
PlsqlAuthenticationMode      Basic
```

e. The `</Location>` tag to close the `<Location>` element.

3. Save the edits.
4. Obfuscate the DAD password by running the dadTool.pl script located in the *ORACLE_HOME*/bin directory.

   See Also: "PlsqlDatabasePassword" for instructions on performing the obfuscation.

5. Restart Oracle HTTP Server as described in Section 4.3.4, "Restarting Oracle HTTP Server Instances."

You can create additional DADs by defining other uniquely named `<Location>` elements in dads.conf.

Example DADs

The following DAD connects as a specific user and has a default home page:

```
<Location /pls/mydad>
SetHandler pls_handler
Order allow,deny
Allow from All
PlsqlDatabaseUsername scott
PlsqlDatabasePassword tiger
PlsqlDatabaseConnectString prod_db
PlsqlDefaultPage scott.myapp.home
</Location>
```

The following DAD uses HTTP Basic Authentication and supports document upload/download operations:

```
<Location /pls/mydad2>
SetHandler pls_handler
```

```
Order allow,deny
Allow from All
PlsqlDatabaseConnectString prod_db2
PlsqlDefaultPage scott.myapp.my_home
PlsqlDocumentTablename scott.my_documents
PlsqlDocumentPath docs
PlsqlDocumentProcedure scott.docpkg.process_download
</Location>
```

2.8.2 Configuration Files for mod_plsql

The mod_plsql configuration parameters reside in the configuration files that are located in the configuration directory (typically, *DOMAIN_HOME*/config/fmwconfig/components/OHS/*componentName*/), as described in Table 2–2.

Table 2–2 mod_plsql Configuration Files In a System Component Instance

Directory Name	Contents
CONFIG_DIR/moduleconf	`plsql.conf`
CONFIG_DIR/mod_plsql	`dads.conf` and `cache.conf`

For information on editing these .conf files, see Section 1.6.2, "Editing the Configuration".

The mod_plsql configuration parameters are described in these sections:

- Section 2.8.2.1, "plsql.conf"
- Section 2.8.2.2, "dads.conf"
- Section 2.8.2.3, "cache.conf"

2.8.2.1 plsql.conf

The plsql.conf file resides in the *CONFIG_DIR*/moduleconf directory and Oracle HTTP Server automatically loads all .conf files under this location. The plsql.conf file contains the `LoadModule` directive to load the mod_plsql module into Oracle HTTP Server, any global settings for the mod_plsql module, and include directives for dads.conf and cache.conf.

mod_plsql Directives in plsql.conf

See Section G.3.1 for a list and description of the directives used in plsql.conf.

> **See Also:** The plsql.README file, located in *ORACLE_HOME*/ohs/mod_plsql, for a detailed description of plsql.conf.

2.8.2.2 dads.conf

The dads.conf file contains the configuration parameters for the PL/SQL database access descriptor. (See Table 2–2 for the file location.) A DAD is a set of values that specifies how the mod_plsql module connects to a database server to fulfill a HTTP request.

mod_plsql Directives in dads.conf

See Section G.3.2 for a list and description of the directives used in dads.conf

2.8.2.3 cache.conf

The cache.conf file contains the configuration settings for the file system caching functionality implemented in the mod_plsql module. This configuration file is relevant only if PL/SQL applications use the OWA_CACHE package to cache dynamically generated content in the file system.

mod_plsql Directives in cache.conf

See Appendix G.3.3 for a list and description of the directives used in cache.conf

2.8.3 Using Configuration Files and Parameters

While specifying a value for a configuration parameter, follow Oracle HTTP Server conventions for specifying values. For instance, if a value has white spaces in it, enclose the value with double quotes. For example:

```
PlsqlNLSLanguage "TRADITIONAL CHINESE_TAIWAN.UTF8"
```

Multi-line directives enable you to specify same directive multiple times in a DAD.

2.8.4 Additional Documentation

For more Oracle HTTP Server-relevant information on PL/SQL, see the following:

- *Oracle® Fusion Middleware User's Guide for mod_plsql*
- *Oracle® Fusion Middleware PL/SQL Web Toolkit Reference*

2.9 mod_webgate

The mod_webgate module enables single sign-on (SSO) for Oracle HTTP Server. WebGate examines incoming requests and determines whether the requested resource is protected, and if so, retrieves the session information for the user.

For more information, see Section 8.4.2.2, "Using WebGate to Authenticate Users" and Section 1.3.2, "Security: Single Sign-On with WebGate." For information on configuring WebGate, see "Configuring Oracle HTTP Server WebGate for Oracle Access Manager" in *Installing and Configuring Oracle HTTP Server*.

See Also: "*Securing Applications with Oracle Platform Security Services*"

2.10 mod_wl_ohs

The mod_wl_ohs module enables requests to be proxied from Oracle HTTP Server 12*c* (12.1.2) to Oracle WebLogic Server. This module is generally referred to as the Oracle WebLogic Server Proxy Plug-In.

For information about the prerequisites and procedure for configuring mod_wl_ohs, see "Configuring the Plug-In for Oracle HTTP Server" in *Using Oracle WebLogic Server Proxy Plug-Ins 12.1.2*. Directives for this module are listed in "Parameters for Oracle WebLogic Server Proxy Plug-Ins" in that document.

Note: mod_wl_ohs is similar to the mod_wl plug-in, which you can use to proxy requests from Apache HTTP Server to Oracle WebLogic server. However, while the mod_wl plug-in for Apache HTTP Server should be downloaded and installed separately, the mod_wl_ohs plug-in is bundled with Oracle HTTP Server.

3

Understanding Oracle HTTP Server Management Tools

This chapter describes the management tools available with the Oracle HTTP Server (OHS). It includes information on OHS management, how to access Fusion Middleware Control, how to access the OHS home page, and how to use the WebLogic Scripting Tool (WLST).

Oracle provides the following management tools for Oracle HTTP Server:

- The Configuration Wizard, which enables you to create and delete Oracle HTTP Server instances. For more information, see *Installing and Configuring Oracle HTTP Server*.
- Fusion Middleware Control, which is a browser-based management tool. For more information, see *Administering Oracle Fusion Middleware*.
- The WebLogic Scripting Tool, which is a command-driven scripting tool. For more information, see *Understanding the WebLogic Scripting Tool*.

> **Note:** The management tools available to your Oracle HTTP Server implementation depend on whether you have configured it in a WebLogic Server domain (with FMW Infrastructure) or in a standalone domain. For details, see Section 1.4, "Domain Types".

This chapter includes the following sections:

3.1 Overview of Oracle HTTP Server Management

The main tool for managing Oracle HTTP Server is Fusion Middleware Control, which is a browser-based tool for administering and monitoring the Oracle Fusion Middleware environment.

See Also: *Administering Oracle Fusion Middleware*

3.2 Special Note on Oracle HTTP Server Mbeans

The Oracle HTTP Server MBeans, which might be visible in Fusion Middleware Control or the WebLogic Scripting Tool (WLST) are provided for the use of Oracle management tools. The interfaces are not supported for other use and are subject to change without notice.

3.3 Accessing Fusion Middleware Control

To display Fusion Middleware Control, you enter the Fusion Middleware Control URL, which includes the name of the WebLogic Administration Server host and the port number assigned to Fusion Middleware Control during the installation. The following shows the format of the URL:

`http://`*`hostname.domain:port`*`/em`

If you saved the installation information by clicking **Save** on the last installation screen, the URL for Fusion Middleware Control is included in the file that is written to disk.

1. Display Fusion Middleware Control by entering the URL in your Web browser. For example:

   ```
   http://host1.example.com:7001/em
   ```

   The Welcome page appears.

2. Enter the Fusion Middleware Control administrator user name and password and click **Login**.

   The default user name for the administrator user is `weblogic`. This is the account you can use to log in to the Fusion Middleware Control for the first time. The `weblogic` password is the one you supplied during the installation of Fusion Middleware Control.

3.4 Accessing the Oracle HTTP Server Home Page

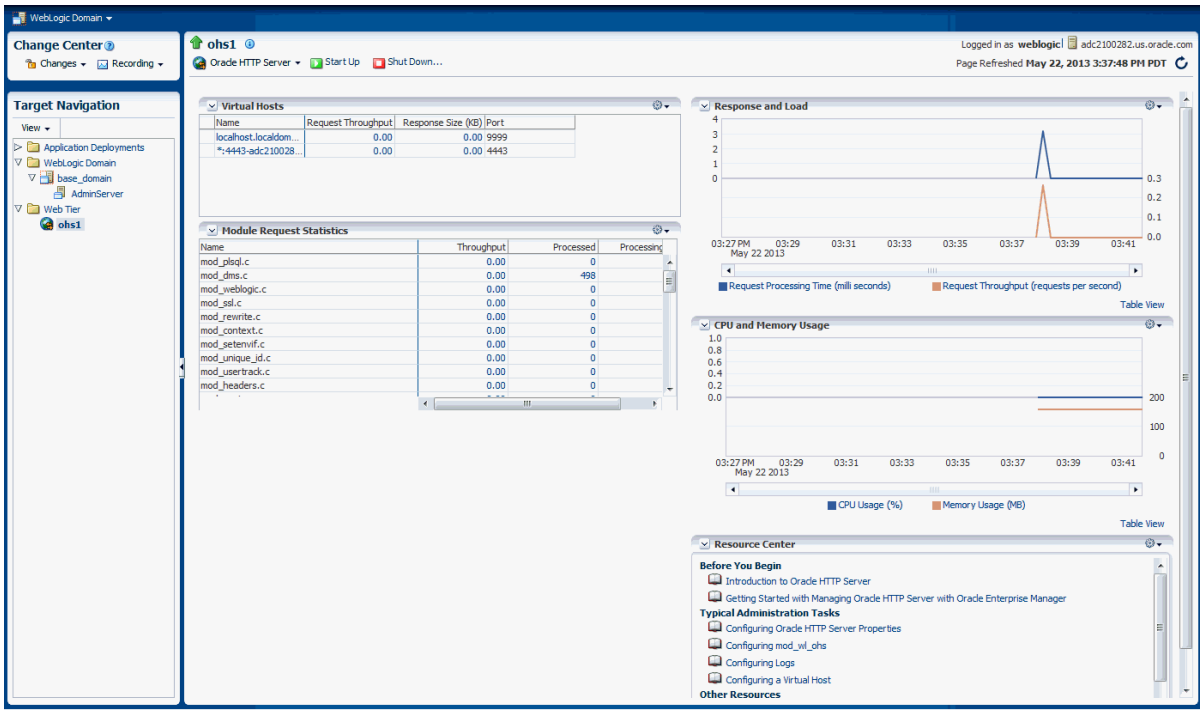
The Oracle HTTP Server Home page in Fusion Middleware Control contains menus and regions that enable you to manage the server. Use the menus for monitoring, managing, routing, and viewing general information.

3.4.1 Navigating Within Fusion Middleware Control

When you select a target, such as a WebLogic Managed Server or a component, such as Oracle HTTP Server, the target's home page is displayed in the content pane and that target's menu is displayed at the top of the page, in the context pane. For example, if you select an Oracle HTTP Server component from the Web Tier folder, the Oracle HTTP Server menu is displayed. You can also view the menu for a target by right-clicking the target in the navigation pane.

Figure 3–1 shows the target navigation pane and the home page of Oracle HTTP Server.

Figure 3–1 Oracle HTTP Server Home in Fusion MIddleware Control



The Oracle HTTP Server home page contains the following regions:

- Virtual Hosts Region: Shows the virtual hosts for Oracle HTTP Server.
- Module Request Statistics Region: Shows the modules for Oracle HTTP Server.
- Response and Load Region: Provides information such as the number of active requests, how many requests were submitted, and how long it took for Oracle HTTP Server to respond to a request. It also provides information about the number of bytes processed with the requests.
- CPU and Memory Usage Region: Shows how much CPU (by percentage) and memory (in megabytes) are being used by an Oracle HTTP Server instance.
- Resource Center: Provides links to books and topics related to Oracle HTTP Server.

> **See Also:** *Administering Oracle Fusion Middleware* contains detailed descriptions of all the items on the target navigation pane and the home page.

3.5 Using Fusion Middleware Control to Edit Configuration Files

The Advanced Server Configuration page in Fusion Middleware Control enables you to edit your Oracle HTTP Server configuration without directly editing the configuration (.conf) files (for details, see Section 4.6.6, "Modifying an Oracle HTTP Server Configuration File"). Be aware that Fusion Middleware Control and other Oracle software that manage the Oracle HTTP Server configuration might save these files in a different, equivalent format. After using the software to make a configuration change, multiple configuration files might be rewritten.

3.6 Using the WebLogic Scripting Tool

Five OHS-specific WLST commands are provided for management of Oracle HTTP Server in WebLogic Server Domains. Most are online commands, which require a connection between WLST and the administration server for the domain.

- `createOHSInstance()`
- `deleteOHSInstance()`
- `addOHSAdminProperties()`
- `addOHSNMProperties()`

One off-line command is provided for creating a domain appropriate for testing OHS:

- `createOHSTestDomain()`

You should use the `createOHSInstance()` and `deleteOHSInstance()` commands to create and delete Oracle HTTP Server instances instead of using the Configuration Wizard or offline WLST, as these custom commands perform additional error checking and, in the case of instance creation, automatic port assignment.

3.6.1 Using WLST in a Standalone Environment

An Oracle HTTP Server standalone implementation can only use WLST to start and stop the server (`nmStart()` and `nmKill()` commands; see Section 4.3, "Performing Basic OHS Tasks"). Other administration tasks are not possible. Thus in a standalone configuration, WLST offers limited benefits.

If you have a remote Oracle HTTP Server in a managed mode and another in standalone with the remote administration mode enabled, you can use WLST to perform management tasks such as SSL configuration. A vanilla Oracle HTTP Serverin a standalone domain can be used only as a WebLogic Server Node Manager and for Oracle HTTP Server start/stop purposes. You can also do this by using a command-line script.

3.6.2 Additional Information

For more information on the custom WLST commands for Oracle HTTP Server, see "Oracle HTTP Server Custom WLST Commands" in the *WLST Command Reference for Infrastructure Components*.

See Also: For more information on WLST, see *Understanding the WebLogic Scripting Tool*

Part II

Managing Oracle HTTP Server

This part presents information about management tasks for Oracle HTTP Server. It contains the following chapters:

4

Working with Oracle HTTP Server

This chapter provides information on how to work with Oracle HTTP Server (OHS). It discusses the procedures needed to configure and use OHS in your environment.

This chapter includes the following sections:

4.1 Before You Begin

Before performing any of the tasks described in this chapter, you need to do the following:

1. Install and configure Oracle HTTP Server, as described in *Installing and Configuring Oracle HTTP Server*.
2. If you run Oracle HTTP Server in a WebLogic Server Domain, start WebLogic Server as described in "Starting and Stopping Servers" in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

   Note: When you start WebLogic Server from the command line, you might encounter many warning messages scrolling by. Despite these messages, WebLogic Server should start normally.

3. Start Node Manager (required for both WebLogic and standalone domains), as described in "Using Node Manager" in *Administering Node Manager for Oracle WebLogic Server*.

   Note: As Node Manager starts, you might encounter many warnings scrolling by. You can ignore these messages.

4.2 Creating an OHS Instance

The Configuration Wizard enables you to create multiple Oracle HTTP Server instances simultaneously when you create a domain. If you are creating a WebLogic Server Domain, then you are not required to create any instances, whereas if you are creating a standalone domain, you need to create at least one Oracle HTTP Server instance. Note that, when creating a WebLogic Server domain, if you elect *not* to create any instances, a warning appears; however, you are allowed to proceed with the configuration process.

Note: If you are attempting to create an Oracle HTTP Server instance that uses a TCP port in the reserved range (typically less than 1024), then you must perform some extra configuration to allow the server to bind to privileged ports. For more information, see Section 4.3.2.4, "Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only)."

Note: Oracle Fusion Middleware contains more than one version of WLST. The WLST commands used in all procedures in this chapter will only work if you run the WLST implementation on *ORACLE_HOME*/ohs/common/bin/.

4.2.1 Creating a Managed Instance in a WebLogic Server Domain

You can create a managed Oracle HTTP Server instance in a WebLogic Server Domain by using either the custom WebLogic Scripting Tool (WLST) command `createOHSInstance()` or from Fusion Middleware Control. These procedures are described in the following sections.

Note: If you are working with a WebLogic Server Domain, you should use the Oracle HTTP Server custom WLST commands, described in Section 3.6, "Using the WebLogic Scripting Tool". These commands offer superior error checking, provide automatic port management, and so on.

4.2.1.1 Creating an Instance by Using WLST

To create an OHS instance in a WebLogic Server Domain by using WLST, do the following:

1. From the command line, launch WLST:

   Linux: `$ORACLE_HOME/ohs/common/bin/wlst.sh`

   Windows: `$ORACLE_HOME\ohs\common\bin\wlst.cmd`

2. Connect to WLST:

   - In a WebLogic Server Domain:

     ```
     > connect('loginID', 'password', '<adminHost>:<adminPort>')
     ```

     For example:

     ```
     > connect('weblogic', 'welcome1', 'abc03lll.myCo.com:7001')
     ```

3. Use the `createOHSInstance()` command, with an instance and machine name—which was assigned during domain creation—to create the instance:

```
> createOHSInstance(instanceName='ohs1', machine='abc03lll.myCo.com'
[listenPort=XXXX] [sslPort=XXXX] [adminPort=XXXX])
```

> **Note:** If Node Manager should be down, the create command will take place partially. The master copy of the config files will appear at OHS/*componentName*. Once Node Manager comes back up, the system will resync and the runtime copy of the files will appear at OHS/instances/*componentName*.

For example:

```
> createOHSInstance(instanceName='ohs1', machine='abc03lll.myCo.com')
```

> **Note:** If you do not provide port numbers, they will be assigned automatically.

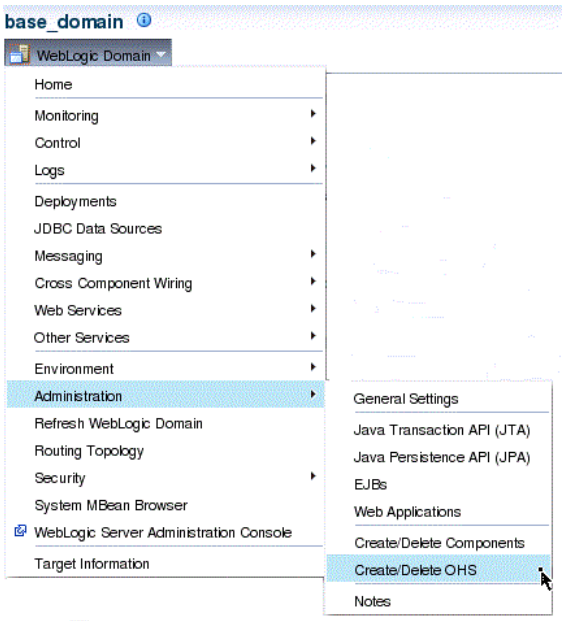
> **See also:** For information on using the WebLogic Scripting Tool (WLST), see *Understanding the WebLogic Scripting Tool*.

4.2.1.2 Creating an Instance by Using Fusion Middleware Control

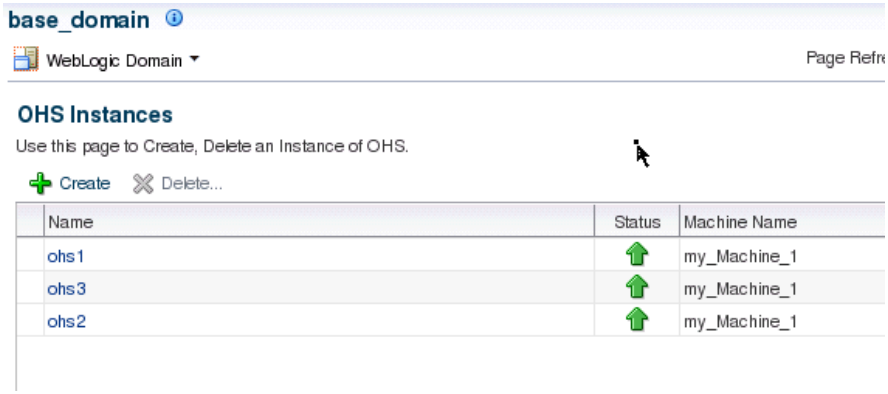
To create an Oracle HTTP Server instance in a WebLogic Server Domain by using Fusion Middleware Control, do the following:

1. Log in to Fusion Middleware Control and navigate to the system component instance home page for the WebLogic Server Domain within which you want to create the Oracle HTTP Server instance.
2. Open the WebLogic Server Domain menu and select **Administration** then **Create/Delete OHS**.

> **Note:** **Create/Delete OHS** will only appear if you have extended the domain by using the Oracle HTTP Server domain template. Otherwise, this command will not be available.

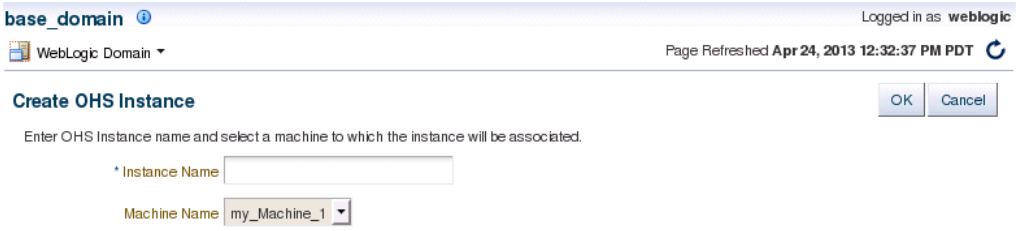


The OHS Instances page appears.



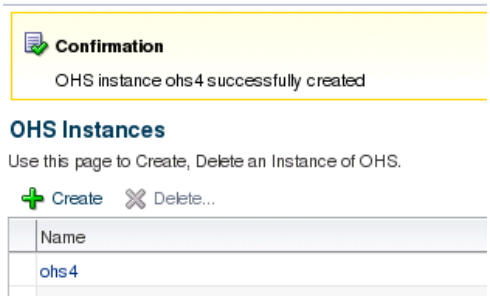
3. Click **Create**.

   The Create OHS Instance page appears.

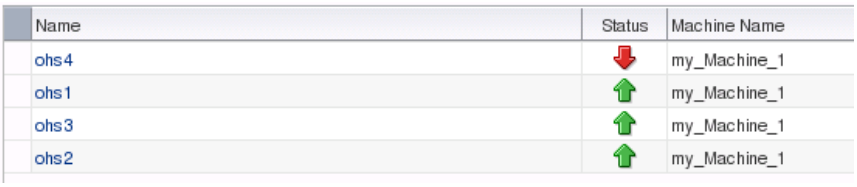


4. In **Instance Name**, type a unique name for the Oracle HTTP Server instance; for example, `ohs4`.
5. In **Machine Name**, click the drop-down control and select the machine to which you want to associate the instance.
6. Click **OK**.

The OHS Instance page reappears, showing a confirmation message and the new instance.



After creating the instance, you will note that the Column on the OHS Instances page shows a down-arrow for that instance.



This indicates that the instance is not running. For instructions on starting an instance, see Section 4.3.2, "Starting Oracle HTTP Server Instances". Once started, the arrow will point up.

4.2.1.3 Instance Provisioning

Once an instance is created, it will be provisioned within the *DOMAIN_HOME*.

- The master copy will be in:

  DOMAIN_HOME/config/fmwconfig/components/OHS/*componentName*

- The runtime will be in:

  DOMAIN_HOME/config/fmwconfig/components/OHS/instances/*componentName*

Immediately after creation, the state reported for an OHS instance will vary depending on how the instance was created:

- If `createOHSInstance()` was used, the reported state for the instance will be SHUTDOWN.
- If the Configuration Wizard was used, the reported state for the instance will be UNKNOWN.

4.2.2 Creating a Standalone Domain Instance

If you select Standalone as your domain during server configuration, the Configuration Wizard will create the domain but during this process you must create at least one Oracle HTTP Server instance. For more information, see *Installing and Configuring Oracle HTTP Server*.

4.2.2.1 Using WLST in a Standalone Domain

If your Oracle HTTP Server instance is running in a standalone domain, you can use WLST but must use the offline, or "agent", commands that route tasks through Node

Manager. The specific commands are described elsewhere in this chapter, in the context of the task they perform; however, you will need to use the `nmConnect()` command to actually connect to offline WLST. For both Linux and Windows, enter:

```
nmConnect('login','password','hostname','port','<domainName>')
```

For example:

```
nmConnect('weblogic','welcome1','localhost','5556','myDomain')
```

4.3 Performing Basic OHS Tasks

You can use Fusion Middleware Control or WebLogic Scripting Tool for the following tasks:

- Starting Oracle HTTP Server Instances
- Stopping Oracle HTTP Server Instances
- Restarting Oracle HTTP Server Instances
- Checking the Status of a Running Oracle HTTP Server Instance
- Deleting an Oracle HTTP Server Instance

About Using the WLST Commands

If you plan to use WLST, you should familiarize yourself with that tool. You should also be aware of the following:

- The online WLST commands described in this section and used in WebLogic Server Domains will only work if you run them from the WLST implementation on *ORACLE_HOME*/ohs/common/bin/wlst.sh (wlst.cmd on Windows).
- If you run a standalone version of Oracle HTTP Server, you must use the offline, or "agent", WLST commands, which are also available in *ORACLE_HOME*/ohs/common/bin/wlst.sh (wlst.cmd on Windows). These commands are described in their appropriate context.

For more information, see "Getting Started Using the Oracle WebLogic Scripting Tool (WLST)" in the *Oracle® Fusion Middleware Administrator's Guide*.

4.3.1 Understanding the PID File

When Oracle HTTP Server starts, it writes the process ID (PID) of the parent `httpd` process to the httpd.pid file located in the following directory:

```
DOMAIN_HOME/servers/<componentName>/logs
```

The process ID can be used by the administrator when restarting and terminating the daemon. If a process stops abnormally, it is necessary to stop the `httpd` child processes using the `kill` command. You must not change the default PID file name or its location.

The `PidFile` directive in httpd.conf specifies the location of the PID file; however, *you should never modify the value of this directive*.

Note: On UNIX/Linux platforms, if you edit the PidFile directive, you also have to edit the *ORACLE_HOME*/ohs/bin/apachectl file to specify the new location of the PID file.

See Also: PidFile directive in the Apache HTTP Server documentation at:

http://httpd.apache.org/docs/current/mod/mpm_common.html#pidfile

4.3.2 Starting Oracle HTTP Server Instances

This section describes how to start Oracle HTTP Server using Fusion Middleware Control and WLST.

4.3.2.1 Starting Oracle HTTP Server Instances by Using Fusion Middleware Control

To start Oracle HTTP Server using Fusion Middleware Control, navigate to the Oracle HTTP Server home page and do one of the following:

- From the Oracle HTTP Server menu:
  1. Select **Control**.
  2. Select **Start Up** from the Control menu.
- From the Target Navigation tree:
  1. Right-click the Oracle HTTP Server instance you want to start.
  2. Select **Control**.
  3. Select **Start Up** from the Control menu.
- From the page header, select **Start Up**.

  The instance will start in the state UNKNOWN.

4.3.2.2 Starting Oracle HTTP Server Instances by Using WLST

To start all Oracle HTTP Server components in a system component instance by using WLST (this procedure assumes you have created as OHS instance, as described in Section 4.2, "Creating an OHS Instance" and WLST is running), use the `start()` command in a WebLogic Server Domain or **nmStart()** for standalone domain, as shown here:

Notes: Node Manager must be running for these commands to work. If it is down, you will receive an error message.

`serverType` is required for standalone domains. If it is not included an error will be thrown referencing an inability to find `startWebLogic`.

Domain	Syntax	Example

WebLogic	`start('instanceName')`
	or
	`nmStart(serverName='name', serverType='type')`
		`start('ohs1')`
		or
		`nmStart(serverName='ohs1', serverType='OHS')`

Standalone	`nmStart(serverName='name', serverType='type')`	`nmStart(serverName='ohs1', serverType='OHS')`

If you used `createOHSInstance()` to create the instance (Section 4.2, "Creating an OHS Instance"), the state initially reported for the instance will be SHUTDOWN.

4.3.2.3 Starting Oracle HTTP Server Instances from the Command Line

You can start Oracle HTTP Server directly from a command line—that is, without launching WLST—by entering the following command:

Linux: `$DOMAIN_HOME/bin/startComponent.sh` *componentName*

Windows: `%DOMAIN_HOME%\bin\startComponent.cmd` *componentName*

For example:

```
$DOMAIN_HOME/bin/startComponent.sh ohs1
```

This command invokes WLST and tells it to run its `start()` command.

After a few seconds, you will be prompted for your Node Manager password. Type that and press Enter.

```
Successfully started server componentName...
Successfully disconnected from Node Manager...

Exiting WebLogic Scripting Tool.
```

You can avoid having to enter your Node Manager password every time you launch the server with `startComponent.sh/.cmd` by starting it with the `storeUserConfig` option for the first time. Do the following:

1. At the prompt, enter the following command:

   ```
   $DOMAIN_HOME/bin/startComponent.sh componentName storeUserConfig
   ```

   The system will prompt for your Node Manager password.

2. Type the password and press Enter.

   The system responds with this message:

   ```
   Creating a key file can reduce the security of your system if it is not a
   secured location after it is created. Do you want to create the key file? y or
   n.
   ```

3. Type `y` to store your Node manager password. When you subsequently use this command, you will not need to enter a password.

4.3.2.4 Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only)

> **WARNING: When this procedure is completed, *any* Oracle HTTP Server processes running from this Oracle Home as a user in the same group will be able to bind to privileged ports.**

On a UNIX system, TCP ports in a reserved range (typically less than 1024) can only be bound by processes with root privilege. Oracle HTTP Server always runs as a non-root user; that is, the user who installed Oracle Fusion Middleware. On UNIX, special configuration is required to allow Oracle HTTP Server to bind to privileged ports.

To enable Oracle HTTP Server to listen on a port in the reserved range (for example, the default port 80 or port 443) as a process without root privilege, use the following one-time setup on each Oracle HTTP Server machine:

1. As the same user who will start Oracle HTTP Server, create a temporary cap.ora file by entering the following:

```
echo `id -ng`: bind  > /tmp/cap.ora
```

> **Note:** The next steps must be performed as the root user. If you do not have root access, have the system administrator perform these steps.

2. Update the *ORACLE_HOME*/oracle_common/bin/hasbind file by performing the following steps:

   a. Change ownership of the file to root:

   ```
   chown root $ORACLE_HOME/oracle_common/bin/hasbind
   ```

   b. Change the permissions on the file as follows:

   ```
   chmod 4755 $ORACLE_HOME/oracle_common/bin/hasbind
   ```

3. Generate the /etc/cap.ora file by performing the following steps:

   a. If /etc/cap.ora *does not* exist, copy the temporary cap.ora file you created in step 1 to the /etc/ directory:

   ```
   cp /tmp/cap.ora /etc/cap.ora
   ```

   If /etc/cap.ora does exist, append the contents of the temporary file you created in step 1 to the existing /etc/cap.ora file:

   ```
   cat /tmp/cap.ora >> /etc/cap.ora
   ```

   b. Change the permissions on the /etc/cap.ora file as follows:

   ```
   chmod 644 /etc/cap.ora
   ```

   c. Change ownership of the file to root:

   ```
   chown root /etc/cap.ora
   ```

   The steps that require root permissions are now complete.

4. If you prefer, remove the temporary cap.ora you created in step 1:

```
rm /tmp/cap.ora
```

5. Modify the port settings for Oracle HTTP Server as described in Section 6.4, "Managing Ports".

6. Start (or restart) the instance by using any of the start-up methods described in Section 4.3.2, "Starting Oracle HTTP Server Instances".

4.3.3 Stopping Oracle HTTP Server Instances

This section describes how to stop Oracle HTTP Server using Fusion Middleware Control. Be aware that other services might be impacted when Oracle HTTP Server is stopped.

4.3.3.1 Stopping Oracle HTTP Server Instances by Using Fusion Middleware Control

To stop Oracle HTTP Server using Fusion Middleware Control, navigate to the Oracle HTTP Server home page and do one of the following:

- From the Oracle HTTP Server menu:
  1. Select **Control**.
  2. Select **Shut Down** from the Control menu.
- From the Target Navigation tree:
  1. Right-click the Oracle HTTP Server component you want to stop.
  2. Select **Control**.
  3. Select **Shut Down** from the Control menu.
- From the page header, select **Shut Down**.

4.3.3.2 Stopping Oracle HTTP Server Instances by Using WLST

To stop Oracle HTTP Server by using WLST, from within the scripting tool, use one of the following commands:

Notes: Node Manager must be running for these commands to work. If it is down, you will receive an error message.

`serverType` is required for standalone domains. If it is not included an error will be thrown referencing an inability to find `startWebLogic`

Domain	Syntax	Example
WebLogic	`shutdown('serverName')`	`shutdown('ohs1')`
Standalone	`nmKill(serverName='serverName', serverType='type')`	`nmKill(serverName='ohs1', serverType='OHS')`

WARNING: If you run shutdown() without specifying any parameters, WLS will terminate and boot you out of WLST. Oracle HTTP Server will continue running.

4.3.3.3 Stopping Oracle HTTP Server Instances from the Command Line

You can stop Oracle HTTP Server directly from a command line—that is, without launching WLST—by entering the following command:

```
$DOMAIN_HOME/bin/stopComponent.sh componentName
```

For example:

```
$DOMAIN_HOME/bin/stopComponent.sh ohs1
```

This command invokes WLST and tells it to run its `shutdown()` command.

After a few seconds, you will be prompted for your Node Manager password. Type that and press Enter. Once the server is stopped, the system will respond:

```
Successfully killed server componentName...
```

```
Successfully disconnected from Node Manager...

Exiting WebLogic Scripting Tool.
```

4.3.4 Restarting Oracle HTTP Server Instances

Restarting Oracle HTTP Server causes the Apache parent process to advise its child processes to exit after their current request (or to exit immediately if they are not serving any requests). Upon restarting, the parent process re-reads its configuration files and reopens its log files. As each child process exits, the parent replaces it with a child process from the new generation of the configuration file, which begins serving new requests immediately.

The following sections describe how to restart Oracle HTTP Server using by Fusion Middleware Control and the WLST.

4.3.4.1 Restarting Oracle HTTP Server Instances by Using Fusion Middleware Control

To restart OHS using Fusion Middleware Control, navigate to the Oracle HTTP Server home page and do one of the following:

- From the Oracle HTTP Server menu:
  1. Select **Control**.
  2. Select **Restart** from the Control menu.
- From the Target Navigation tree:
  1. Right-click the OHS instance you want to stop.
  2. Select **Control**.
  3. Select **Restart** from the Control menu.

4.3.4.2 Restarting Oracle HTTP Server Instances by Using WLST

To restart OHS by using WLST, use the `softRestart()` command. From within the scripting tool, enter one of the following commands:

Notes: Node Manager must be running for these commands to work. If it is down, you will receive an error message.

All parameters are required for standalone domains. If they are not included, an error will be thrown referencing an inability to find `startWebLogic`.

Domain	Syntax	Example
WebLogic	`softRestart('serverName')`	`softRestart('ohs1')`
Standalone	`nmSoftRestart(serverName='name', serverType='type')`	`nmSoftRestart(serverName='ohs1', serverType='OHS')`

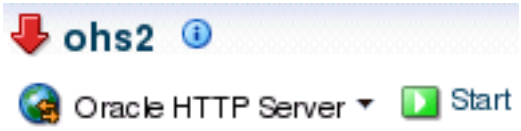
4.3.5 Checking the Status of a Running Oracle HTTP Server Instance

This section describes how to check the status of a running Oracle HTTP Server instance. You can check this information from either Fusion Middleware Control or by using WLST.

4.3.5.1 Checking Server Status by Using Fusion Middleware Control

An up or down arrow in the top left corner of any Oracle HTTP Server page's header indicates whether the selected server instance is running. This image shows the up arrow, indicating that the server instance, in this case, "ohs2", is running:

This image shows a down arrow, indicating that the server instance, in this case, "ohs2", is not running:



4.3.5.2 Checking Server Status by Using WLST

In a WebLogic Server Domain, if you used `createOHSInstance()` to create the Oracle HTTP Server instance, its initial state (that is, before starting it) will be SHUTDOWN.

If you used the Configuration Wizard to generate the instance (both WebLogic Server Domain and standalone domain), its initial state (that is, before starting) will be UNKNOWN.

To check the status of a running Oracle HTTP Server instance by using WLST, from within the scripting tool, enter the following:

Notes: Node Manager must be running for these commands to work. If it is down, you will receive an error message. If Node Manager goes down in a WebLogic Server Domain, the state will be returned as UNKNOWN, regardless of the real state of the instance. Additionally `state()` does not inform you that it cannot connect to Node Manager.

Unlike other WLST commands, `state()` will not tell you when Node Manager is down so there is no way to distinguish an instance that truly is in state UNKNOWN as opposed to Node Manager simply being down.

All parameters are required for standalone domains. If they not included an error will be thrown referencing an inability to find `startWebLogic`.

Domain	Syntax	Example
WebLogic	`state('`*`serverName`*`')`	`state('ohs1')`
Standalone	`nmServerStatus(serverName='`*`name`*`', serverType='`*`type`*`')`	`nmServerStatus(serverName='ohs1', serverType='OHS')`

Note: This command does not distinguish between non-existent components and real components in state UNKNOWN. Thus, if you enter a non-existent instance (for example, if you mis-identify the instance with a non-existent instance name—for example, `ohsz` instead of `ohs2`)— UNKNOWN will be returned.

4.3.6 Deleting an Oracle HTTP Server Instance

You can delete an Oracle HTTP Server instance in both a WebLogic Server Domain and a standalone domain.

4.3.6.1 Deleting an Oracle HTTP Server Instance in a WebLogic Server Domain

In a WebLogic Server Domain, you can use either the custom WLST command `deleteOHSInstance()` or from Fusion Middleware Control. These procedures are described in the following sections.

4.3.6.1.1 Deleting an Instance by Using WLST If you are in a WebLogic Server Domain, you can delete an Oracle HTTP Server instance by using the customer WLST command `deleteOHSInstance()`. When you use this command, the following happens:

- The selected instance information is removed form config.xml.
- All OHS configuration directories and their contents are deleted; for example, OHS/*instanceName* and OHS/instances/*instanceName*).
- All logfiles associated with the deleted instance are deleted.
- All state information for the deleted instance is removed.

Note: You cannot delete an instance by using deleteOHSInstance() if Node Manager is down.

To delete an instance by using WLST:

1. Connect to WLST, as described in Section 4.3.2.2, "Starting Oracle HTTP Server Instances by Using WLST".
2. At the command prompt, enter:

   ```
   deleteOHSInstance(instanceName='instanceName')
   ```

   For example, to delete an OHS instance named `ohs1` use the following command:

   ```
   deleteOHSInstance(instanceName='ohs1')
   ```

You cannot delete any OHS instance in either an UNKNOWN or a RUNNING state.

4.3.6.1.2 Deleting an Instance by Using Fusion Middleware Control To delete an Oracle HTTP Server instance by using Fusion Middleware Control:

Note: You cannot delete a running Oracle HTTP Server instance. If the instance is running, stop it, as described in Section 4.3.3, "Stopping Oracle HTTP Server Instances" and then proceed with the following steps.

1. Log in to Fusion Middleware Control and navigate to the system component instance home page for the WebLogic Server Domain within which you want to delete the Oracle HTTP Server instance.
2. Open the WebLogic Server Domain menu and select **Administration** then **Create/Delete OHS**.

   The OHS Instances page appears.
3. Select the instance you want to delete and click **Delete**.

   A confirmation window appears.
4. Click **Yes** to complete the deletion.

   The OHS Instances page appears, with an information method indicating that the selected Oracle HTTP Server instance was deleted.

4.3.6.2 Deleting an Oracle HTTP Server Instance from a Standalone Domain

You can delete an Oracle HTTP Server instance in a standalone domain by using the Configuration Wizard so long as it is not the only instance in the domain. The Configuration Wizard always requires at least one Oracle HTTP Server instance in a standalone domain so you will not be able to delete one if it's the only instance in the domain. If you want to delete the only instance in a standalone domain, you should instead completely remove the entire domain directory.

Deleting Oracle HTTP Server instances by using the Configuration Wizard is actually only a partial deletion (and is inconsistent with the way deletion is done on the WebLogic Server domain side by using `deleteOHSInstance()`; see Section 4.3.6.1.1, "Deleting an Instance by Using WLST"). When you delete a standalone instance by using the Configuration Wizard, the following occurs:

- Information on the specific instance is removed from config.xml, so this instance is no longer recognized as valid. When you launch the Configuration Wizard again for another update, the deleted instance will not appear.
- The logs compiled for the deleted instance are left intact at: *DOMAIN_HOME*/servers/ohs1... If a new instance with the same name is subsequently created, it will inherit and continue logging to these files.
- The deleted instance's configuration directories and their contents are *not* deleted; they remain intact at: *DOMAIN_HOME*/config/fmwconfig/components/OHS/*instanceName* and *DOMAIN_HOME*/config/fmwconfig/components/OHS/instances/*instanceName*. The only change in both directories is that the following files are renamed: httpd.conf becomes httpd.conf.bak; ssl.conf becomes ssl.conf.bak; and admin.conf becomes admin.conf.bak. This prevents the instance from being started. (If you create a new instance with the same name as the instance you deleted, this information will be overwritten, but the *.bak files will remain).
- The deleted instance's state information is left intact at *DOMAIN_HOME*/system_components/... If a new instance of the same name is subsequently created, it will inherit the state of the old instance. Instead of starting in UNKNOWN, it could be SHUTDOWN or even FAILED_NOT_RESTARTABLE out of the gate.

To delete an Oracle HTTP Server instance in a standalone domain, do the following:

1. Shutdown all running instances (see Section 4.3.3, "Stopping Oracle HTTP Server Instances"). Be aware the Configuration Wizard will not check the state of the Oracle HTTP Server instance so you will need to verify that all instances are indeed stopped.

2. If it is running, shut down Node Manager.
3. Launch the Configuration Wizard (see *Installing and Configuring Oracle HTTP Server*) and do the following:
   a. Select **Update an existing domain** and select the path to the domain.
   b. Skip both the Templates screen and the JDK Selection screen by clicking **Next** on each.
   c. On the System Components screen, select the instance you want to delete and click **Delete**.

      The selected instance is deleted.
   d. Click **Next** and, on the OHS Server screen, click **Next** again.
   e. On the Configuration Summary screen, verify that the selected instance has been deleted and click **Update**.
   f. On the Success screen, click **Finish**.

4.4 Remotely Administering Oracle HTTP Server

You can remotely manage an Oracle HTTP Server running in a standalone environment from a collocated Oracle HTTP Server implementation running on a separate machine. This feature enables you to use the WebLogic Scripting Tool (WLST) or Fusion Middleware Control from the remote machine to start, restart, stop, and configure the component. This chapter describes how to set up the environments to

- Section 4.4.1, "Setting Up a Remote Environment"
- Section 4.4.2, "Running Oracle HTTP Server Remotely"

4.4.1 Setting Up a Remote Environment

The following instructions describe how to set up a remote environment, which will enable you to run Oracle HTTP Server installed on one machine from an installation on another. This section contains the following information:

- Section 4.4.1.1, "Host Requirements."
- Section 4.4.1.2, "Task 1: Set Up an Expanded Domain on host1."
- Section 4.4.1.3, "Task 2: Pack the Domain on host1."
- Section 4.4.1.4, "Task 3: Unpack the Domain on host2."

4.4.1.1 Host Requirements

To remotely manage Oracle HTTP Server, you need to have separate hosts installed on separate machines:

- A collocated installation (for this document, this installation will be called *host1*).
- A standalone installation (*host2*). The path to standalone *MW_HOME* on *host2* must be the same as the path to collocated *MW_HOME* on *host1*; for example:

  ```
  /scratch/user/work
  ```

4.4.1.2 Task 1: Set Up an Expanded Domain on host1

The following steps describe how to set up an expanded domain and link it to a database on the collocated version of Oracle HTTP Server (host1).

1. Using the Repository Configuration Utility (RCU), set up and install a database for the expanded domain. For more information, see *Oracle Fusion Middleware Creating Schemas with the Repository Creation Utility*.
2. Launch the Configuration Wizard and create an expanded domain. Use the values specified in Table 4–1.

Table 4–1 Setting Up an Expanded Domain

For...	Select or Enter...
Create Domain	**Create a new domain** and specify its path (for example, *MW_HOME*/user_projects/domains/ohs1_domain)
Templates	**Oracle HTTP Server (Collocated)**
Application Locations	The default
Administrator Account	A username and password (for example, *weblogic* and *welcome1*)
Database Configuration Type	The RCU data. Then, click **Get RCU Configuration** and then **Next**.
Optional Configuration	The following items: ■ Administration Server ■ Node Manager ■ System Components ■ Deployment and Services
Administration Server	The listen address (All Local Addresses or the valid name or address for host1) and port
Node Manager	Per Domain and specify the NodeManager credentials (for example, `weblogic` and `welcome1`).
System Components	**Add** and set the fields, using OHS as the Component Type (for example, use a System Component value of ohs1).
OHS Server	The listen addresses and ports or use the defaults.
Machines	**Add**. This will add a machine to the domain (for example, ohs1_Machine) and the Node Manager listen and port values. You must specify a listen address for host2 that is accessible from host1, such the valid name or address for host2 (do not use localhost or All Local Addresses).
Assign System Components	The OHS component (for example, `ohs1`) then use the right arrow to assign the component to the machine (`ohs1_machine`, for example).
Configuration Summary	**Create** (note that the OPSS steps may take some minutes).

4.4.1.3 Task 2: Pack the Domain on host1

On host1, use the following command to pack the domain:

```
<MW_HOME>/ohs/common/bin/pack.sh -domain=path to domain -template=path to template
-template_name=name -managed=true
```

For example:

```
<MW_HOME>/ohs/common/bin/pack.sh -domain=<MW_HOME>/user_projects/domains/ohs1_
domain -template=/tmp/ohs1_tmplt.jar -template_name=ohs1 -managed=true
```

4.4.1.4 Task 3: Unpack the Domain on host2

Use the following steps to unpack the domain you packed on host1, above, on host2:

1. Copy the template file created in "Task 2: Pack the Domain on host1" from host1 to host2.
2. Use the following command to unpack the domain:

   ```
   <MW_HOME>/ohs/common/bin/unpack.sh -domain=path to domain -template=path to
   template
   ```

   For example:

   ```
   <MW_HOME>/ohs/common/bin/unpack.sh -domain=<MW_HOME>/user_
   projects/domains/ohs1_domain -template=/tmp/ohs1_tmplt.jar
   ```

4.4.2 Running Oracle HTTP Server Remotely

Once you have unpacked the domain created on host1 onto host2, you can use the same set of WLST commands and Fusion Middleware Control tools you would in a collocated environment to start, stop, restart, and configure the component.

To run an Oracle HTTP Server remotely, do the following:

1. Start the WebLogic Administration Server on host1:

   ```
   <MW_HOME>/user_projects/domains/ohs1_domain/bin/startWebLogic.sh &
   ```

2. Start Node Manager on host2:

   ```
   <MW_HOME>/user_projects/domains/ohs1_domain/bin/startNodeManager.sh &
   ```

You can now run the Oracle HTTP Server instance on host2 from the collocated implementation on host1. You can use any of the WLST commands or any of the Fusion Middleware Control tools. For example, to connect host2 to Node Manager and start the server ohs1, from host1 enter:

```
<MW_HOME>/ohs/common/bin/wlst.sh
nmConnect('weblogic', 'welcome1', '<nm-host>', '<nm-port>', 'ohs1_domain')
nmStart(serverName='ohs1', serverType='OHS')
```

See Section 4.3, "Performing Basic OHS Tasks" for information on starting, stopping, restarting, and configuring Oracle HTTP Server components.

4.5 Specifying Server Properties

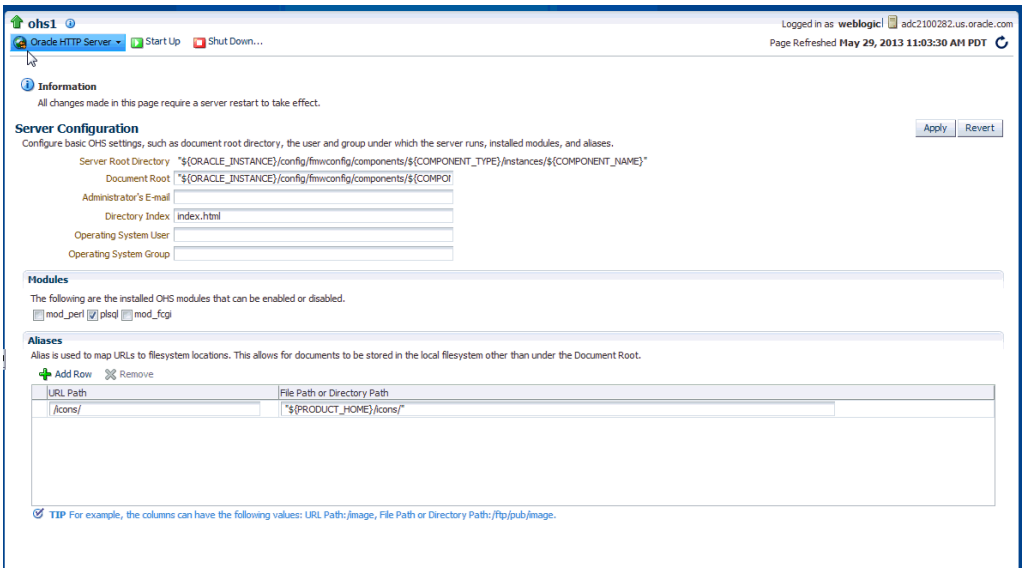
Server properties for Oracle HTTP Server can be set using Fusion Middleware Control or direct editing of the Oracle HTTP Server configuration files. You cannot use WLST commands to specify the server properties.

- Specifying Server Properties by Using Fusion Middleware Control
- Editing the httpd.conf File to Specify Server Properties

4.5.1 Specifying Server Properties by Using Fusion Middleware Control

To specify the server properties using the Fusion Middleware Control:

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **Server Configuration** from the Administration menu. The Server Configuration page appears.



3. Enter the documentation root directory in the **Document Root** field that forms the main document tree visible from the website.
4. Enter the e-mail address in the **Administrator's E-mail** field that the server will includes in error messages sent to the client.
5. Enter the directory index in the **Directory Index** field. The is the main (index) page that will be displayed when a client first accesses the website.
6. Optional: Enter the user name in the **Operating System User** field.

   This field is normally blank. It may be set to the user that installed Oracle HTTP Server and starts Node Manager.
7. Optional: Enter the group name in the **Operating System Group** field.

   This field is normally blank. It may be set to the group of the user that installed Oracle HTTP Server and starts Node Manager.
8. The Modules region is used to enable or disable modules. There are three modules that you can enable or disable: mod_perl, mod_fcgi, and mod_plsql.

   For instructions on configuring the mod_perl module, see "Configuring mod_perl" on page 4-25.
9. Create an alias, if necessary in the Aliases table. An alias maps to a specified directory. For example, to use a specific set of content pages for a group you can create an alias to the directory that has the content pages.
10. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
11. Restart Oracle HTTP Server as described in Section 4.3.4.

The server properties are saved, and shown on the Server Configuration page.

4.5.2 Editing the httpd.conf File to Specify Server Properties

To specify the server properties using the httpd.conf file:

> **Note:** Before attempting to edit any .conf file, you should familiarize yourself with the layout of the configuration file directories, mechanisms for editing the files, and learn more about the files themselves. For this information, see Section 1.6, "Understanding Configuration Files".

1. Open the httpd.conf file using either a text editor or the Advanced Server Configuration page in Fusion Middleware Control. (See Section 4.6.6, "Modifying an Oracle HTTP Server Configuration File.")
2. In the DocumentRoot section of the file, enter the directory that stores the main content for the website. The following is an example of the syntax:

   ```
   DocumentRoot "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/htdocs"
   ```

3. In the ServerAdmin section of the file, enter the administrator's email address. This is the e-mail address that will appear on client pages. The following is an example of the syntax:

   ```
   ServerAdmin WebMaster@example.com
   ```

4. In the DirectoryIndex section of the file, enter the directory index. This is the main (index) page that will be displayed when a client first accesses the website. The following is an example of the syntax:

   ```
   DirectoryIndex index.html index.html.var
   ```

5. Create aliases, if needed. An alias maps to a specified directory. For example, to use a specific set of icons, you can create an alias to the directory that has the icons for the Web pages. The following is an example of the syntax:

   ```
   Alias /icons/ "${PRODUCT_HOME}/icons/"

   <Directory "${PRODUCT_HOME}/icons">
       AllowOverride None
       Order allow,deny
       Allow from all
   </Directory>
   ```

6. Save the file.
7. Restart Oracle HTTP Server as described in Section 4.3.4.

4.6 Configuring Oracle HTTP Server

This section includes the following sections:

- Section 4.6.1, "Configuring Secure Sockets Layer"
- Section 4.6.3, "Configuring MIME Settings"
- Section 4.6.4, "Configuring mod_perl"
- Section 4.6.5, "Configuring the Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)"
- Section 4.6.6, "Modifying an Oracle HTTP Server Configuration File"
- Section 4.6.7, "Removing Access to Unneeded Content"

- Section 4.6.8, "Using the apxs Command to Install Extension Modules"
- Section 4.6.9, "Disabling the Options Method"

Note: Fusion Middleware Control and other Oracle software which manage the Oracle HTTP Server configuration might save configuration files in a different, equivalent format. After using the software to make a configuration change, multiple configuration files might be rewritten.

4.6.1 Configuring Secure Sockets Layer

Secure Sockets Layer (SSL) is an encrypted communication protocol that is designed to securely send messages across the Internet. It resides between Oracle HTTP Server on the application layer and the TCP/IP layer, transparently handling encryption and decryption when a secure connection is made by a client.

One common use of SSL is to secure Web HTTP communication between a browser and a Web server. This case does not preclude the use of non-secured HTTP. The secure version is simply HTTP over SSL (HTTPS). The differences are that HTTPS uses the URL scheme `https://` rather than `http://`. The default communication port is 4443 in Oracle HTTP Server. Oracle HTTP Server does not use the 443 standard `https://` privileged port because of security implications. For information about running Oracle HTTP Server on privileged ports see Section 4.3.2.4, "Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only)."

By default, an SSL listen port is configured and enabled using a default **wallet** during installation. Wallets store your credentials, such as certificate requests, certificates, and private keys.

The default wallet that is automatically installed with Oracle HTTP Server is for testing purposes only. A real wallet must be created for your production server. The default wallet is located in the *DOMAIN_HOME*/config/fmwconfig/components/OHS/instances/*componentName*/keystores/ default directory. You can either place the new wallet in this location, or change the `SSLWallet` directive in *DOMAIN_HOME*/config/fmwconfig/components/OHS/instances/*componentName*/ssl.conf to point to the location of your real wallet.

For the changes to take effect, you should restart the Oracle HTTP Server components as described in Section 4.3.4.

For information about configuring wallets and SSL using Fusion Middleware Control, see "Enabling SSL for Oracle HTTP Server Virtual Hosts" in the *Administering Oracle Fusion Middleware*.

4.6.2 Configuring Secure Sockets Layer in Standalone Mode

The following sections contain information about how to enable and configure SSL for Oracle HTTP Server in standalone mode. These instructions make use of the mod_ossl plug-in to Oracle HTTP Server which enables the server to use SSL.

- Section 4.6.2.1, "Configure SSL"
- Section 4.6.2.2, "Specify SSLVerifyClient on the Server Side"
- Section 4.6.2.3, "Enable SSL Between Oracle HTTP Server and Oracle WebLogic Server"

4.6.2.1 Configure SSL

By default, SSL is enabled when you install Oracle HTTP Server. Perform these tasks to modify and configure SSL:

- Section 4.6.2.1.1, "Create a Real Wallet"
- Section 4.6.2.1.2, "(Optional) Customize Your Configuration"
- Section 4.6.2.1.3, "Basic Configuration Example"

4.6.2.1.1 Create a Real Wallet To configure Oracle HTTP Server for SSL, you need a wallet that contains the certificate for the server. Wallets store your credentials, such as certificate requests, certificates, and private keys.

The default wallet that is automatically installed with Oracle HTTP Server is for testing purposes only. A real wallet must be created for your production server. The default wallet is located in *$ORACLE_INSTANCE*/config/fmwconfig/components/*$COMPONENT_TYPE*/instances/*$COMPONENT_NAME*/keystores/default. You can either place the new wallet in that location, or change the SSLWallet directive in *$ORACLE_INSTANCE*/config/fmwconfig/components/*$COMPONENT_TYPE*/instances/*$COMPONENT_NAME*/ssl.conf (the pre-install location) to point to the location of your real wallet.

> **See Also:** "orapki" in *Administering Oracle Fusion Middleware* for instructions on creating a wallet. It is important that you do the following:
>
> Generate a certificate request: For the Common Name, specify the name or alias of the site you are configuring. Make sure that you enable this auto_login_only feature.

4.6.2.1.2 (Optional) Customize Your Configuration Optionally, you can further customize your configuration using mod_ossl directives.

> **See Also:** Section G.2, "mod_ossl" for a list and descriptions of directives accepted by mod_ossl.

> **Note:** The files installed during configuration contain all of the necessary SSL configuration directives and a default setup for SSL.

4.6.2.1.3 Basic Configuration Example Your SSL configuration must contain, at minimum, the following directives:

```
LoadModule ossl_module        "${PRODUCT_HOME}/modules/mod_ossl.so"
Listen 4443
ServerName www.testohs.com
SSLEngine on
#  SSL Protocol Support:
#  List the supported protocols.
SSLProtocol nzos_Version_1_2 nzos_Version_1_1 nzos_Version_1_0
#  SSL Cipher Suite:
#  List the ciphers that the client is permitted to negotiate.
SSLCipherSuite  SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,SSL_RSA_WITH_
3DES_EDE_CBC_SHA,SSL_RSA_WITH_DES_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_
WITH_AES_256_CBC_SHA
SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_
TYPE}/instances/${COMPONENT_NAME}/keystores/default"
</VirtualHost>To enable client authentication, do the following:
```

4.6.2.2 Specify SSLVerifyClient on the Server Side

Use the appropriate client certificate on your client side for the HTTPS connection. See your client documentation for information on getting and using a client certificate. Be sure that your client certificate is trusted by the server wallet.

- Section 4.6.2.2.1, "Forcing Clients to Authenticate Using Certificates"
- Section 4.6.2.2.2, "Forcing a Client to Authenticate for a Particular URL"
- Section 4.6.2.2.3, "Authorizing a Client for a Particular URL"
- Section 4.6.2.2.4, "Allowing Clients with Strong Ciphers and CA Client Certificate or Basic Authentication"

> **See Also:** "Importing a Certificate or a Trusted Certificate Using WLST" in *Administering Oracle Fusion Middleware Guide* for instructions on how to import a trusted certificate into your wallet.

4.6.2.2.1 Forcing Clients to Authenticate Using Certificates You can force the client to validate its client certificate and allow access to the server using the following method. This scenario is valid for all clients having a client certificate supplied by the server Certificate Authority (CA). The server can validate client's supplied certificates against its CA for additional permission.

```
# require a client certificate which has to be directly
# signed by our CA certificate
SSLVerifyClient require
SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_
TYPE}/instances/${COMPONENT_NAME}/keystores/default"
```

4.6.2.2.2 Forcing a Client to Authenticate for a Particular URL To force a client to authenticate using certificates for a particular URL, you can use the per-directory reconfiguration features of mod_ossl:

```
SSLVerifyClient none
SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_
TYPE}/instances/${COMPONENT_NAME}/keystores/default"
<Location /secure/area>
   SSLVerifyClient require
</Location>
```

4.6.2.2.3 Authorizing a Client for a Particular URL To do this, check that part of the client certificate matches what you expect. Usually, this means checking all or part of the Distinguished Name (DN), to see if it contains some known string. There are two ways to do this, using either mod_auth_basic or SSLRequire.

The mod_auth_basic method is generally required when the certificates are completely arbitrary, or when their DNs have no common fields (usually the organization, and so on). In this case, you should establish a password database containing all of the clients allowed, for example:

```
SSLVerifyClient      none
<Directory /access/required>
    SSLVerifyClient      require
    SSLOptions           +FakeBasicAuth
    SSLRequireSSL
    AuthName             "Oracle Auth"
    AuthType             Basic
    AuthBasicProvider    file
    AuthUserFile         httpd.passwd
    Require              valid-user
```

```
</Directory>
```

The password used in this example is the DES encrypted string `password`. For more information on the directive, see Section G.2.10, "SSLOptions" which describes the SSLOptions directive of the mod_ossl module.

```
httpd.passwd

Subject:        OU=Class 3 Public Primary Certification Authority,O=VeriSign\,
Inc.,C=US
Subject:        CN=GTE CyberTrust Global Root,OU=GTE CyberTrust Solutions\,
Inc.,O=GTE Corporation,C=US
Subject:        CN=localhost,OU=FOR TESTING ONLY,O=FOR TESTING ONLY
Subject:        OU=Class 2 Public Primary Certification Authority,O=VeriSign\,
Inc.,C=US
Subject:        OU=Class 1 Public Primary Certification Authority,O=VeriSign\,
Inc.,C=US
```

When your clients are all part of a common hierarchy, which is encoded into the DN, you can match them more easily using SSLRequire, for example:

```
SSLVerifyClient      none
SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_
TYPE}/instances/${COMPONENT_NAME}/keystores/default"

<Directory /access/required>
  SSLVerifyClient      require
  SSLOptions           +FakeBasicAuth
  SSLRequireSSL
  SSLRequire       %{SSL_CLIENT_S_DN_O}  eq "VeriSign\, Inc." \
and %{SSL_CLIENT_S_DN_OU} in {"Class", "Public", "Primary"}
</Directory>
```

4.6.2.2.4 Allowing Clients with Strong Ciphers and CA Client Certificate or Basic Authentication

The following examples presume that clients on the Intranet have IPs in the range 192.168.1.0/24, and that the part of the Intranet website you want to allow Internet access to is /access/required. This configuration should remain outside of your HTTPS virtual host, so that it applies to both HTTPS and HTTP.

```
SSLWallet "$ORACLE_INSTANCE/config/fmwconfig/components/$COMPONENT_
TYPE/instances/$COMPONENT_NAME/keystores/default"
<Directory  /access/required>
    #   Outside the subarea only Intranet access is granted
    Require             ip 192.168.1.0/24
</Directory>

<Directory  /access/required>
    #   Inside the subarea any Intranet access is allowed
    #   but from the Internet only HTTPS + Strong-Cipher + Password
    #   or the alternative HTTPS + Strong-Cipher + Client-Certificate

    #   If HTTPS is used, make sure a strong cipher is used.
    #   Additionally allow client certs as alternative to basic auth.
    SSLVerifyClient      optional
    SSLOptions           +FakeBasicAuth +StrictRequire
    SSLRequire           %{SSL_CIPHER_USEKEYSIZE}>= 128
    #   Force clients from the Internet to use HTTPS
    RewriteEngine        on
    RewriteCond          %{REMOTE_ADDR} !^192\.168\.1\.[0-9]+$
    RewriteCond          %{HTTPS} !=on
    RewriteRule          . - [F]
```

```
    #   Allow Network Access and/or Basic Auth
    Satisfy             any

    #   Network Access Control
    Require             ip 192.168.1.0/24
    #   HTTP Basic Authentication
    AuthType            basic
    AuthName            "Protected Intranet Area"
    AuthBasicProvider   file
    AuthUserFile        htpasswd
    Require             valid-user
</Directory>
```

4.6.2.3 Enable SSL Between Oracle HTTP Server and Oracle WebLogic Server

Use the Oracle WebLogic Server Proxy Plug-In to enable SSL between Oracle HTTP Server and Oracle WebLogic Server. The plug-ins allow you to configure SSL libraries and configure one-way and two- way SSL communications. For more information, see "Use SSL with Plug-Ins" and "Parameters for Oracle WebLogic Server Proxy Plug-Ins" in *Using Oracle WebLogic Server Proxy Plug-Ins 12.1.2*.

4.6.3 Configuring MIME Settings

Multipurpose Internet Mail Extension (MIME) settings are used by Oracle HTTP Server to interpret file types, encodings, and languages. MIME settings for Oracle HTTP Server can only be set using Fusion Middleware Control. You cannot use WLST commands to specify the MIME settings.

The following tasks can be completed on the MIME Configuration page:

- Configuring MIME Types
- Configuring MIME Encoding
- Configuring MIME Languages

4.6.3.1 Configuring MIME Types

MIME type maps a given file extension to a specified content type. The MIME type is used for filenames containing an extension.

4.6.3.1.1 Using Fusion Middleware Control to Configure MIME Types

To configure a MIME type using Fusion Middleware Control, do the following:

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **MIME Configuration** from the Administration menu. The MIME configuration page appears.
3. Click **Add Row** in MIME Configuration region. A new, blank row is added to the list.
4. Enter the MIME type.
5. Enter the file extension.
6. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
7. Restart Oracle HTTP Server, as described in Section 4.3.4.

The MIME configuration is saved, and shown on the MIME Configuration page.

4.6.3.2 Configuring MIME Encoding

MIME encoding enables Oracle HTTP Server to determine the file type based on the file extension. You can add and remove MIME encodings. The encoding directive maps the file extension to a specified encoding type.

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **MIME Configuration** from the Administration menu. The MIME configuration page appears.
3. Expand the MIME Encoding region by clicking the plus sign (+) next to MIME Encoding.
4. Click **Add Row** in MIME Encoding region. A new, blank row is added to the list.
5. Enter the MIME encoding, such as `x-gzip`.
6. Enter the file extension, such as .gx.
7. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
8. Restart Oracle HTTP Server as described in Section 4.3.4.

4.6.3.3 Configuring MIME Languages

The MIME language setting maps file extensions to a particular language. This directive is commonly used for content negotiation, in which Oracle HTTP Server returns the document that most closely matched the preferences set by the client.

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **MIME Configuration** from the Administration menu. The MIME configuration page appears.
3. Expand the MIME Languages region by clicking the plus sign (+) next to MIME Languages.
4. Click **Add Row** in MIME Languages region. A new, blank row is added to the list.
5. Enter the MIME language, such as en-US.
6. Enter the file extension, such as en-us.
7. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
8. Restart Oracle HTTP Server as described in Section 4.3.4.

4.6.4 Configuring mod_perl

The mod_perl module embeds the Perl interpreter into Oracle HTTP Server. This eliminates start-up overhead and enables you to write modules in Perl. The module is disabled, by default.

To enable the mod_perl module using Fusion Middleware Control, do the following:

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **mod_perl Configuration** from the Administration menu. The mod_perl configuration page appears.

> **Note:** If mod_perl has not been enabled, then you will be redirected to the Server Configuration page. Select mod_perl and click **Apply** to enable mod_perl. After the confirmation page has been displayed, restart Oracle HTTP Server, and then return to the mod_perl Configuration page.

3. Enter the switch information in the Switches field.
4. Enter the environment variables to be passed to the scripts in the Environment field.
5. Enter the required script names in the Require field.
6. Click **Add Row** to create a new row.
7. Configure mod_perl directives for a Location in the Perl Locations table. The Location assigns many rules that the server should follow when the request's URI matches the Location.
   a. Enter the base URI for the Perl scripts in the Locations field. Just as it is the widely accepted convention to use /cgi-bin for your mod_cgi scripts, it is also conventional to use /perl as the base URI of the Perl scripts that run under mod_perl.
   b. Enter options in the Options field. The `PerlOptions` directive provides fine-grained configuration by providing control over which class of Perl interpreter pool to be used. Options are enabled by prepending them with a plus sign (+) and are disabled by prepending them with a minus sign (-).
   c. If you want to send headers, then click the **Send Header** check box. The `PerlSendHeader` directive is for mod_perl 1.0 backwards-compatibility. When enabled, the server sends an HTTP header to the browser on every script invocation. You should disable this option for NPH (non-parsed-headers) scripts.
   d. Enter the environment in the Environment field. The `PerlSetEnv` directive enables you to specify system environment variables and pass them into your mod_perl handlers.
   e. Enter the response handler in the Response Handler field. The `PerlResponseHandler` directive tells mod_perl which callback is going to do the job.
   f. Enter the authentication handler in the Authentication Handler field. The PerlAuthenHandler directive is used to set the handler to verify a user's identification credentials.
8. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
9. Restart Oracle HTTP Server as described in Section 4.3.4.

The mod_perl module configuration is saved and shown on the mod_perl Configuration page.

Note: If you are manually editing the mod_perl configuration instead of using Fusion Middleware Control, then all directives must be defined within the `<IfModule mod_perl.c>` block of the mod_perl.conf file. Any mod_perl related directives defined outside of this block might be ignored.

4.6.5 Configuring the Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)

You can configure the Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs) either by using Fusion Middleware Control or by editing the mod_wl_ohs.conf configuration file manually.

For information about the prerequisites and procedure for configuring the Oracle WebLogic Server Proxy Plug-In to proxy requests from Oracle HTTP Server to Oracle WebLogic Server, see "Configuring the Plug-In for Oracle HTTP Server" in *Using Oracle WebLogic Server Proxy Plug-Ins 12.1.2*.

4.6.6 Modifying an Oracle HTTP Server Configuration File

Note: Fusion Middleware Control and other Oracle software that manage the Oracle HTTP Server configuration might save these files in a different, equivalent format. After using the software to make a configuration change, multiple configuration files might be rewritten.

To modify an Oracle HTTP Server configuration file by using Fusion Middleware Control, do the following:

1. Select **Administration** from the HTTP Server menu.
2. Select **Advanced Configuration** from the Administration menu item. The Advanced Server Configuration page appears.
3. Select the configuration file from the list, such as the `httpd.conf` file.
4. Edit the file, as needed.
5. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
6. Restart Oracle HTTP Server as described in Section 4.3.4.

The file is saved and shown on the Advanced Server Configuration page.

See Also: Section 1.6, "Understanding Configuration Files"

4.6.7 Removing Access to Unneeded Content

By default, the httpd.conf file allows server access to extra content such as documentation and sample scripts. This access might present a low level security risk.

You might want to tailor this extra content in your own environment to suit your use cases. Follow the instructions in Section 4.6.6, "Modifying an Oracle HTTP Server Configuration File" to access the file.

- Section 4.6.7.1, "Edit the cgi-bin Section"
- Section 4.6.7.2, "Edit the Fancy Indexing Section"

- Section 4.6.7.3, "Edit the Product Documentation Section"

4.6.7.1 Edit the cgi-bin Section

Examine the contents of the `cgi-bin` directory. You can either remove the code from the httpd.conf file that you do not need, or change the following Directory directive to point to your own CGI script directory.

```
...
#
# "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_
TYPE}/instances/${COMPONENT_NAME}/cgi-bin" should be changed to whatever your
ScriptAliased
# CGI directory exists, if you have that configured.
#
<Directory "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_
TYPE}/instances/${COMPONENT_NAME}/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
...
```

4.6.7.2 Edit the Fancy Indexing Section

Edit the following sections pertaining to fancy indexing in the httpd.conf file for your use cases.

```
...
#
# IndexOptions: Controls the appearance of server-generated directory
# listings.
#
IndexOptions FancyIndexing HTMLTable VersionSort

# We include the /icons/ alias for FancyIndexed directory listings.  If
# you do not use FancyIndexing, you may comment this out.
#
Alias /icons/ "${PRODUCT_HOME}/icons/"

<Directory "${PRODUCT_HOME}/icons">
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

#
# AddIcon* directives tell the server which icon to show for different
# files or filename extensions.  These are only displayed for
# FancyIndexed directories.
#
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip

AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*

AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
```

```
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core


AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^

#
# DefaultIcon is which icon to show for files which do not have an icon
# explicitly set.
#
DefaultIcon /icons/unknown.gif

#
# AddDescription allows you to place a short description after a file in
# server-generated indexes.  These are only displayed for FancyIndexed
# directories.
# Format: AddDescription "description" filename
#
#AddDescription "GZIP compressed document" .gz
#AddDescription "tar archive" .tar
#AddDescription "GZIP compressed tar archive" .tgz
...
#
# ReadmeName is the name of the README file the server will look for by
# default, and append to directory listings.
#
# HeaderName is the name of a file which should be prepended to
# directory indexes.
ReadmeName README.html
HeaderName HEADER.html

#
# IndexIgnore is a set of filenames which directory indexing should ignore
# and not include in the listing.  Shell-style wildcarding is permitted.
#
IndexIgnore .??* *~ *# HEADER* README* RCS CVS *,v *,t
...
```

4.6.7.3 Edit the Product Documentation Section

You can remove the following documentation configuration sections from the httpd.conf file if they are not needed.

```
...
#
# This should be changed to the ServerRoot/manual/.  The alias provides
```

```
# the manual, even if you choose to move your DocumentRoot.  You may comment
# this out if you do not care for the documentation.
#
AliasMatch ^/manual(?:/(?:de|en|es|fr|ja|ko|pt-br|ru|tr))?(/.*)?$ "${PRODUCT_
HOME}/manual$1"

<Directory "${PRODUCT_HOME}/manual">
    AllowOverride None
    Order allow,deny
    Allow from all

    <Files *.html>
        SetHandler type-map
    </Files>
    # .tr is text/troff in mime.types!
    <Files *.html.tr.utf8>
        ForceType text/html
    </Files>

    SetEnvIf Request_URI ^/manual/(de|en|es|fr|ja|ko|pt-br|ru|tr)/
prefer-language=$1
    RedirectMatch 301 ^/manual(?:/(de|en|es|fr|ja|ko|pt-br|ru|tr)){2,}(/.*)?$
/manual/$1$2

    LanguagePriority en de es fr ja ko pt-br ru tr
    ForceLanguagePriority Prefer Fallback
</Directory>

...
```

4.6.8 Using the apxs Command to Install Extension Modules

> **Note:** This command is only for UNIX and Linux and should be used only for modules which are supplied in source code form. Follow the installation instructions for modules supplied in binary form.
>
> For more information about the apxs command, see the Apache HTTP Server documentation at:
>
> http://httpd.apache.org/docs/2.2/programs/apxs.html

The Apache Extension Tool (apxs) can be used to build and install Apache HTTP Server extension modules for Oracle HTTP Server. apxs installs modules in the *ORACLE_HOME*/ohs/modules directory for access by any Oracle HTTP Server instances which run from this installation.

Recommended apxs options for use with Oracle HTTP Server are:

Option	Purpose	Example Command
-c	Compile module source	`$ORACLE_HOME/ohs/bin/apxs -c mod_example.c`
-i	Install module binary into *ORACLE_HOME*	`$ORACLE_HOME/ohs/bin/apxs -ci mod_example.c`

When the module binary has been installed into O*RACLE_HOME*, a `LoadModule` directive in httpd.conf or other configuration file is used to load the module into the server processes; for example:

```
LoadModule example_module "${ORACLE_HOME}/ohs/modules/mod_example.so"
```

The directive is required in the configurations for all instances which must load the module.

When the -a or -A option is specified, apxs will edit httpd.conf to add a LoadModule directive for the module. Do not use the `-a` and `-A` options with Oracle HTTP Server instances that are part of a WebLogic Server Domain. Instead, use Fusion Middleware Control to update the configuration, as described in Section 1.6.2, "Editing the Configuration".

You can use the -a or -A option with Oracle HTTP Server instances that are part of a standalone domain if the CONFIG_FILE_PATH environment variable is set to the staging directory for the instance before invoking apxs; for example:

```
CONFIG_FILE_PATH=$ORACLE_HOME/user_projects/domains/base_
domain/config/fmwconfig/components/OHS/ohs1
export CONFIG_FILE_PATH
$ORACLE_HOME/ohs/bin/apxs -cia mod_example.c
```

By default, apxs uses the Perl interpreter in /usr/bin. If apxs cannot locate the product install or encounters other operational errors when using /usr/bin/perl, use the Perl interpreter within the Middleware home by invoking apxs as follows:

```
$ORACLE_HOME/perl/bin/perl $ORACLE_HOME/ohs/bin/apxs -c mod_example.c
```

Modules often require directives besides `LoadModule` to properly function. After the module has been installed and loaded using the LoadModule directive, refer to the documentation for the module for any additional configuration requirements.

4.6.9 Disabling the Options Method

The Options method enables clients to determine which methods are supported by a web server. If enabled, it appears in the `Allow` line of HTTP response headers.

For example, if you send a request such as:

```
---- Request -------
OPTIONS / HTTP/1.0
Content-Length: 0
Accept: */*
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Win32)
Host: host123:80
```

you might get the following response from the web server:

```
---- Response --------
HTTP/1.1 200 OK
Date: Wed, 23 Apr 2008 20:20:49 GMT
Server: Oracle-Application-Server-11g/11.1.1.0.0 Oracle-HTTP-Server
Allow: GET,HEAD,POST,OPTIONS
Content-Length: 0
Connection: close
Content-Type: text/html
```

Some sources consider exposing the Options method a low security risk because malicious clients could use it to determine the methods supported by a web server. However, because web servers support only a limited number of methods, disabling this method will just slow down malicious clients, not stop them. In addition, the Options method may be used by legitimate clients.

If your Oracle Fusion Middleware environment does not have clients that require the Options method, you can disable it by including the following lines in the httpd.conf file:

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^OPTIONS
RewriteRule .* - [F]
</IfModule>
```

4.6.10 Updating Oracle HTTP Server Component Configurations on a Shared Filesystem

Functional or performance issues may be encountered when an Oracle HTTP Server component is created on a shared filesystem, including NFS (Network File System). In particular, lock files or UNIX sockets used by Oracle HTTP Server may not work or may have severe performance degradation; WLS requests routed by mod_wl_ohs may have severe performance degradation due to filesystem accesses in the default configuration.

Table 4-1 provides information about the Lock file issues and the suggested changes in the httpd.conf file specific to the operating systems.

Table 4–2 Lock File issues

Operating System	Description	httpd.conf changes
Linux	Lock files are not required. The Sys V semaphore is the preferred cross-process mutex implementation.	Change `AcceptMutex fcntl` to `AcceptMutex sysvsem` (two places). Comment out the `LockFile` directive (three places).
Solaris	Lock files are not required. The cross-process pthread mutex is the preferred cross-process mutex implementation.	Change `AcceptMutex fcntl` to `AcceptMutex pthread` (two places). Comment out the `LockFile` directive (three places).
Other UNIX platforms		Change the `LockFile` directive to point to a local filesystem (three places).
UNIX socket issues	mod_cgid is not enabled by default. If enabled, use the `ScriptSock` directive to place mod_cgid's UNIX socket on a local filesystem. mod_fastcgi is not enabled by default. If enabled, use the `FastCgiIpcDir` directive to place mod_fastcgi's UNIX sockets on a local filesystem.	

5

Managing and Monitoring Server Processes

This chapter describes how to manage and monitor Oracle HTTP Server. It discusses the procedures and tools to manage OHS in your environment.

This chapter includes the following sections:

5.1 Oracle HTTP Server Processing Model

The following sections explain the processing model for Oracle HTTP Server.

5.1.1 Request Process Model

After Oracle HTTP Server is started, it is ready to listen for and respond to HTTP(S) requests. The request processing model on Microsoft Windows systems differs from that on UNIX systems.

- On Microsoft Windows, there is a single parent process and a single child process. The child process creates threads that are responsible for handling client requests. The number of created threads is static and can be configured for performance.
- On UNIX, there is a single parent process that manages multiple child processes. The child processes are responsible for handling requests. The parent process brings up additional child processes as necessary, based on configuration. Although the server has the ability to dynamically bring up additional child processes, it is best to configure the server to start enough child processes initially so that requests can be handled without having to spawn more child processes.

5.1.2 Single Unit Process Model

Oracle HTTP Server provides functionality that allows it to terminate as a single unit if the parent process fails. The parent process is responsible for starting and stopping all the child processes for an Oracle HTTP Server instance. The failure of the parent process without first shutting down the child processes leaves Oracle HTTP Server in an inconsistent state that can only be fixed by manually shutting down all the orphaned child processes. Until all the child processes are closed, a new Oracle HTTP Server instance cannot be started because the orphaned child processes still occupy the ports the new Oracle HTTP Server instance needs to access.

To prevent this from occurring, the DMS instrumentation layer in child processes on UNIX and monitor functionality within WinNT MPM on Windows monitor the parent process. If they detect that the parent process has failed, then all of the remaining child processes are shut down.

5.2 Monitoring Oracle HTTP Server Performance

Oracle Fusion Middleware automatically and continuously measures run-time performance for Oracle HTTP Server. The performance metrics are automatically enabled; you do not need to set options or perform any extra configuration to collect them. If you encounter a problem, such as an application that is running slowly or is hanging, you can view particular metrics to find out more information about the problem.

Note that Fusion Middleware Control provides real-time data. If you are interested in viewing historical data, consider using Grid Control.

5.2.1 Viewing Oracle HTTP Server Performance Metrics

You can view metrics from the Oracle HTTP Server home menu of Fusion Middleware Control:

1. Select the Oracle HTTP Server that you want to monitor.

   The Oracle HTTP Server home page is displayed.

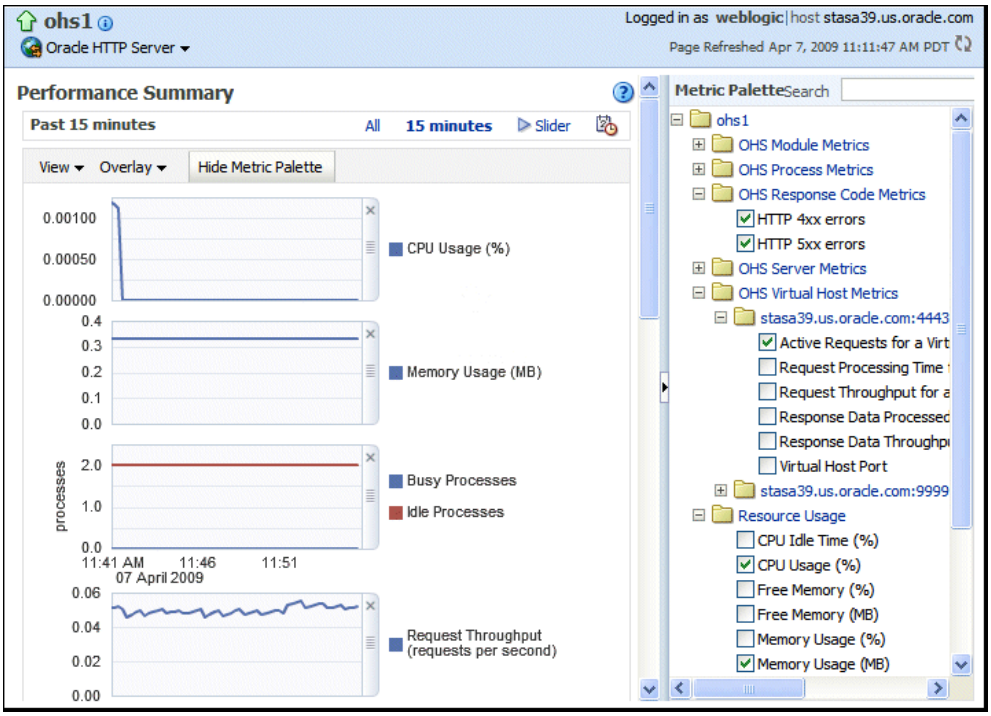
2. From the Oracle HTTP Server menu, choose **Monitoring**, and then select **Performance Summary**.

   The Performance Summary page is displayed. It shows performance metrics, and information about response time and request processing time for the Oracle HTTP Server instance.

3. To see additional metrics, click **Show Metric Palette** and expand the metric categories.

   Tip: Oracle HTTP Server port usage information is also available from the Oracle HTTP Server home menu.

The following figure shows the Oracle HTTP Server Performance Summary page with the Metric Palette displayed:



4. Select additional metrics to add them to the Performance Summary.

5.2.2 Understanding Oracle HTTP Server Performance Metrics

This section lists some most commonly-used metrics that can help you analyze Oracle HTTP Server performance.

OHS Server Metrics

The OHS Server Metrics folder contains performance metric options for Oracle HTTP Server. The following table describes the metrics in the OHS Server Metrics folder:

Element	Description
CPU Usage	CPU usage and idle times
Memory Usage	Memory usage and free memory, in MB
Processes	Busy and idle process metrics
Request Throughput	Request throughput, as measured by requests per second
Request Processing Time	Request processing time, in seconds
Response Data Throughput	Response data throughput, in KB per second
Response Data Processed	Response data processed, in KB per response
Active HTTP Connections	Number of active HTTP connections
Connection Duration	Length of time for connections
HTTP Errors	Number of HTTP 4xx and 5xx errors

OHS Virtual Host Metrics

The OHS Virtual Host Metrics folder contains performance metric options for virtual hosts, also known as access points. The following table describes the metrics in the OHS Virtual Host Metrics folder:

Element	Description
Request Throughput for a Virtual Host	Number of requests per second for each virtual host
Request Processing Time for a Virtual Host	Time to process each request for each virtual host
Response Data Throughput for a Virtual Host	Amount of data being sent for each virtual host
Response Data Processed for a Virtual Host	Amount of data being processed for each virtual host

OHS Module Metrics

The OHS Module Metrics folder contains performance metric option for modules. The following table describes the metrics in the OHS Module Metrics folder.

Element	Description
Request Handling Throughput	Request handling throughput for a module, in requests per second
Request Handling Time	Request handling time for a module, in seconds
Module Metrics	Modules including active requests, throughput, and time for each module

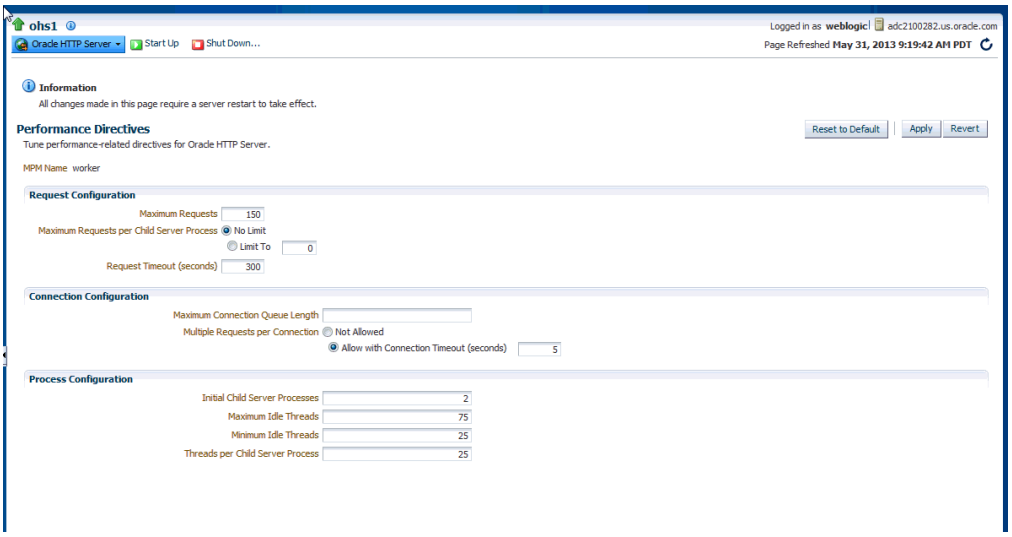
5.3 Configuring Oracle HTTP Server Performance Directives

Oracle HTTP Server uses directives in httpd.conf. This configuration file specifies the maximum number of HTTP requests that can be processed simultaneously, logging details, and certain limits and timeouts. Oracle HTTP Server supports and ships with the following three Multi-Processing Modules (MPMs) which are responsible for binding to network ports on the machine, accepting requests, and dispatching children to handle the requests:

- Worker: This is the default MPM for Oracle HTTP Server on UNIX/Linux platforms. This MPM implements a hybrid multi-process multi-threaded server. By using threads to serve requests, it is able to serve a large number of requests with fewer system resources than a process-based server. However, it retains much of the stability of a process-based server by keeping multiple processes available, each with many threads.
- WinNT: This is the default MPM for Oracle HTTP Server on Windows platforms. It uses a single control process which launches a single child process which in turn creates threads to handle requests.
- Prefork: This MPM implements a non-threaded, pre-forking server that handles requests in a manner similar to Apache 1.3. It is appropriate for sites that need to avoid threading for compatibility with non-thread-safe libraries. It is also the best MPM for isolating each request, so that a problem with a single request will not affect any other.

The discussion and recommendations in this section are based on the use of Worker or WinNT MPM, which uses threads. The thread-related directives listed below are not applicable if you are using the Prefork MPM.

The Performance Directives page enables you to tune performance-related directives for Oracle HTTP Server, as illustrated in the following figure:



Performance directives management consists of three areas: request configuration, connection configuration, and process configuration. You can set these configurations using the Performance Directive page of Fusion Middleware Control and by following the instructions in the following sections:

- Using Fusion Middleware Control to Set the Request Configuration
- Using Fusion Middleware Control to Set the Connection Configuration
- Using Fusion Middleware Control to Set the Process Configuration

5.3.1 Using Fusion Middleware Control to Set the Request Configuration

To specify the Oracle HTTP Server request configuration using Fusion Middleware Control, do the following:

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **Performance Directives** from the Administration menu. The Performance Directives page appears.
3. Enter the maximum number of requests in the **Maximum Requests** field (`MaxClients` directive). This setting limits the number of requests that can be dealt with at one time. The default and recommended value is 150. This is applicable for all Linux/UNIX platforms.
4. Set the maximum requests per child process in the Maximum Request per Child Process field (`MaxRequestPerChild` directive). You can choose to have no limit, or a maximum number. If you choose to have a limit, enter the maximum number in the field.
5. Enter the request timeout value in the Request Timeout (seconds) field (`Timeout` directive). This value sets the maximum time, in seconds, Oracle HTTP Server waits to receive a GET request, the amount of time between receipt of TCP packets

on a POST or PUT request, and the amount of time between ACKs on transmissions of TCP packets in responses.

6. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
7. Restart Oracle HTTP Server. See Section 4.3.4.

The request configuration settings are saved, and shown on the Performance Directives page.

5.3.2 Using Fusion Middleware Control to Set the Connection Configuration

To specify the connection configuration using Fusion Middleware Control, do the following:

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **Performance Directives** from the Administration menu. The Performance Directives page appears.
3. Enter the maximum connection queue length in the Maximum Connection Queue Length field (`ListenBacklog` directive). This is the queue for pending connections. This is useful if the server is experiencing a TCP SYN overload, which causes numerous new connections to open up, but without completing the pending task.
4. Set the Multiple Requests per Connection field (`KeepAlive` directive) to indicate whether to allow multiple connections. If you choose to allow multiple connections, enter the number of seconds for timeout in the Allow With Connection Timeout field.

   The Allow With Connection Timeout value sets the number of seconds the server waits for a subsequent request before closing the connection. Once a request has been received, the specified value applies. The default is 15 seconds.
5. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
6. Restart Oracle HTTP Server. See Section 4.3.4.

The connection configuration settings are saved, and shown on the Performance Directives page.

5.3.3 Using Fusion Middleware Control to Set the Process Configuration

The child process and configuration settings impact the ability of the server to process requests. You may need to modify the settings as the number of requests increase or decrease to maintain a well-performing server.

For UNIX, the default number of child server processes is 2. For Microsoft Windows, the default number of threads to handle requests is 150.

To specify the process configuration using Fusion Middleware Control, do the following:

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **Performance Directives** from the Administration menu. The Performance Directives page appears.

3. Enter the number for the initial child server processes in the Initial Child Server Processes field (`StartServers` directive). This is the number of child server processes created when Oracle HTTP Server is started. The default is 2. This is for UNIX only.
4. Enter the number for the maximum idle threads in the Maximum Idle Threads field (`MaxSpareThreads` directive). An idle thread is a process that is running, but not handling a request.
5. Enter the number for the minimum idle threads in the Minimum Idle Threads field (`MinSpareThreads` directive).
6. Enter the number for the threads per child server process in the Threads per Child Server Process field (`ThreadsPerChild` directive).
7. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
8. Restart Oracle HTTP Server. See Section 4.3.4, "Restarting Oracle HTTP Server Instances".

The process configuration settings are saved, and shown on the Performance Directives page.

5.4 Understanding Process Security

By default, Oracle HTTP Server is not able to bind to ports on UNIX in the reserved range (typically less than 1024). To enable Oracle HTTP Server to listen on ports in the reserved range (for example, port 80 and port 443) on UNIX, see Section 4.3.2.4, "Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only)".

If your PL/SQL application is using the file system caching functionality in mod_plsql, then Oracle HTTP server should have read and write privileges to the cache directory, specified through the parameter `PlsqlCacheDirectory` in *DOMAIN_HOME*/config/fmwconfig/components/OHS/*componentName*/mod_plsql/cache.conf. By default, this parameter points to *DOMAIN_HOME*/servers/*componentName*.

Finally, given that the cached content might contain sensitive data, the contents of the file system cache should be protected. So, access to the system as this user should be well-protected.

See Also: Section 2.8, "mod_plsql"

6 Managing Connectivity

This chapter describes how to manage Oracle HTTP Server connectivity. It includes procedures for viewing port number usage, managing ports, and configuring virtual hosts.

This chapter includes the following sections:

6.1 Default Listen Ports

Oracle HTTP Server comes configured with two listen ports: a non-SSL port (http) and an SSL port (https). The default, non-SSL port is 7777. If port 7777 is occupied, the next available port number, within a range of 7777-7877, is assigned. The default SSL port is 4443. Similarly, if port 4443 is occupied, the next available port number, within a range of 4443-4543, is assigned.

You can set these ports when you create the instance or modify the instance configuration later. Automatic port assignment occurs only if you use `createOHSInstance()` or Fusion Middleware Control. You must do your own port management if you create instances by using the Configuration Wizard.

For information about specifying ports when creating a new Oracle HTTP Server component, see Section 4.2, "Creating an OHS Instance".

6.2 Defining the Admin Port

The Admin or *Proxy MBean* port is an additional SSL port (9999) that is used internally by Oracle HTTP Server to communicate with Fusion Middleware Control. It is also used to monitor Oracle HTTP Server through Node Manager. This port is configured to run out-of-the-box in the admin.conf file; however, if for any reason you need to use the default port for another purpose, you can reconfigure the Admin port by using the Configuration Wizard to update the domain and manually reset ports there.

6.3 Viewing Port Number Usage

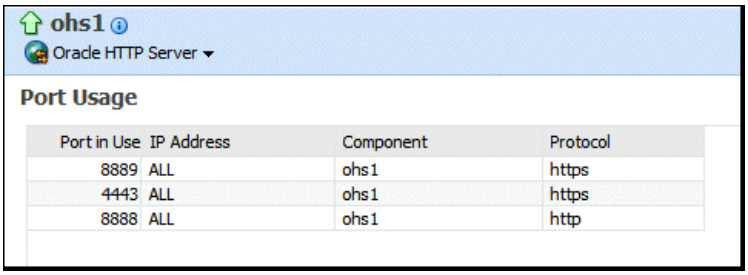
This section describes how to view ports using Fusion Middleware Control.

6.3.1 Using the Fusion Middleware Control to View Port Number Usage

To view the port number usage using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Port Usage** from the Oracle HTTP Server menu.

   The Port Usage detail page shows the component, the ports that are in use, the IP address the ports are bound to, and the protocol being used, as illustrated in the following figure:



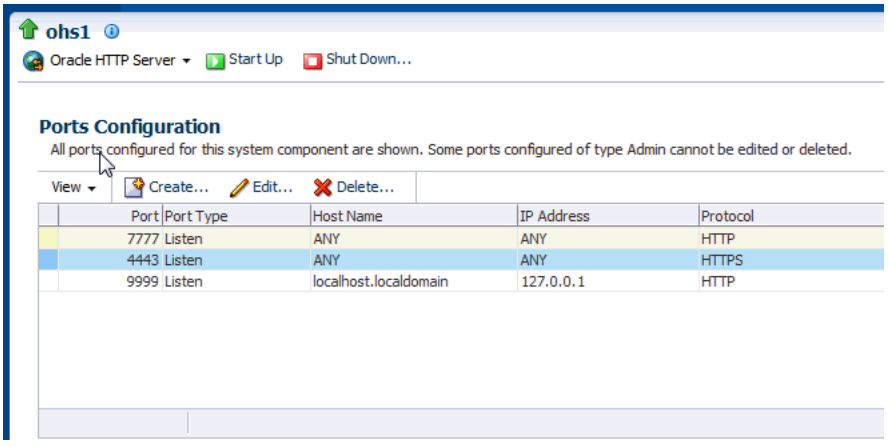
6.4 Managing Ports

The ports used by Oracle HTTP Server can be set during and after installation. In addition, you can change the port numbers, as needed. This section describes how to create, edit, and delete ports using Fusion Middleware Control.

Caution: The Oracle HTTP Server administration (proxy MBean) virtual host and its configuration, defined in the admin.conf file, must not be edited with the WebLogic Scripting Tool (WLST).

See Also: "Changing the Oracle HTTP Server Listen Ports" in the *Administering Oracle Fusion Middleware*.

- Using Fusion Middleware Control to Create Ports
- Using Fusion Middleware Control to Edit Ports

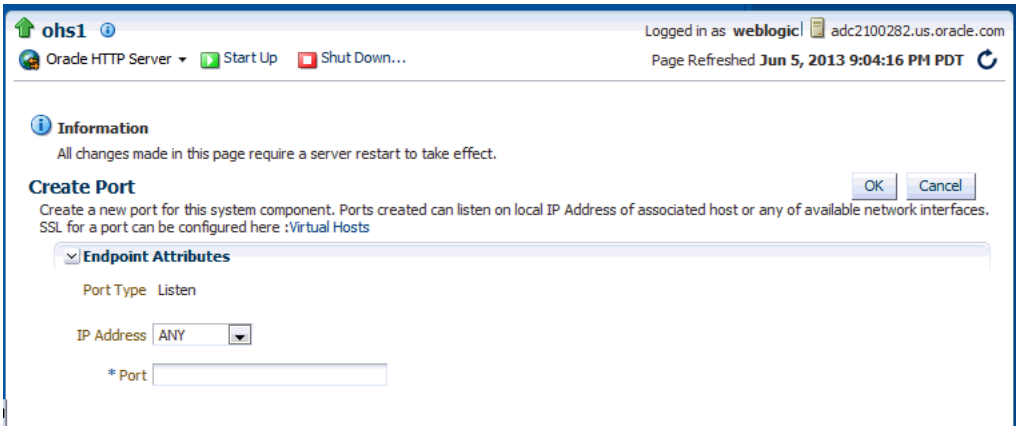


Note: When deleting a port, if there is a virtual host configured to use the port you want to delete, you must first delete that virtual host before deleting the port.

6.4.1 Using Fusion Middleware Control to Create Ports

To create ports using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Administration** from the Oracle HTTP Server menu.
3. Select **Ports Configuration** from the Administration menu.
4. Click **Create**.



5. Use the IP Address menu to select an IP address for the new port. Ports can listen on a local IP Address of an associated host or on any available network interfaces.

   SSL for a port can be configured on the Virtual Hosts page, as described in Section 6.5.2, "Using Fusion Middleware Control to Configure Virtual Hosts".
6. In **Port**, enter the port number.
7. Click **OK**.
8. Restart Oracle HTTP Server. See Section 4.3.4.

Note: If you change the port or make other changes that affect the URL, such as changing the host name, enabling or disabling SSL, you need to re-register partner applications with the SSO server using the new URL. For more information, see "Registering Oracle HTTP Server mod_osso with OSSO Server 10.1.4" in *Securing Applications with Oracle Platform Security Services*.

6.4.2 Using Fusion Middleware Control to Edit Ports

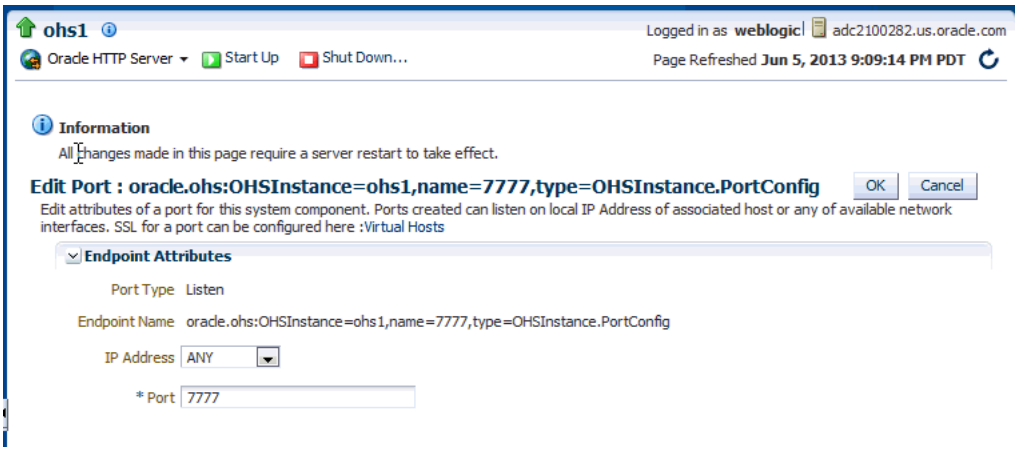
To create the ports using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Administration** from the Oracle HTTP Server menu.
3. Select **Ports Configuration** from the Administration menu.

4. Select the port for which you want to change the port number.

   The Admin port cannot be edited by using Fusion Middleware Control. Although this is a port Oracle HTTP Server uses for its internal communication with Fusion Middleware Control, in most of the cases it does not need to be changed. If you really want to change it, manually edit the *DOMAIN_HOME*/config/fmwconfig/components/OHS/*componentName*/admin.conf file.

5. Click **Edit**.



6. Edit the IP Address and/or Port number for the port.

   SSL for a port can be configured on the Virtual Hosts page, as described in Section 6.5.2, "Using Fusion Middleware Control to Configure Virtual Hosts".

7. Click **OK**.

8. Restart Oracle HTTP Server. See Section 4.3.4.

> **Note:** If you change the port or make other changes that affect the URL, such as changing the host name, enabling or disabling SSL, you need to re-register partner applications with the SSO server using the new URL.

6.4.3 Disabling a Listening Port in a Standalone Environment

While you can use Fusion Middleware Control to disable a listen port in a WebLogic Server environment, to do so in a standalone environment, you must directly update master configuration file (*DOMAIN_HOME*/config/fmwconfig/components/OHS/*componentName*/httpd.conf) by commenting-out the line where port is exposed; for example:

```
#Listen slc01qtd.us.myCo.com:7777
```

> **Note:** Before attempting to edit any .conf file, you should familiarize yourself with the layout of the configuration file directories, mechanisms for editing the files, and learn more about the files themselves. For this information, see Section 1.6, "Understanding Configuration Files".

6.5 Configuring Virtual Hosts

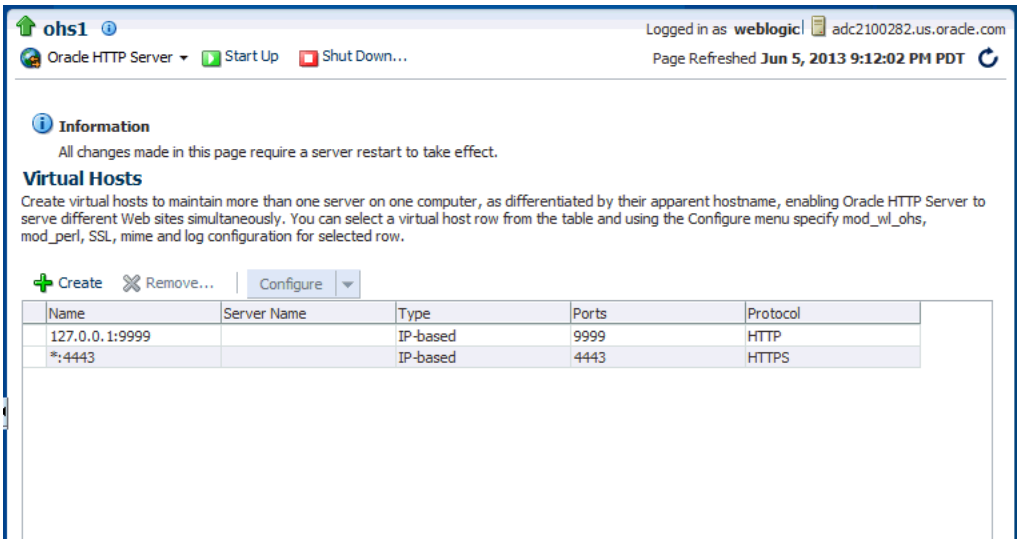
You can create virtual hosts to run more than one website (such as `www.company1.com` and `www.company2.com`) on a single machine. Virtual hosts can be *IP-based*, meaning that you have a different IP address for every website, or *name-based*, meaning that you have multiple names running on each IP address. The fact that they run on the same physical server is not apparent to the end user.

> **Caution:** The Oracle HTTP Server administration (proxy MBean) virtual host and its configuration, defined in the admin.conf file, must not be edited with the WebLogic Scripting Tool (WLST).

This section describes how to create and edit virtual hosts using Fusion Middleware Control.

- Using Fusion Middleware Control to Create Virtual Hosts
- Using Fusion Middleware Control to Configure Virtual Hosts

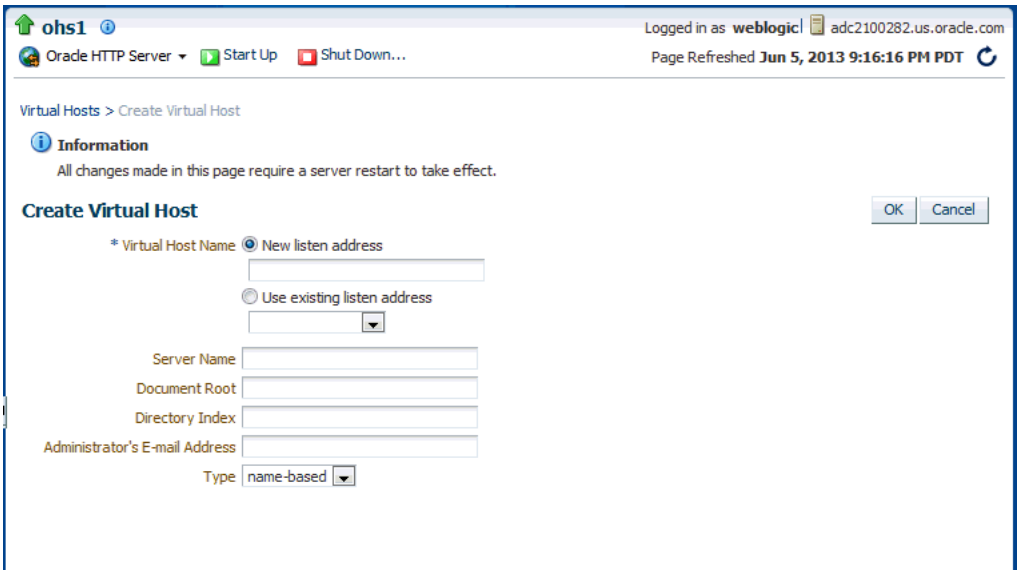
> **See Also:** For more information about virtual hosts, refer to the Apache HTTP Server documentation.



6.5.1 Using Fusion Middleware Control to Create Virtual Hosts

To create a virtual Host using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Administration** from the Oracle HTTP Server menu.
3. Select **Virtual Hosts** from the Administration menu.
4. Click **Create**.



5. Enter a name for the virtual host field and then choose whether to enter a new listen address or to use an existing listen address.

   - New listen address - use this option when you want to create a virtual host that maps to a specific hostname or IP address, for example mymachine.com:8080. This will create following type NameVirtualHost and VirtualHost directives:

     ```
     NameVirtualHost mymachine.com:8080
     <VirtualHost mymachine.com:8080>
     ```

   - Use existing listen address - use this option when you want to create a virtual host using an existing listen port and the one that maps to all IP addresses. This will create following type VirtualHost directive:

     ```
     <VirtualHost *:8080>
     ```

6. Enter the remaining attributes for the new virtual host.

   - **Server Name**—the name of the server for Oracle HTTP Server
   - **Document Root**— documentation root directory that forms the main document tree visible from the website
   - **Directory Index**—the main (index) page that will be displayed when a client first accesses the website
   - **Administrator's E-mail Address**—the e-mail address that the server will include in error messages sent to the client
   - **Type**—Virtual hosts can be IP-based, meaning that you have a different IP address for every website, or name-based, meaning that you have multiple names running on each IP address.

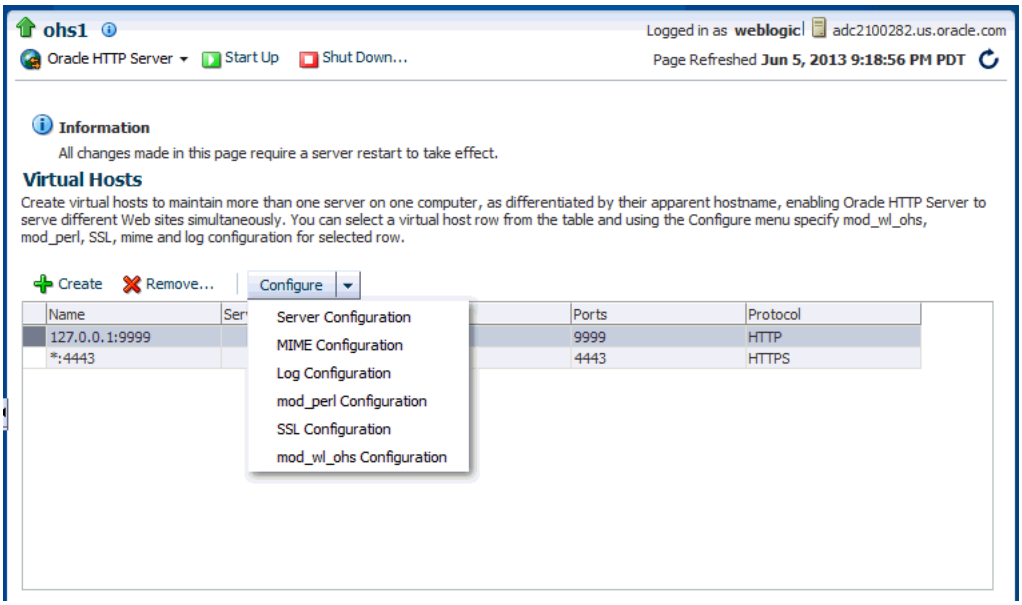
7. Use the **Type** field to select whether the virtual host will be IP-based or name-based.
8. Click **OK**.
9. Restart Oracle HTTP Server. See Section 4.3.4.

6.5.2 Using Fusion Middleware Control to Configure Virtual Hosts

You can use the options on the Configure menu to specify Server, MIME, Log, mod_perl, SSL, and mod_wl_ohs configuration for a selected virtual host.

To configure a virtual host using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Administration** from the Oracle HTTP Server menu.
3. Select **Virtual Hosts** from the Administration menu.
4. Highlight an existing virtual host in the table.
5. Click **Configure**.



6. Select one of the following options from Configure menu to open its corresponding configuration page. The values on these pages apply only to the virtual host. If the fields are blank, the virtual host uses the values configured at the server level.
   - Server Configuration: Configure basic virtual host properties, such as document root directory, installed modules, and aliases. See Section 4.5.1, "Specifying Server Properties by Using Fusion Middleware Control."
   - MIME Configuration: Configure MIME settings, which are used by Oracle HTTP Server to interpret file types, encodings, and languages. Section 4.6.3, "Configuring MIME Settings."
   - Log Configuration: Configure access logs that will record all requests processed by the virtual host. The logs contain basic information about every HTTP transaction handled by the virtual host. See Section 7.2, "Configuring Oracle HTTP Server Logs."
   - mod_perl Configuration: Configure the mod_perl module to embed the Perl interpreter into the virtual host, thereby eliminating startup overhead and enabling you to write modules in Perl. This module is disabled, by default. See Section 4.6.4, "Configuring mod_perl."

- SSL Configuration: For instructions on configuring SSL using Fusion Middleware Control, see "Enabling SSL for Oracle HTTP Server Virtual Hosts" in the *Administering Oracle Fusion Middleware*.
- mod_wl_ohs Configuration: Configure the mod_wl_ohs module to allow requests to be proxied from an Oracle HTTP Server to Oracle WebLogic Server. See Section 4.6.5, "Configuring the Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)."

7. Review the settings on each configuration page. If the settings are correct, click **OK** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Cancel** to return to the original settings.
8. Restart Oracle HTTP Server. See Section 4.3.4, "Restarting Oracle HTTP Server Instances".

7

Managing Oracle HTTP Server Logs

This chapter describes how to manage Oracle HTTP Server logs. It describes how to configure server logs, how to find information about the cause of an error and its corrective action, to view and manage log files to assist in monitoring system activity and to diagnose problems

Oracle HTTP Server generates log files containing messages that record all types of events, including startup and shutdown information, errors, warning messages, access information on HTTP requests, and additional information.

This chapter includes the following sections:

7.1 Overview of Server Logs

You can view Oracle Fusion Middleware log files using either Fusion Middleware Control or a text editor. The log files for Oracle HTTP Server are located in the following directory:

ORACLE_HOME/user_projects/domains/base_domain/servers/*componentName*/logs

Oracle HTTP Server has two types of logs:

- Error logs, which record server problems.
- Access logs, which record which components and applications are being accessed and by whom.

This section contains the following topics:

7.1.1 About Error Logs

Oracle HTTP Server enables you to choose the format in which you want to generate log messages. You can choose to generate log messages in the legacy Apache HTTP

Server message format, or use Oracle Diagnostic Logging (ODL) to generate log messages in text or XML-formatted logs, which complies with Oracle standards for generating error log messages.

By default, Oracle HTTP Server error logs use ODL for generating diagnostic messages. It provides a common format for all diagnostic messages and log files, and a mechanism for correlating the diagnostic messages from various components across Oracle Fusion Middleware.

The default name of the error log file is instance_name.log.

7.1.2 About Access Logs

Access logs record all requests processed by the server. The logs contain basic information about every HTTP transaction handled by the server. The access log contains the following information:

- Host name
- Remote log name
- Remote user and time
- Request
- Response code
- Number of transferred bytes

The default name of the access log file is access_log.

Access Log Format

You can specify the information to include in the access log, and the manner in which it is written. The default format is the Common Log Format (CLF).

The CLF format contains the following fields:

```
host ident authuser date request status bytes
```

- `host`: This is the client domain name or its IP number. Use `%h` to specify the host field in the log.
- `ident`: If IdentityCheck is enabled and the client system runs identd, this is the client identity information. Use `%i` to specify the client identity field in the log.
- `authuser`: This is the user ID for the authorized user. Use `%a` to specify the authorized user field in the log.
- `date`: This is the date and time of the request in the day/month/year:hour:minute:second format. Use `%t` to specify date and time in the log.
- `request`: This is the request line, in double quotes, from the client. Use `%r` to specify request in the log.
- `status`: This is the three-digit status code returned to the client. Use `%s` to specify the status in the log. If the request will be forwarded from another server, use `%>s` to specify the last server in the log.
- `bytes`: This is the number of bytes, excluding headers, returned to the client. Use `%b` to specify number of bytes in the log. Use `%i` to include the header in the log.

See Also: Access Log in the Apache HTTP Server documentation.

7.1.3 Log Rotation

Oracle HTTP Server supports two types of log rotation policies: size-based and time-based. You can configure the Oracle HTTP Server logs to use either of the two rotation polices, by using `odl_rotatelogs` in *ORACLE_HOME*/ohs/bin. By default, Oracle HTTP Server uses `odl_rotatelogs` for both error and access logs.

`odl_rotatelogs` supports all the features of Apache HTTP Server's `rotatelogs` and the additional feature of log retention.

You can find information about the features and options provided by `rotatelogs` at the following URL:

http://httpd.apache.org/docs/2.2/programs/rotatelogs.html

The following is the general syntax of `odl_rotatelogs`:

```
odl_rotatelogs [-u:offset] logfile {size-|time-based-rotation-options}
```

`odl_rotatelogs` is meant to be used with the piped logfile feature. This feature allows error and access log files to be written through a pipe to another process, rather than directly to a file. This increases the flexibility of logging, without adding code to the main server. To write logs to a pipe, replace the filename with the pipe character "|", followed by the name of the executable which should accept log entries on its standard input. For more information on the piped logfile feature, see the following URL:

http://httpd.apache.org/docs/2.2/logs.html#piped

Used with the piped logfile feature, the syntax of `odl_rotatelogs` becomes the following:

```
CustomLog " |${PRODUCT_HOME}/bin/odl_rotatelogs [-u:offset] logfile
{size-|time-based-rotation-options}" log_format
```

Whenever there is an input to `odl_rotatelogs`, it checks if the specified condition for rotation has been met. If so, it rotates the file. Otherwise it simply writes the content. If no input is provided, then it will do nothing.

Table 7–1 describes the size- and time-based rotation options:

Table 7–1 Options for odl_rotatelogs

Option	Description
`-u`	The time (in seconds) to offset from UTC.
`logfile`	The path and name of the log file, followed by a hyphen (-) and then the timestamp format. The following are the common timestamp format strings: ■ `%m`: Month as a two-digit decimal number (01-12) ■ `%d`: Day of month as a two-digit decimal number (01-31) ■ `%Y`: Year as a four-digit decimal number ■ `%H`: Hour of the day as a two-digit decimal number (00-23) ■ `%M`: Minute as a two-digit decimal number (00-59) ■ `%S`: Second as a two-digit decimal number (00-59) It should not include formats that expand to include slashes.
`frequency`	The time (in seconds) between log file rotations.
`retentionTime`	The maximum time for which the rotated log files are retained.
`startTime`	The time when time-based rotation should start.

Table 7–1 (Cont.) Options for odl_rotatelogs

Option	Description
`maxFileSize`	The maximum size (in MB) of log files.
`allFileSize`	The total size (in MB) of files retained.

Syntax and Examples for Time- and Size-Based Rotation

- Time-based rotation

  Syntax:

  ```
  odl_rotatelogs u:offset logfile frequency retentionTime startTime
  ```

  Example:

  ```
  CustomLog "| odl_rotatelogs u:-18000 /varlog/error.log-%Y-%m-%d 21600 172800
  2014-03-10T08:30:00" common
  ```

  This configures log rotation to be performed for a location UTC-05:00 (18000 seconds, such as New York). The rotation will be performed every 21600 seconds (6 hours) starting from 8:30 a.m. on March 10, 2014, and it specifies that the rotated log files should be retained for 172800 seconds (2 days). The log format is `common`.

  Syntax:

  ```
  odl_rotatelogs logfile frequency retentionTime startTime
  ```

  Example:

  ```
  CustomLog "| odl_rotatelogs /varlog/error.log-%Y-%m-%d 21600 172800
  2014-03-10T08:30:00" common
  ```

  This configures log rotation to be performed every 21600 seconds (6 hours) starting from 8:30 a.m. on March 10, 2014, and it specifies that the rotated log files should be retained for 172800 seconds (2 days). The log format is `common`.

- Size-based rotation

  Syntax:

  ```
  odl_rotatelogs logfile maxFileSize allFileSize
  ```

  Example:

  This configures log rotation to be performed when the size of the log file reaches 10 MB, and it specifies the maximum size of all the rotated log files as 70 MB (up to 7 log files (=70/10) will be retained). The log format is `common`.

  ```
  CustomLog "| odl_rotatelogs /var/log/error.log-%Y-%m-%d 10M 70M" common
  ```

7.2 Configuring Oracle HTTP Server Logs

You can use Fusion Middleware Control to configure error and access logs. The following logging tasks can be set from the Log Configuration page:

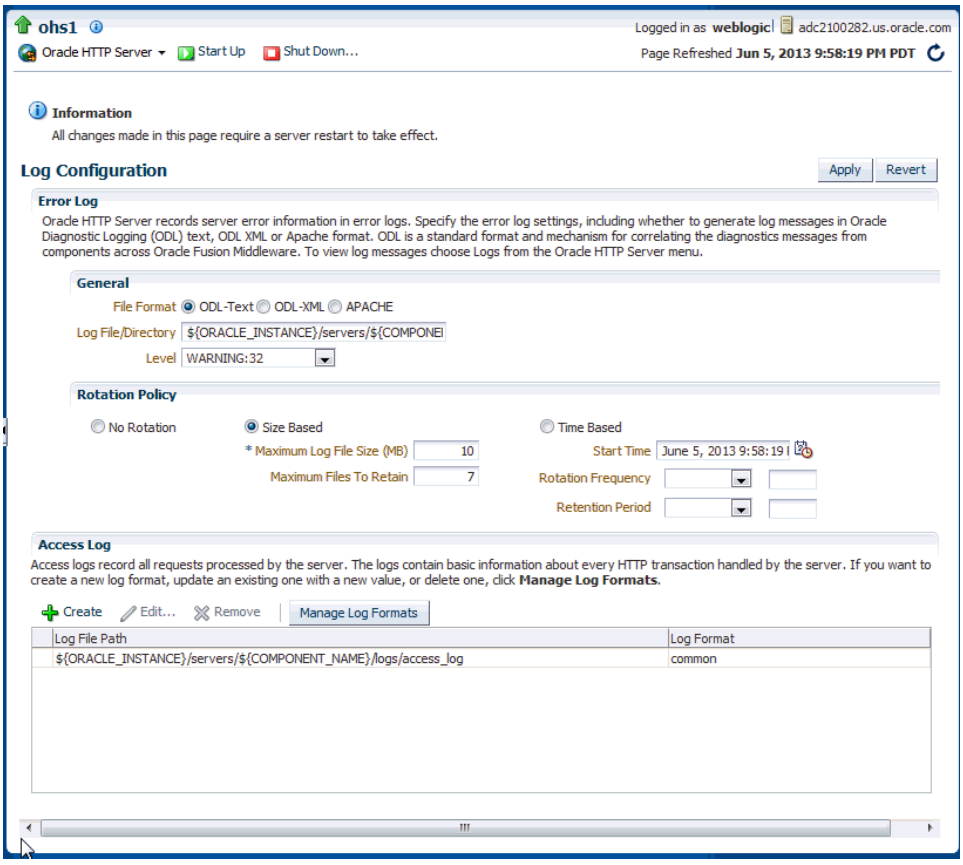
- Using Fusion Middleware Control to Configure Error Logs
- Configuring Access Logs by Using Fusion Middleware Control

7.2.1 Using Fusion Middleware Control to Configure Error Logs

To configure an error log for Oracle HTTP Server using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Log Configuration** from the Administration menu.

   The Log Configuration page is displayed, as shown in the following figure.



3. The following error log configuration tasks can be set from this page:
   - Configuring the Error Log Format and Location
   - Configuring the Error Log Level
   - Configuring Error Log Rotation Policy

7.2.1.1 Configuring the Error Log Format and Location

Oracle HTTP Server by default uses ODL-Text as the error log format and creates the log file with the name `component_name.log` under the *DOMAIN_HOME*/servers/*component_name*/logs directory. To use a different format or log location, do the following:

1. From the Log Configuration page, navigate to the General section under the Error Log section.
2. Select the desired file format. Although both ODL-Text and ODL-XML formats provide the same information, the ODL-XML file includes XML elements and wrappers, and so may be easier to read.

- ODL-Text: the format of the diagnostic messages conform to an Oracle standard and are written in text format.
- ODL-XML: the format of the diagnostic messages conform to an Oracle standard and are written in XML format.
- Apache: the format of the diagnostic messages conform to the legacy Apache HTTP Server message format.

3. Enter a path for the error log in the Log File/Directory field. This directory must exist before you enter it here.
4. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
5. Restart Oracle HTTP Server. See Section 4.3.4.

7.2.1.2 Configuring the Error Log Level

You can configure the amount and type of information written to log files by specifying the message type and level. Error log level for Oracle HTTP Server by default is configured to WARNING:32. To use a different error log level do the following:

1. From the Log Configuration page, navigate to the General section under the Error Log section.
2. Select a level for the logging from the Level menu. The higher the log level, the more information that is included in the log.
3. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
4. Restart Oracle HTTP Server. See Section 4.3.4.

Note: The log levels are different for the Apache HTTP Server log format from ODL-Text and the ODL-XML log format.

- For details on ODL log levels, refer to "Setting the Level of Information Written to Log Files" in the *Administering Oracle Fusion Middleware*.
- For details on Apache HTTP Server log levels, refer to the LogLevel Directive in the Apache HTTP Server documentation.

7.2.1.3 Configuring Error Log Rotation Policy

Log rotation policy for error logs can either be time-based, such as once a week, or sized-based, such as 120MB. By default, the error log file is rotated when it reaches 10 MB in size and a maximum of 7 error log files will be retained. To use a different rotation policy, do the following:

1. From the Log Configuration page, navigate to the General section under the Error Log section.
2. Select a rotation policy.
   - No Rotation: if you do not want to have the log file rotated ever.

- Size Based: rotate the log file whenever it reaches a configured size. Set the maximum size for the log file in Maximum Log File Size (MB) field and the maximum number of error log files to retain in Maximum Files to Retain field.
- Time Based: rotate the log file whenever configured time is reached. Set the start time, rotation frequency, and retention period.

3. Review the settings. If the settings are correct, click **Apply** to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click **Revert** to return to the original settings.
4. Restart Oracle HTTP Server. See Section 4.3.4.

7.2.2 Configuring Access Logs by Using Fusion Middleware Control

To configure an access log for Oracle HTTP Server using Fusion Middleware Control, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Log Configuration** from the Administration menu.

The following access log configuration tasks can be set from this page:

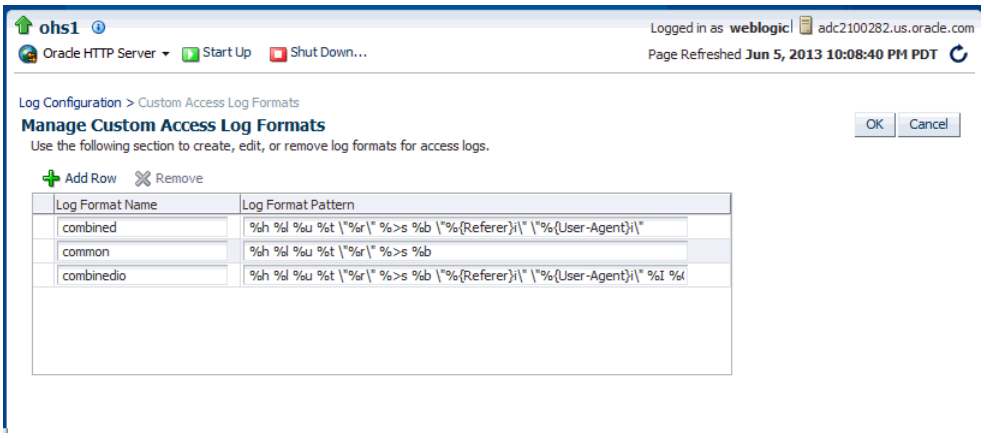
- Configuring the Access Log Format
- Configuring the Access Log File

7.2.2.1 Configuring the Access Log Format

Log format specifies the information included in the access log file and the manner in which it is written. To add a new access log format or to edit or remove an existing format, do the following:

1. From the Log Configuration page, navigate to the Access Log section.
2. Click **Manage Log Formats**.

   The Manage Custom Access Log Formats page is displayed, as shown in the following figure.



3. Select an existing format to change or remove, or click **Add Row** to create a new format.
4. If you choose to create a new format, then enter the new log format in the **Log Format Name** field and the log format in the **Log Format Pattern** field.

See Also: Refer to the Apache HTTP Server documentation for information about log format directives.

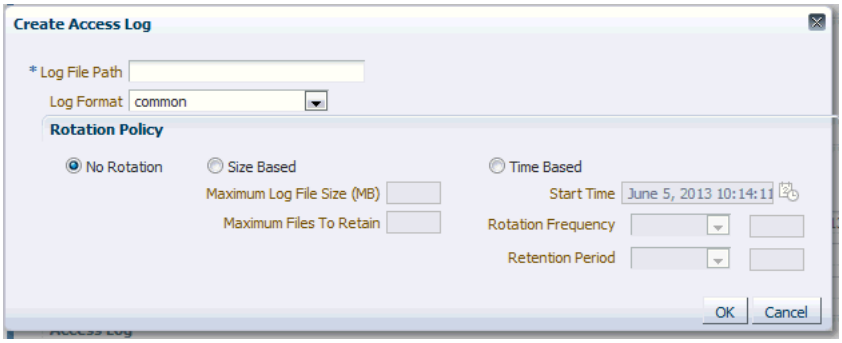
5. Click **OK** to save the new format.

7.2.2.2 Configuring the Access Log File

To configure an access log for file Oracle HTTP Server, do the following:

1. From the Log Configuration page, navigate to the Access Log section.
2. Click **Create** to create a new access log, or select a row from the table and click **Edit** button to edit an existing access log file.

   The Create or Edit Access Log page is displayed.



3. Enter the path for the access log in the Log File Path field. This directory must exist before you enter it.
4. Select an existing access log format from the Log Format menu.
5. Select a rotation policy.
   - No Rotation: if you do not want to have the log file rotated ever.
   - Size Based: rotate the log file whenever it reaches a configured size. Set the maximum size for the log file in Maximum Log File Size (MB) field and the maximum number of error log files to retain in Maximum Files to Retain field.
   - Time Based: rotate the log file whenever configured time is reached. Set the start time, rotation frequency, and retention period.
6. Click **OK** to continue.

   Note that you can create multiple access log files.

7.3 Log Directives for Oracle HTTP Server

This section discuss Oracle HTTP Server error and access log-related directives in the `httpd.conf` file. The directives discussed are:

- Oracle Diagnostic Logging Directives
- Apache HTTP Server Log Directives

7.3.1 Oracle Diagnostic Logging Directives

Oracle HTTP Server by default uses Oracle Diagnostic Logging (ODL) for generating diagnostic messages. The following directives are used to set up logging using ODL:

- OraLogMode
- OraLogDir
- OraLogSeverity
- OraLogRotationParams

7.3.1.1 OraLogMode

Enables you to choose the format in which you want to generate log messages. You can choose to generate log messages in the legacy Apache HTTP Server, ODL text, or ODL XML format.

```
OraLogMode Apache | ODL-Text | ODL-XML
```

Default value: `ODL-Text`

For example: `OraLogMode ODL-XML`

> **Note:** The Apache HTTP Server log directives `ErrorLog` and `LogLevel` are only effective when `OraLogMode` is set to `Apache`. When `OraLogMode` is set to either `ODL-Text` or `ODL-XML`, the `ErrorLog` and `LogLevel` directives are ignored.

7.3.1.2 OraLogDir

Specifies the path to the directory that contains all log files. This directory must exist.

This directive is used only when OraLogMode is set to either `ODL-Text` or `ODL-XML`. When `OraLogMode` is set to `Apache`, `OraLogDir` is ignored and ErrorLog is used instead.

```
OraLogDir <path>
```

Default value: *ORACLE_INSTANCE*/servers/*componentName*/logs

For example: `OraLogDir /tmp/logs`

7.3.1.3 OraLogSeverity

Enables you to set message severity. The message severity specified with this directive is interpreted as the lowest desired message severity, and all messages of that severity level and higher are logged.

This directive is used only when OraLogMode is set to either `ODL-Text` or `ODL-XML`. When `OraLogMode` is set to `Apache`, `OraLogSeverity` is ignored and LogLevel is used instead.

```
OraLogSeverity <msg_type>[:msg_level]
```

Default value: `WARNING:32`

For example: `OraLogSeverity NOTIFICATION:16`

msg_type

Message types can be specified in upper or lower case, but appear in the message output in upper case. This parameter must be of one of the following values:

- INCIDENT_ERROR
- ERROR
- WARNING

- NOTIFICATION
- TRACE

msg_level

This parameter must be an integer in the range of 1–32, where 1 is the most severe, and 32 is the least severe. Using level 1 will result in fewer messages than using level 32.

7.3.1.4 OraLogRotationParams

Enables you to choose the rotation policy for an error log file. This directive is used only when OraLogMode is set to either `ODL-Text` or `ODL-XML`. When `OraLogMode` is set to `Apache`, `OraLogRotationParams` is ignored.

`OraLogRotationParams <rotation_type> <rotation_policy>`

Default value: `S 10:70`

For example: `OraLogRotationParams T 43200:604800 2009-05-08T10:53:29`

rotation_type

This parameter can either be `S` (for sized-based rotation) or `T` (for time-based rotation).

rotation_policy

When `rotation_type` is set to `S` (sized-based), set the rotation_policy parameter to:

`maxFileSize:allFilesSize` (in MB)

For example, when configured as `10:70`, the error log file is rotated whenever it reaches 10MB in size and a total of 70MB is allowed for all error log files (a maximum of 70/10=7 error log files will be retained).

When rotation_type is set to `T` (time-based), set the rotation_policy parameter to:

`frequency(in sec) retentionTime(in sec) startTime(in YYYY-MM-DDThh:mm:ss)`

For example, when configured as `43200:604800 2009-05-08T10:53:29`, the error log is rotated every 43200 seconds (that is, 12 hours), rotated log files are retained for maximum of 604800 seconds (7 days) starting from May 5, 2009 at 10:53:29.

7.3.2 Apache HTTP Server Log Directives

Although Oracle HTTP Server uses ODL by default for error logs, you can configure the `OraLogMode` directive to `Apache` to generate error log messages in the legacy Apache HTTP Server message format. The following directives are discussed in this section:

- ErrorLog
- LogLevel
- LogFormat
- CustomLog

7.3.2.1 ErrorLog

The `ErrorLog` directive sets the name of the file where the server logs any errors it encounters. If the filepath is not absolute then it is assumed to be relative to the ServerRoot.

This directive is used only when `OraLogMode` is set to `Apache`. When `OraLogMode` is set to either `ODL-Text` or `ODL-XML`, ErrorLog is ignored and `OraLogDir` is used instead.

> **See Also:** For information about the Apache ErrorLog directive, see:
>
> http://httpd.apache.org/docs/current/mod/core.html#errorlog

7.3.2.2 LogLevel

The `LogLevel` directive adjusts the verbosity of the messages recorded in the error logs.

This directive is used only when 3 is set to `Apache`. When `OraLogMode` is set to either `ODL-Text` or `ODL-XML`, LogLevel is ignored and OraLogSeverity is used instead.

> **See Also:** For information about the Apache HTTP Server LogLevel directive see:
>
> http://httpd.apache.org/docs/current/mod/core.html#loglevel

7.3.2.3 LogFormat

The `LogFormat` directive specifies the format of the access log file. By default, Oracle HTTP Server comes with the following four access log formats defined:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
```

> **See Also:** For information about the Apache HTTP Server LogFormat directive, see:
>
> http://httpd.apache.org/docs/current/mod/mod_log_config.html#logformat

7.3.2.4 CustomLog

The `CustomLog` directive is used to log requests to the server. A log format is specified and the logging can optionally be made conditional on request characteristics using environment variables. By default, the access log file is configured to use the common log format.

> **See Also:** For information about the Apache CustomLog directive, see:
>
> http://httpd.apache.org/docs/current/mod/mod_log_config.html#customlog

7.4 Viewing Oracle HTTP Server Logs

You can search, view, and list Oracle HTTP Server log files using Fusion Middleware Control, or you can download a log file to your local client and view the log files using another tool.

You can also use the text editor of your choice to view Oracle HTTP Server log files directly from the *DOMAIN_HOME* directory. By default, Oracle HTTP Server log files are located in the *DOMAIN_HOME*/servers/*component_name*/logs directory.

As discussed in Section 7.1, "Overview of Server Logs", there are mainly two types of log files for Oracle HTTP Server: error logs and access logs. The error log file is an

important source of information for maintaining a well-performing server. The error log records all of the information about problem situations so that the system administrator can easily diagnose and fix the problems. The access log file contains basic information about every HTTP transaction that the server handles. This information can be used to generate statistical reports about the server's usage patterns.

> **See Also:** For information about searching and viewing log files, see the *Administering Oracle Fusion Middleware*

7.5 Terminating SSL Requests

This section describes how to terminate SSL before or within Oracle HTTP Server, where the `mod_wl_ohs` module is used to forward requests to WebLogic Server. Whether you terminate SSL before the request reaches Oracle HTTP Server or when the request is in the server, depends on your topology. A common reason to terminate SSL is for performance considerations when an internal network is otherwise protected with no risk of a third-party intercepting data within the communication. Another reason is when WebLogic Server is not configured to accept HTTPS requests.

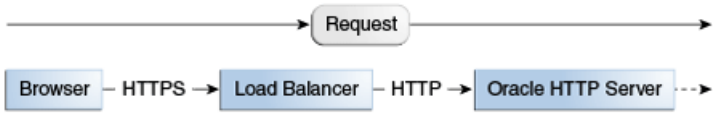
- Terminating SSL Before Oracle HTTP Server
- Terminating SSL at Oracle HTTP Server

7.5.1 Terminating SSL Before Oracle HTTP Server

If you are using another device such as a load balancer or a reverse proxy which terminates requests using SSL before reaching Oracle HTTP Server, then you must configure the server to treat the requests as if they were received through HTTPS. The server must also be configured to send HTTPS responses back to the client.

Figure 7–1 illustrates an example where the request transmitted from the browser through HTTPS to WebLogic Server. The load balancer terminates SSL and transmits the request as HTTP. Oracle HTTP Server must be configured to treat the request as if it was received through HTTPS.

Figure 7–1 Terminating SSL Before Oracle HTTP Server



To instruct the Oracle HTTP Server to treat requests as if they were received through HTTPS, configure the `httpd.conf` file with the `SimulateHttps` directive in the `mod_certheaders` module.

For more information on `mod_certheaders` module, see Section G.1, "mod_certheaders."

> **Note:** This procedure is not necessary if SSL is configured on Oracle HTTP Server (that is, if you are directly accessing Oracle HTTP Server using HTTPS).

1. Configure the `httpd.conf` configuration file with the external name of the server and its port number, for example:

```
ServerName <www.company.com:port>
```

2. Configure the `httpd.conf` configuration file to load the `mod_certheaders` module, for example:

   - On UNIX:

     ```
     LoadModule certheaders_module libexec/mod_certheaders.so
     ```

   - On Windows:

     ```
     LoadModule certheaders_module modules/ApacheModuleCertHeaders.dll
     AddModule mod_certheaders.c
     ```

     Note: Oracle recommends that the `AddModule` line should be included with other `AddModule` directives.

3. Configure the `SimulateHttps` directive at the bottom of the `httpd.conf` file to send HTTPS responses back to the client, for example:

   ```
   # For use with other load balancers and front-end devices:
   SimulateHttps On
   ```

4. Restart Oracle HTTP Server and test access to the server. Especially, test whether you can access static pages such as `https://`*`host:port`*`/index.html`

   Test your configuration as a basic setup. If you are having issues, then you should troubleshoot from here to avoid overlapping with other potential issues, such as with virtual hosting.

5. Ideally, you may want to configure a `VirtualHost` in the `httpd.conf` file to handle all HTTPS requests. This separates the HTTPS requests from the HTTP requests as a more scalable approach. This may be more desirable in a multi-purpose site or if a load balancer or other device is in front of Oracle HTTP Server which is also handling both HTTP and HTTPS requests.

   The following sample instructions load the `mod_certheaders` module, then creates a virtual host to handle only HTTPS requests.

   ```
   # Load correct module here or where other LoadModule lines exist:
   LoadModule certheaders_module libexec/mod_certheaders.so
   # This only handles https requests:
   NameVirtualHost <name>:<port>
      <VirtualHost <name>:<port>
          # Use name and port used in url:
          ServerName <www.company.com:port>
          SimulateHttps On
          # The rest of your desired configuration for this VirtualHost goes here
      </VirtualHost>
   ```

6. Restart Oracle HTTP Server and test access to the server, First test a static page such as `https://`*`host:port`*`/index.html` and then your test your application.

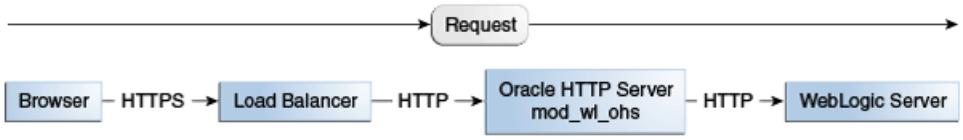
7.5.2 Terminating SSL at Oracle HTTP Server

If SSL is configured in Oracle HTTP Server but not on WebLogic Server, then you can terminate SSL for requests sent by Oracle HTTP Server.

The following figures illustrate request flows, showing where HTTPS stops. In Figure 7–2, an HTTPS request is sent from the browser. The load balancer transmits the

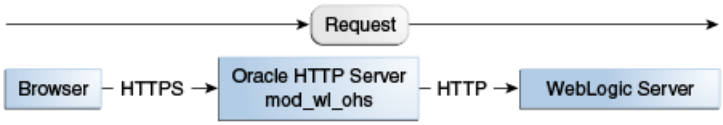
HTTPS request to Oracle HTTP Server. SSL is terminated in Oracle HTTP Server and the HTTP request is sent to WebLogic Server.

Figure 7–2 Terminating SSL at Oracle HTTP Server—With Load Balancer



In Figure 7–3, there is no load balancer and the HTTPS request is sent directly to Oracle HTTP Server. Again, SSL is terminated in Oracle HTTP Server and the HTTP request is sent to WebLogic Server.

Figure 7–3 Terminating SSL at Oracle HTTP Server—Without Load Balancer



1. Configure the `mod_wl_ohs.conf` file to add the `WLSProxySSL` directive for the location of your non-SSL configured managed servers, for example:

   ```
   WLProxySSL ON
   ```

2. If using a load balancer or other device in front of Oracle HTTP Server (which is also using SSL), you might need to configure the `WLProxySSLPassThrough` directive instead, depending on if it already sets `WL-Proxy-SSL`, for example:

   ```
   WLProxySSLPassThrough ON
   ```

   For more information, see your load balancer documentation. For more information on WLProxySSLPassThrough, see "Parameters for Web Server Plug-Ins" in *Using Oracle WebLogic Server Proxy Plug-Ins 12.1.2*.

3. Ensure that the `SecureProxy` directive is **not** configured, as it will interfere with the intended communication between the components. This directive is to be used only when SSL is used throughout. The `SecureProxy` directive is commented out in the following example:

   ```
   # To configure SSL throughout (all the way to WLS):
   # SecureProxy ON
   # WLSSLWallet  "<Path to Wallet>"
   ```

4. Restart Oracle HTTP Server and test access to a Java application, for example: `https://host:port/path/application_name`.

8

Managing Application Security

This chapter contains an overview of Oracle HTTP Server security features and provides configuration information for setting up a secure website.

This chapter includes the following sections:

8.1 About Oracle HTTP Server Security

Security can be organized into the three categories of authentication, authorization, and confidentiality. Oracle HTTP Server provides support for all three of these categories. It is based on the Apache HTTP Server, and its security infrastructure is primarily provided by the Apache modules, mod_auth_basic, mod_authn_file, mod_auth_user, and mod_authz_groupfile, and WebGate. The mod_auth_basic, mod_authn_file, mod_auth_user, and mod_authz_groupfile modules provide authentication based on user name and password pairs, while mod_authz_host controls access to the server based on the characteristics of a request, such as host name or IP address, mod_ossl provides confidentiality and authentication with X.509 client certificates over SSL.

Oracle HTTP Server provides access control, authentication, and authorization methods that can be configured with access control directives in the `httpd.conf` file. When URL requests arrive at Oracle HTTP Server, they are processed in a sequence of steps determined by server defaults and configuration parameters. The steps for handling URL requests are implemented through a module or plug-in architecture that is common to many Web listeners.

8.2 Classes of Users and Their Privileges

Oracle HTTP Server authorizes and authenticates users before allowing them to access, or modify resources on the server. The following are three classes of users that access the server using Oracle HTTP Server, and their privileges:

- Users who access the server without providing any authentication. They have access to unprotected resources only.
- Users who have been authenticated and potentially authorized by modules within Oracle HTTP Server. This includes users authenticated by Apache HTTP Server

modules like mod_auth_basic, mod_authn_file, mod_auth_user, and mod_authz_groupfile modules and Oracle's mod_ossl. Such users have access to URLs defined in `http.conf` file.

> **See Also:** Section 8.4, "Authentication, Authorization and Access Control".

- Users who have been authenticated through Oracle Access Manager. These users have access to resources allowed by Single Sign-On.

> **See Also:** *Securing Applications with Oracle Platform Security Services*

8.3 Resources Protected

Oracle HTTP Server can be configured to protect all resources that it manages. You are responsible for configuring any protection that your resources require.

8.4 Authentication, Authorization and Access Control

Oracle HTTP Server provides user authentication and authorization at two stages:

- Access Control (stage one): This is based on the details of the incoming HTTP request and its headers, such as IP addresses or host names.
- User Authentication and Authorization (stage two): This is based on different criteria depending on the HTTP server configuration. The server can be configured to authenticate users with user name and password pairs that are checked against a list of known users and passwords. You can also configure the server to use single sign-on authentication for Web applications or X.509 client certificates over SSL.

8.4.1 Access Control

Access control refers to any means of controlling access to any resource.

> **See Also:** Refer to the Apache HTTP Server documentation for more information on how to configure access control to resources.

8.4.2 User Authentication and Authorization

Authentication is any process by which you verify that someone is who they claim they are. Authorization is any process by which someone is allowed to be where they want to go, or to have information that they want to have.

8.4.2.1 Using Apache HTTP Server Modules to Authenticate Users

Access control refers to any means of controlling access to any resource.

> **See Also:** For more information on how to authenticate users, see the Apache HTTP Server documentation on "Authentication and Authorization" at:
>
> `http://httpd.apache.org/docs/2.2/howto/auth.html`

8.4.2.2 Using WebGate to Authenticate Users

WebGate enables single sign-on (SSO) for Oracle HTTP Server. WebGate examines incoming requests and determines whether the requested resource is protected, and if so, retrieves the session information for the user.

Through WebGate, Oracle HTTP Server becomes an SSO partner application enabled to use SSO to authenticate users, obtain their identity by using Oracle Single Sign-On, and to make user identities available to web applications accessed through Oracle HTTP Server.

By using WebGate, web applications can register URLs that require SSO authentication. WebGate detects which requests received by Oracle HTTP Server require SSO authentication, and redirects them to the SSO server. Once the SSO server authenticates the user, it passes the user's authenticated identity back to WebGate in a secure token. WebGate retrieves the user's identity from the token and propagates it to applications accessed through Oracle HTTP Server, including applications running in Oracle WebLogic Server and CGIs and static files handled by Oracle HTTP Server.

> **See Also:** *Securing Applications with Oracle Platform Security Services*

8.4.3 Support for FMW Audit Framework

Oracle HTTP Server supports authentication and authorization auditing by using the FMW Common Audit Framework. As part of enabling auditing, Oracle HTTP Server supports a directive called `OraAuditEnable`, which defaults to `On`. When it is enabled, audit events enabled in auditconfig.xml will be recorded in an audit log. By default, no audit events are enabled in auditconfig.xml.

When `OraAuditEnable` is set to `Off`, auditing is disabled regardless of the settings in auditconfig.xml.

Audit filters can be configured using Fusion Middleware Control or by editing auditconfig.xml directly.

> **See Also:** "Overview of Audit Features" in *Securing Applications with Oracle Platform Security Services*

8.5 Disable SSLv2 and SSLv3 Security Protocols

Because of security concerns, Oracle strongly recommends that you disable the SSLv3 security protocol from Oracle HTTP Server.

To disable SSL security protocols from Oracle HTTP Server:

1. Locate the `ssl.conf` file in the staging directory and the runtime directory.

   You can find the `ssl.conf` files in the following locations:

   Staging directory: *`DOMAIN_HOME`*`/config/fmwconfig/components/OHS/`*`componentName`*

   Runtime directory: `DOMAIN_HOME/config/fmwconfig/components/OHS/instances/`*`componentName`*

2. Edit the security declaration to use a non-SSL protocol.

   For example, to remove the SSLv3 security protocol:

   ```
   SSLProtocol -SSLv3
   ```

or to add the TLS version 1.0 and 1.2 security protocols:

```
SSLProtocol nzos_Version_1_1 nzos_Version_1_2
```

or to add the TLS version 1.0, 1.1, and 1.2 security protocols:

```
SSLProtocol nzos_Version_1_0 nzos_Version_1_1 nzos_Version_1_2
```

3. Save the files and restart Oracle HTTP Server.

Note: ■

- If you are editing files manually, ensure you edit a currently configured value instead of adding another. It could be easy to add a global parameter when it will be overridden by a value in the VirtualHost.
- Using the new `nzos_Version_*` syntax is now preferred. If you are using Oracle Fusion Middleware Control, this is how security will be configured.

Part III

Appendixes and Glossary

This part contains the following appendices plus a glossary:

A

OHS Introspector Plug-in for OVAB

The Oracle HTTP Server (OHS) introspector plug-in for the Oracle Virtual Assembly Builder (OVAB) plug-in introspects all the available Oracle HTTP Server instances in a WebLogic Server domain. This plug-in is an extension of WebLogic Server plug-in for OVAB.

This chapter contains the following sections:

A.1 Versions Supported

This plug-in supports version 12.1.2.

A.2 Oracle HTTP Server Introspection Parameters

The OHS Introspector plug-in for OVAB is an extension of WebLogic Server plug-in for OVAB, thus it works with the Introspector parameters provided for the WLS plug-in.

For the parameters required by WebLogic Server, see "Using the Introspection Plug-in for Oracle Virtual Assembly Builder," in *Administering Server Environments for Oracle WebLogic Server*.

A.3 Resulting Artifact Type

Multiple scalable appliances, one per Oracle HTTP Server instance.

A.4 Requirements

All of WebLogic Server requirements must be satisfied. For these requirements, see "Using the Introspection Plug-in for Oracle Virtual Assembly Builder," in *Administering Server Environments for Oracle WebLogic Server*.

In addition to the WebLogic Server requirements, reference system implementations require that WLS and Oracle HTTP Server be installed in the same *ORACLE_HOME*.

A.5 Wiring

Inputs are created on the Oracle HTTP Server appliance for each Listen or Port directive found in the configuration. The protocol of an Oracle HTTP Server input is set to `http` unless the `Listen` directive is found inside a `VirtualHost` directive and the directive `SSLEngine` is set to *on*. In that case, the protocol is `https`.

Outputs on the Oracle HTTP Server appliance are created based on various directives related to Oracle WebLogic Server in the Oracle HTTP Server configuration. The outputs indicate which inputs on an Oracle WebLogic Server assembly to connect to through the output 'description'.

A.6 Wiring Properties

All instance appliance input endpoints have one editable property, `port`, and two non-editable properties, `name` and a list of `protocol`s. The `protocol`s indicate what sort of outputs can be connected to the input. An administration server appliance will always have one secure http listener input endpoint, `port`, which is editable.

All output endpoints have three non-editable properties, `description`, `protocol` and `singleton`. The `protocol` indicates what sort of input can be connected to the output. `Singleton` indicates what sort of appliance the output can be connected to. If `singleton` is true, the output can only be connected to an input on an appliance that has a scalability absolute max value of 1. Administration Server appliance do not have output endpoints.

A.7 Oracle HTTP Server Appliance Properties

There are no relevant Oracle HTTP Server appliance properties.

A.8 Extensions of the Plug-in

None.

A.9 Supported Template Types

The supported template type is Oracle Enterprise Linux (OEL).

A.10 Plug-in Limitations

Be aware of the following plug-in limitations:

- Any changes done manually to Oracle HTTP Server instance(s) on the reference system without the administration interfaces will not be introspected by the plug-in.

- On the reference implementation, the OVAB Oracle HTTP Server plug-in does not introspect Oracle HTTP Server standalone deployments. It only supports WebLogic Server Oracle HTTP Server deployments.

A.11 Related Documents

For more information on using OVAB, see the following documents:

- *Developing Applications and Introspection Plug-ins for Oracle Virtual Assembly Builder*
- *Using Oracle Virtual Assembly Builder*

B

Frequently Asked Questions

This appendix provides answers to frequently asked questions about Oracle HTTP Server (OHS). It includes the following topics:

Documentation from the Apache Software Foundation is referenced when applicable.

> **Note:** Readers using this guide in PDF or hard copy formats will be unable to access third-party documentation, which Oracle provides in HTML format only. To access the third-party documentation referenced in this guide, use the HTML version of this guide and click the hyperlinks.

B.1 How Do I Create Application-Specific Error Pages?

Oracle HTTP Server has a default content handler for dealing with errors. You can use the `ErrorDocument` directive to override the defaults.

See Also: Apache HTTP Server documentation on the `ErrorDocument` directive at:

http://httpd.apache.org/docs/current/mod/core.html#errordocument

B.2 What Type of Virtual Hosts Are Supported for HTTP and HTTPS?

For HTTP, Oracle HTTP Server supports both name-based and IP-based virtual hosts. Name-based virtual hosts are virtual hosts that share a common listening address (IP plus port combination), but route requests based on a match between the Host header sent by the client and the `ServerName` directive set within the `VirtualHost`. IP-based virtual hosts are virtual hosts that have distinct listening addresses. IP-based virtual hosts route requests based on the address they were received on.

For HTTPS, only IP-based virtual hosts are possible with Oracle HTTP Server. This is because for name-based virtual hosts, the request must be read and inspected to determine which virtual host is used to process the request. If HTTPS is used, an SSL handshake must be performed before the request can be read. To perform the SSL handshake, a server certificate must be provided. To have a meaningful server certificate, the host name in the certificate must match the host name the client requested, which implies a unique server certificate per virtual host. However, because the server cannot know which virtual host to route the request to until it has read the request, and it can't properly read the request unless it knows which server certificate to provide, there is no way to make name-based virtual hosting work with HTTPS.

B.3 Can I Use Different Language and Character Set Versions of Document?

Yes, you can use multiviews, a general name given to the Apache HTTP Server's ability to provide language and character-specific document variants in response to a request.

See Also: Multiviews option in the Apache HTTP Server documentation on Content Negotiation, at:

http://httpd.apache.org/docs/current/content-negotiation.html

B.4 Can I Apply Apache HTTP Server Security Patches to Oracle HTTP Server?

No, you cannot apply the Apache HTTP Server security patches to Oracle HTTP Server for the following reasons:

- Oracle tests and appropriately modifies security patches before releasing them to Oracle HTTP Server users.
- In many cases, the Apache HTTP Server alerts, such as OpenSSL alerts, may not be applicable because Oracle has removed those components from the stack.

The latest security related fixes to Oracle HTTP Server are performed through the Oracle Critical Patch Update (CPU). For more details, refer to Oracle's Critical Patch Updates and Security Alerts Web page.

Note: After applying a CPU, the Apache HTTP Server-based version may stay the same, but the vulnerability will be fixed. There are third-party security detection tools that can check the version, but do not check the vulnerability itself.

B.5 Can I Upgrade the Apache HTTP Server Version of Oracle HTTP Server?

No, you cannot upgrade only the Apache HTTP Server version inside Oracle HTTP Server. Oracle provides a newer version of Apache HTTP Server that Oracle HTTP Server is based on, which is part of either a patch update or the next major or minor release of Oracle Fusion Middleware.

B.6 Can I Compress Output From Oracle HTTP Server?

In general, Oracle recommends using mod_deflate, which is included with Oracle HTTP Server. For more information pertaining to mod_deflate, see http://httpd.apache.org/docs/current/mod/mod_deflate.html

B.7 How Do I Create a Namespace That Works Through Firewalls and Clusters?

The general idea is that all servers in a distributed website should use a single URL namespace. Every server serves some part of that namespace, and is able to redirect or proxy requests for URLs that it does not serve to a server that is closer to that URL. For example, your namespaces could be the following:

```
/app1/login.html
/app1/catalog.html
/app1/dologin.jsp
/app2/orderForm.html
/apps/placeOrder.jsp
```

You could initially map these name spaces to two Web servers by putting app1 on server1 and app2 on server2. The configuration for server1 might look like the following:

```
Redirect permanent /app2 http://server2/app2
Alias /app1 /myApps/application1
<Directory /myApps/application1>
  ...
</Directory>
```

The configuration for Server2 is complementary.

If you decide to partition the namespace by content type (HTML on server1, and JSP on server2), then you can change server configuration and move files around, but you do not have to make changes to the application itself. The resulting configuration of server1 might look like the following:

```
RedirectMatch permanent (.*) \.jsp$ http://server2/$1.jsp
AliasMatch ^/app(.*) \.html$ /myPages/application$1.html
<DirectoryMatch "^/myPages/application\d">
  ...
</DirectoryMatch>
```

The amount of actual redirection can be minimized by configuring a hardware load balancer like F5 system BIG-IP to send requests to server1 or server2 based on the URL.

B.8 How Do I Protect the Website from Hackers?

There are many attacks by hackers, and new attacks are invented everyday. The following are some general guidelines for securing your site. You can never be completely secure, but you can avoid being an easy target.

- Use a commercial firewall, such as Checkpoint FW-1 or Cisco PIX between your ISP and your Web server. Remember not all hackers are outside your organization.
- Use switched Ethernet to limit the amount of traffic a compromised server can detect. Use additional firewalls between Web server machines and highly sensitive internal servers running the database and enterprise applications.
- Remove unnecessary network services such as RPC, Finger, and telnet from your server.
- Carefully validate all input from Web forms. Be especially wary of long input strings and input that contains non-printable characters, HTML tags, or javascript tags.
- Encrypt or randomize the contents of cookies that contain sensitive information to prevent a hacker from hijacking a valid session. For example, it should be difficult to guess a valid sessionID.
- Check often for security patches for all your system and application software, and install them as soon as possible. Be sure these patches come from reliable sources. Only download patches from trusted sites and verify the cryptographic checksum.
- Use an intrusion detection package to monitor for defaced Web pages, viruses, and presence of rootkits that indicate hackers have broken into your site. If possible, mount system executables and Web content on read-only file systems.
- Have a forensic analysis package on hand to capture evidence of a break in as soon as detected. This aids in prosecution of the hackers.
- Perform Pen testing or other relevant security testing on your application. Configure the appropriate custom mod_security rules to protect your application. For more information on mod_security, see Appendix F, "Configuring mod_security."
- If you want to use the open source Open Web Application Security Project (OWASP)-based core rule set, then you must ensure that the rule set corresponds to the version of mod_security included with Oracle HTTP Server. Note that the core rule set provides only a general set of rules and is not very application-specific. Thus, the core rule set might not adequately protect your application. For more information on the OWASP core rule set, see the following URL:

  `https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project`
- Remove unneeded content from the httpd.conf file. For more information, see Section 4.6.7, "Removing Access to Unneeded Content."

B.9 Should I Re-register Partner Applications with SSO Server If I Disable or Enable SSL?

Yes, if you enable or disable SSL, you have to re-register partner applications with the SSO server. When you make any changes that affect the URL (for example, changing the host name or port, or enabling or disabling SSL), you have to re-register partner applications with the SSO server because the old URL registered with the SSO server is no longer valid. You have to re-register the partner applications with the new URL.

B.10 Why is REDIRECT_ERROR_NOTES not set for "File Not Found" errors?

The REDIRECT_ERROR_NOTES CGI environment variable is not set for "File Not Found" errors in Oracle HTTP Server because compatibility with Apache HTTP Server does not make that information available to CGI and other applications for this condition.

B.11 How can I hide information about the Web Server Vendor and Version

Specify `ServerSignature Off` to remove this information from web server generated responses. Specify `ServerTokens Custom` *`some-server-string`* to disguise the web server software when Oracle HTTP Server generates the web Server response header. (When a backend server generates the response, the server response header may come from the backend server depending on the proxy mechanism.)

Note: `ServerTokens Custom some-server-string` is a replacement for the `ServerHeader Off` setting in Oracle HTTP Server 10g.

B.12 Can I Start OHS by Using apachectl or Other Command-Line Tool?

Oracle HTTP Server 12.1.2 process management is handled by Node Manager. The `startComponent` command can be used to start Oracle HTTP Server without using WLST or Fusion Middleware Control directly. For more information, see Section 4.3.2.3, "Starting Oracle HTTP Server Instances from the Command Line".

C

Troubleshooting Oracle HTTP Server

This appendix describes common problems that you might encounter when using Oracle HTTP Server (OHS), and explains how to solve them. It includes the following topics:

C.1 Oracle HTTP Server Unable to Start Due to Port Conflict

You can get the following error if Oracle HTTP Server cannot start due to port conflict:

```
[VirtualHost: main] (98)Address already in use: make_sock: could not bind to
address [::]:7777
```

Solution

Determine what process is already using that port, and then either change the IP:port address of Oracle HTTP Server or the port of the conflicting process.

C.2 System Overloaded by Number of httpd Processes

When too many httpd processes run on a system, the response time degrades because there are insufficient resources for normal processing.

Solution

Lower the value of `MaxClients` to a value the machine can accommodate.

C.3 Permission Denied When Starting Oracle HTTP Server On a Port Below 1024

You will get the following error if you try to start Oracle HTTP Server on a port below 1024:

```
[VirtualHost: main] (13)Permission denied: make_sock: could not bind to address
[::]:443
```

Oracle HTTP Server will not start on ports below 1024 because root privileges are needed to bind these ports.

Solution

Follow the steps in Section 4.3.2.4, "Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only)" to start Oracle HTTP Server on a Privileged Port.

C.4 Oracle HTTP Server May Fail To Start If PM Files Are Not Located Correctly

If Oracle HTTP Server is not able to locate Perl module (PM) files in the path defined in the `PERL5LIB` variable, Oracle HTTP Server may encounter the following errors, and fail to start:

```
[error] Can't locate mod_perl.pm in @INC (@INC contains:$ORACLE_HOME/perl/...)
```

or:

```
[error] Can't locate Apache::Registry.pm in @INC (@INC contains: $ORACLE_
HOME/perl/...)
```

Solution

Check that *ORACLE_HOME*/ohs/bin/apachectl is correctly defined in the PERL5LIB variable. It should point to the path(s) containing the PM files. By default, it points to PM files in the following directories:

```
ORACLE_HOME/ohs/mod_perl/lib/site_perl/5.10.0
ORACLE_HOME/perl/lib/5.10.0
ORACLE_HOME/perl/lib/site_perl/5.10.0
```

C.5 Exception Thrown when Unsetting PerSetEnv and Removing Variable

If you configure mod_perl by using the EM mod_perl configuration page and try to remove a previously configured `PerSetEnv` variable from the Environment field, this error is thrown:

```
Failed to invoke operation save on MBean
oracle.as.management.mbeans.register:type=component,name=ohs1,instance=webtier

_inst7971,Location=AdminServer
Apply failed, modify required parameters and save again. Validation of
configuration trying to apply failed
.
.
.
```

Solution

To correct this situation:

1. Close the pop-up error and click **Revert**.
2. Remove the `PerSetEnv` by doing one of the following:
   - Go to the Advanced Configuration page of Fusion Middleware Control and modify the mod_perl.conf file directly.

     OR

   - Go to the *DOMAIN_HOME*/config/fmwconfig/components/OHS/*componentName*/moduleconf/mod_perl.conf and edit the configuration file directly to remove the `PerSetEnv` value.
3. Restart OHS.

C.6 Using Log Files to Locate Errors

You can use the following log files to help locate errors:

- Rewrite Log
- Script Log
- Error Log

C.6.1 Rewrite Log

This log file is necessary for debugging when mod_rewrite is used. The log file produces a detailed analysis of how the rewriting engine transforms requests. The level of detail is controlled by the `RewriteLogLevel` directive.

C.6.2 Script Log

This log file enables you to record the input to and output from the CGI scripts. This should only be used in testing, and not for production servers.

> **See Also:** ScriptLog in the Apache HTTP Server documentation at:
>
> `http://httpd.apache.org/docs/current/mod/mod_cgi.html#scriptlog`

C.6.3 Error Log

This log file records overall server problems. Refer to Chapter 7, "Managing Oracle HTTP Server Logs" for details on configuring and viewing error logs.

C.7 Recovering an OHS Instance on a Remote Host

If you need to recover an Oracle HTTP Server instance that is installed on a remote host (that is, a host with just managed servers but no Administration Server), you must use `tar` and `untar`; `pack.sh` and `unpack.sh` do not work in this scenario.

C.8 Oracle HTTP Server Performance Issues

The following are performance issues, along with their solutions, that you might encounter when running Oracle HTTP Server:

- Special Runtime Files Reside on a Network File System
- UNIX Sockets on a Network File System
- DocumentRoot on a Slow File System

C.8.1 Special Runtime Files Reside on a Network File System

Oracle HTTP Server uses locks for its internal processing, which in turn use lock files. These files are created dynamically when the lock is created and are accessed every time the lock is taken or released. If these files reside on a slower file system (for example, network file system), then there could be severe performance degradation. To counter this issue:

- On Linux
  1. In httpd.conf, change `AcceptMutex fcntl` to `AcceptMutex sysvsem` (two places).
  2. In httpd.conf, comment-out the `LockFile` directive (three places).
- On Solaris
  1. In httpd.conf, change `AcceptMutex fcntl` to `AcceptMutex pthread` (two places).
  2. In httpd.conf, comment-out the `LockFile` directive (three places).
- Other UNIX Platforms

  In httpd.conf, change the `LockFile` directive to point to a local filesystem (three places).

C.8.2 UNIX Sockets on a Network File System

mod_cgid and mod_fastcgi are not enabled by default. If enabled, these modules use UNIX sockets internally. If UNIX sockets reside on a slower file system (e.g., network file system), a severe performance degradation could be observed. You can set the following directives to avoid the issue:

- If mod_cgid is enabled, use the `ScriptSock` directive to place mod_cgid's UNIX socket on a local filesystem.
- If mod_fastcgi is enabled, use the `FastCgiIpcDir` directive to place mod_fastcgi's UNIX sockets on a local filesystem.

C.8.3 DocumentRoot on a Slow File System

If you are using mod_wl_ohs to route the requests to back-end WLS server/cluster, and the DocumentRoot is on a slower file system (e.g., network file system), then every request that is routed to the backend server can experience performance issues. This can be overcome by setting `WLSRequest` to `ON` instead of `SetHandler weblogic-handler`.

C.9 Out of DMS Shared Memory

In some extreme configurations, you might see the following message in the OHS error log:

```
dms_fail_shm_expansion: out of DMS shared memory in pid XXX, disabling DMS;
increase DMSThreadSharedMem directive from YYY
```

This is because of an incorrect calculation of required shared memory for OHS DMS. This can be resolved by setting `DMSThreadSharedMem` to a larger value than the default of 350. Continue setting `DMSThreadSharedMem` 50% higher until the problem is resolved.

In a configuration with a very large number of virtual hosts (hundreds or thousands), if the above workaround does not work, you can instead set the environment variable OHS_DMS_BLOCKSIZE to the desired value.

D

Configuration Files

The default Oracle HTTP Server configuration contains the files described in the following sections:

- Section D.1, "httpd.conf"
- Section D.2, "ssl.conf"
- Section D.3, "admin.conf"
- Section D.4, "mod_wl_ohs.conf"
- Section D.5, "moduleconf/*.conf"
- Section D.6, "disabled/*.conf"
- Section D.7, "mime.types"
- Section D.8, "ohs.plugins.nodemanager.properties"
- Section D.9, "magic"
- Section D.10, "keystores/<wallet-directory>"
- Section D.11, "auditconfig.xml"
- Section D.12, "component-logs.xml"
- Section D.13, "component_events.xml"
- Section D.14, "Additional Reference"

For more information about the configuration files, see Section 1.6, "Understanding Configuration Files"

D.1 httpd.conf

Description	Top-level web server configuration file
Format	Apache HTTP Server .conf file format
Primary feature configured	Various, including non-SSL listening socket

D.2 ssl.conf

Description	Web server configuration file for SSL
Format	Apache HTTP Server .conf file format
Primary feature configured	mod_ossl

D.3 admin.conf

Description	Web server configuration file for administration port
Format	Apache HTTP Server .conf file format
Primary feature configured	mod_dms; administration port used for communication with node manager

Note: Only the listen port and local address are intended for customer configuration.

D.4 mod_wl_ohs.conf

Description	Web server configuration file for WebLogic plug-in
Format	Apache HTTP Server .conf file format
Primary feature configured	WebLogic plug-in (mod_wl_ohs)

D.5 moduleconf/*.conf

Description	Optional, enabled web server configuration files for specific features, such as mod_plsql
Format	Apache HTTP Server .conf file format
Primary feature configured	default: mod_plsql

Note: To disable .conf move it from moduleconf/ to disabled/.

D.6 disabled/*.conf

Description	Optional, disabled web server configuration files for specific features, such as mod_plsql
Format	Apache HTTP Server .conf file format
Primary feature configured	default: mod_perl, mod_fastcgi (if .conf file is moved to moduleconf/)

Note: To enable a .conf file in the disabled directory, move it from moduleconf/ to disabled/.

D.7 mime.types

Description	Web server configuration file for mod_mime
Format	mod_mime file format
Primary feature configured	Mime types used by mod_mime

D.8 ohs.plugins.nodemanager.properties

Description	Configuration file for Oracle HTTP Server node manager plug-ins
Format	Java property file format
Primary feature configured	Oracle HTTP Server Node Manager plug-ins

D.9 magic

Description	Optional, disabled web server configuration file for mod_mime_magic
Format	mod_mime_magic file format
Primary feature configured	File content patterns used by mod_mime_magic

D.10 keystores/<*wallet-directory*>

Name example: keystores/default

Description	Oracle wallet
Format	Oracle wallet format
Primary feature configured	Oracle wallets for SSL/TLS communication

D.11 auditconfig.xml

Description	Configuration of OHS auditing and logging
Format	FMW audit framework audit configuration XML format
Primary feature configured	FMW audit framework auditing of Oracle HTTP Server operations

D.12 component-logs.xml

Description	Configuration of OHS log files for log collection
Format	FMW log file configuration XML format
Primary feature configured	Log collection

D.13 component_events.xml

Description	Static configuration of OHS audit event definitions
Format	FMW audit framework component event XML format
Primary feature configured	FMW audit framework

Note: This configuration file is not intended for modification by customers.

D.14 Additional Reference

For additional information, see the following documentation:

- Apache HTTP Server .conf file format:

  http://httpd.apache.org/docs/2.2/configuring.html

- mod_mime file format:

  http://httpd.apache.org/docs/2.2/mod/mod_mime.html

- mod_mime_magic file format:

  http://httpd.apache.org/docs/2.2/mod/mod_mime_magic.html

E

Property Files

This appendix documents the property files used by Oracle HTTP Server. The files include:

- Section E.1, "ohs_admin.properties"
- Section E.2, "ohs_nm.properties"
- Section E.3, "ohs.plugins.nodemanager.properties"

E.1 ohs_admin.properties

The ohs_admin.properties file is a per domain file used to configure the Oracle HTTP Server administration server MBeans. This file must be edited manually and the administration server restarted for the change to take effect.

File path: *DOMAIN_HOME*/config/fmwconfig/components/OHS/ohs_admin.properties

Editable properties in this file are listed here:

Property	Description
LogLevel	The log level for the OHS Node Manager plug-in. Accepted Values: ■ SEVERE (highest value) ■ WARNING ■ INFO ■ CONFIG ■ FINE ■ FINER ■ FINEST (lowest value) Default: INFO

E.2 ohs_nm.properties

The ohs_nm.properties file is a per domain file used to configure the Oracle HTTP Server Node Manager plug-in. This file must be edited manually and Node Manager restarted for the change to take effect.

File path: *DOMAIN_HOME*/config/fmwconfig/components/OHS/ohs_nm.properties

Property	Description
LogLevel	The log level for the OHS undemanding plug-in. Accepted values: ■ SEVERE (highest value) ■ WARNING ■ INFO ■ CONFIG ■ FINE ■ FINER ■ FINEST (lowest value) Default: INFO

E.3 ohs.plugins.nodemanager.properties

The ohs.plugins.nodemanager.properties file exists for each configured Oracle HTTP Server and contains configured parameters OHS process management. This file must be manually edited and propagated to the run-time area.

File path: *DOMAIN_HOME*/config/fmwconfig/components/OHS/ohs1/ohs.plugins.nodemanager.properties

E.3.1 Cross-platform Properties

The following table lists the cross-platform properties:

Property	Description
`config-file`	The base filename of the initial Oracle HTTP Server configuration file. `config-file` accepts any valid .conf file in the instance configuration directory. **Caution:** The specified .conf file must include admin.conf in the same manner as the default httpd.conf. Default: httpd.conf
`command-line`	Extra arguments to add to the httpd invocation. `command-line` accepts any valid httpd command-line parameters. **Caution:** These must not conflict with the usual start, stop, and restart parameters. Using -D and symbol is the expected use of this property. Default: None
`start-timeout`	The maximum number of seconds to wait for Oracle HTTP Server to start and initialize. `start-timeout` accepts any numeric value from 5 to 3600. Default: 120
`stop-timeout`	The maximum number of seconds to wait for the Oracle HTTP Server to terminate. `stop-timeout` accepts any numeric value from 5 to 3600. Default: 60

Property	Description
`restart-timeout`	The maximum number of seconds to wait for the Oracle HTTP Server to restart. `restart-timeout` accepts any numeric value from 5 to 3600. Default: 180
`ping-interval`	The number of seconds from the completion of one Node Manager health check ping to the Oracle HTTP Server until the start of the next. A value of 0 disables pings. `ping-interval` accepts any numeric value from 0 to 3600. Default: 30
`ping-timeout`	The maximum number of seconds to wait for an Oracle HTTP Server health check ping to complete. `ping-tmeout` accepts any numeric value from 5 to 3600. Default: 60

Example:

```
config-file = httpd.conf
command-line = -DSYMBOL
start-timeout = 120
stop-timeout = 60
restart-timeout = 180
ping-interval = 30
ping-timeout = 60
```

E.3.2 Environment Variable Configuration Properties

Additional environment variables for the OHS server may be specified using environment properties.

The environment property syntax is:

`environment[.append][.<order>].<name> = <value>`

Where:

- The optional `.append` will append the new <value> to any existing value for <*name*>. If <*name*> has not yet been defined, then <value> will be the new value.
- The optional `.<order>` value sets order for this definition's setting in the environment (the default is 0). The order determines when the configured variable is added to the process' environment (and its value evaluated). Environment properties with lower order values are processed before those with higher order values. The order value must be an integer with a value greater than or equal to 0.
- <*name*> is the environment variable name, which must begin with a letter or underscore, and consist of letters, numeric digits or underscores.
- <*value*> is the value of environment variable <*name*>. The value can reference other environment variable names, including its own.

  The following special references may be included in the value:

  – "$:" for the path separator

  – "$/" for the file separator

  – "$$" for '$'

With the exception of these special characters, UNIX variable syntax references ("$*name*" or "${*name*}") and the Windows variable syntax reference ("%*name*%") are supported.

Note that each property name within the same property file must be unique (the behavior is not defined for multiple properties defined with the same name), thus the .<*order*> field should be used to keep property names unique when multiple definitions are provided for the same environment variable <*name*>.

The following environment variables are set by the Oracle HTTP Server Node Manager plug-in:

- SHELL: From Node Manager's environment, or defaults to /bin/sh, or cmd.exe for Windows
- ORA_NLS33: Set to $ORACLE_HOME/nls/data
- NLS_LANG: From Node Manager's environment, otherwise default
- LANG: From Node Manager's environment, otherwise default
- LC_ALL: From Node Manager's environment, if set
- TZ: From Node Manager's environment, if set
- ORACLE_HOME: Full path to the Oracle home
- ORACLE_INSTANCE: Full path to the domain home
- INSTANCE_NAME: The name of the domain
- PRODUCT_HOME: The path to the OHS install: $ORACLE_HOME/ohs
- PATH: Defaults to
  - On UNIX:

    $*PRODUCT_HOME*/bin:$*ORACLE_HOME*/bin:

    $*ORACLE_HOME*/jdk/bin:/bin:/usr/bin:/usr/local/bin
  - On Windows:

    %*PRODUCT_HOME*%\bin;%*ORACLE_HOME*%\bin;

    %*ORACLE_HOME*%\jdk\bin;%SystemRoot%;%SystemRoot%\system32

These variables apply to UNIX only:

- TNS_ADMIN: From Node Manager's environment, or $ORACLE_ HOME/network/admin
- LD_LIBRARY_PATH: $PRODUCT_HOME/lib:$ORACLE_ HOME/lib:$ORACLE_HOME/jdk/lib
- LIBPATH: Same as LD_LIBARY_PATH
- X_LD_LIBRARY_PATH_64: Same as LD_LIBRARY_PATH

These variables apply to Windows only:

- ComSpec: Defaults to %ComSpec% value from the system.
- SystemRoot: Defaults to %SystemRoot% value from the system.
- SystemDrive: Defaults to %SystemDrive% value from the system.

Example

On a UNIX like system with the web tier installed as /oracle and the environment variable "MODX_RUNTIME=special" set in the NodeManager's environment, the following definitions:

```
environment.MODX_RUNTIME = $MODX_RUNTIME
environment.1.MODX_ENV = Value A
environment.1.MODX_PATH = $PATH$:/opt/modx/bin
environment.2.MODX_ENV = ${MODX_ENV}, Value B
environment.append.2.MODX_PATH = /var/modx/bin
MODX_ENV = Value A, Value B
MODX_PATH = /oracle/ohs/bin:/oracle/bin:/oracle/jdk/bin:/bin:/usr/bin:
/usr/local/bin:/opt/modx/bin:/var/modx/bin
```

would result in the following additional environment variables set for Oracle HTTP Server:

```
MODX_RUNTIME = special
```

E.3.3 Properties Specific to Oracle HTTP Server Instances Running on Linux and UNIX

These should only be configured for instances running on Linux or other UNIX like systems.

Property	Description
`restart-mode`	Determines whether to use graceful or hard restart for the Oracle HTTP Server when configuration changes are activated. `restart-mode` accepts these values: ■ restart ■ graceful Default: graceful
`stop-mode`	Determines whether to use a graceful or hard stop when stopping Oracle HTTP Server. `stop-mode` accepts these values: ■ stop ■ graceful-stop Default: stop
`mpm`	Determines whether to use the prefork or worker MPM for Oracle HTTP Server. `mpm` accepts these values: ■ prefork ■ worker Default: worker
allow-`corefiles`	Determines whether ulimit should be set to allow core files to be written for OHS server crashes. `allow-corefiles` accepts these values: ■ yes ■ no Default: no

Example

```
restart-mode = graceful
stop-mode = stop
mpm = worker
allow-corefiles = no
```

F

Configuring mod_security

mod_security is an open-source module that you can use to detect and prevent intrusion attacks against Oracle HTTP Server; for example, you can specify a mod_security rule to screen all incoming requests and deny requests that match the conditions specified in the rule. The mod_security module (version 2.7.2) and its prerequisites are included in the Oracle HTTP Server installation as a shared object named mod_security2.so in the ORACLE_HOME/ohs/modules directory.

This version of Oracle HTTP Server supports only mod_security (version 2.7.2) directives, variables, action, phases and functions. *It will not be supported if you replace this module with a later version.*

This appendix contains a usable example (Example F–1) of the mod_security.conf file, including the `LoadModule` statement.

Notes: Be aware of the following:

- mod_security was removed from earlier versions of Oracle HTTP Server but was reintroduced in version 11.1.1.7. This version follows the recommendations and practices prescribed for open source mod_security 2.7.2. Only documentation applicable to open source mod_security 2.7.2 is applicable to the Oracle HTTP Server implementation of the module.
- In Oracle HTTP Server 11.1.1.7 and later, mod_security is not loaded or configured by default; however, if you have an installation patched from 11.1.1.6, implementing the patch might have already loaded and configured the module.
- Oracle only supports the Oracle-supplied version of mod_security. Newer versions from modsecurity.org will not be supported.

For more information on mod_security, see the mod_security documentation site, at:

`http://www.modsecurity.org/documentation/`

This chapter contains the following sections:

F.1 Enabling mod_security

To make the mod_security module available for use when Oracle HTTP Server is running, ensure that mod_security.conf begins with the following lines:

```
#Load module
LoadModule security2_module "${PRODUCT_HOME}/modules/mod_security2.so"
```

as shown in the mod_security.conf example inExample F–1.

F.2 Configuring mod_security

Configuring mod_security involves specifying certain directives in the Oracle HTTP Server configuration file. You can specify the directives directly in the httpd.conf file in an `IfModule` container. Alternatively, you can specify the mod_security directives in a separate mod_security.conf file and include that .conf file in httpd.conf by using the `Include` directive.

By default, mod_security.conf does not exist, so you need to create it, preferably by using the template in Example F–1. Copy and paste the sample into a text editor and read the entire file, editing it for your system. Then save it as your own mod_security.conf and include it from your httpd.conf. If you implement mod_security.conf as described in this appendix, it will use the `LoadModule` directive to load mod_security2.so into the run time environment.

Note: Oracle strongly recommends that you change the value of the /tmp/ directory to a location where users do not have access.

Example F–1 mod_security.conf Sample

```
#Load module
LoadModule security2_module "${PRODUCT_HOME}/modules/mod_security2.so"
# -- Rule engine initialization ----------------------------------------------

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine DetectionOnly


# -- Request body handling ---------------------------------------------------

# Allow ModSecurity to access request bodies. If you don't, ModSecurity
# won't be able to see any POST parameters, which opens a large security
# hole for attackers to exploit.
#
SecRequestBodyAccess On

# Enable XML request body parser.
# Initiate XML Processor in case of xml content-type
#
SecRule REQUEST_HEADERS:Content-Type "text/xml"
"id:'200000',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=XML"


# Maximum request body size we will accept for buffering. If you support
# file uploads then the value given on the first line has to be as large
# as the largest file you are willing to accept. The second value refers
# to the size of data, with files excluded. You want to keep that value as
# low as practical.
#
SecRequestBodyLimit 13107200
```

```
SecRequestBodyNoFilesLimit 131072

# Store up to 128 KB of request body data in memory. When the multipart
# parser reachers this limit, it will start using your hard disk for
# storage. That is slow, but unavoidable.
#
SecRequestBodyInMemoryLimit 131072

# What do do if the request body size is above our configured limit.
# Keep in mind that this setting will automatically be set to ProcessPartial
# when SecRuleEngine is set to DetectionOnly mode in order to minimize
# disruptions when initially deploying ModSecurity.
#
SecRequestBodyLimitAction Reject

# Verify that we've correctly processed the request body.
# As a rule of thumb, when failing to process a request body
# you should reject the request (when deployed in blocking mode)
# or log a high-severity alert (when deployed in detection-only mode).
#
SecRule REQBODY_ERROR "!@eq 0" \
"id:'200001', phase:2,t:none,log,deny,status:400,msg:'Failed to parse request
 body.',logdata:'%{reqbody_error_msg}',severity:2"

# By default be strict with what we accept in the multipart/form-data
# request body. If the rule below proves to be too strict for your
# environment consider changing it to detection-only. You are encouraged
# _not_ to remove it altogether.
#
SecRule MULTIPART_STRICT_ERROR "!@eq 0" \
"id:'200002',phase:2,t:none,log,deny,status:44, \
msg:'Multipart request body failed strict validation: \
PE %{REQBODY_PROCESSOR_ERROR}, \
BQ %{MULTIPART_BOUNDARY_QUOTED}, \
BW %{MULTIPART_BOUNDARY_WHITESPACE}, \
DB %{MULTIPART_DATA_BEFORE}, \
DA %{MULTIPART_DATA_AFTER}, \
HF %{MULTIPART_HEADER_FOLDING}, \
LF %{MULTIPART_LF_LINE}, \
SM %{MULTIPART_MISSING_SEMICOLON}, \
IQ %{MULTIPART_INVALID_QUOTING}, \
IP %{MULTIPART_INVALID_PART}, \
IH %{MULTIPART_INVALID_HEADER_FOLDING}, \
FL %{MULTIPART_FILE_LIMIT_EXCEEDED}'"

# Did we see anything that might be a boundary?
#
SecRule MULTIPART_UNMATCHED_BOUNDARY "!@eq 0" \
"id:'200003',phase:2,t:none,log,deny,status:44,msg:'Multipart parser detected a possible unmatched
boundary.'"

# PCRE Tuning
# We want to avoid a potential RegEx DoS condition
#
SecPcreMatchLimit 1000
SecPcreMatchLimitRecursion 1000

# Some internal errors will set flags in TX and we will need to look for these.
# All of these are prefixed with "MSC_".  The following flags currently exist:
#
```

```
# MSC_PCRE_LIMITS_EXCEEDED: PCRE match limits were exceeded.
#
SecRule TX:/^MSC_/ "!@streq 0" \
        "id:'200004',phase:2,t:none,deny,msg:'ModSecurity internal error flagged: %{MATCHED_VAR_
NAME}'"

# -- Response body handling ---------------------------------------------

# Allow ModSecurity to access response bodies.
# You should have this directive enabled in order to identify errors
# and data leakage issues.
#
# Do keep in mind that enabling this directive does increases both
# memory consumption and response latency.
#
SecResponseBodyAccess On

# Which response MIME types do you want to inspect? You should adjust the
# configuration below to catch documents but avoid static files
# (e.g., images and archives).
#
SecResponseBodyMimeType text/plain text/html text/xml

# Buffer response bodies of up to 512 KB in length.
SecResponseBodyLimit 524288

# What happens when we encounter a response body larger than the configured
# limit? By default, we process what we have and let the rest through.
# That's somewhat less secure, but does not break any legitimate pages.
#
SecResponseBodyLimitAction ProcessPartial

# -- Filesystem configuration -------------------------------------------

# The location where ModSecurity stores temporary files (for example, when
# it needs to handle a file upload that is larger than the configured limit).
#
# This default setting is chosen due to all systems have /tmp available however,
# this is less than ideal. It is recommended that you specify a location that's private.
#
SecTmpDir /tmp/

# The location where ModSecurity will keep its persistent data.  This default setting
# is chosen due to all systems have /tmp available however, it
# too should be updated to a place that other users can't access.
#
SecDataDir /tmp/

# -- File uploads handling configuration --------------------------------

# The location where ModSecurity stores intercepted uploaded files. This
# location must be private to ModSecurity. You don't want other users on
# the server to access the files, do you?
#
#SecUploadDir /opt/modsecurity/var/upload/

# By default, only keep the files that were determined to be unusual
# in some way (by an external inspection script). For this to work you
# will also need at least one file inspection rule.
#
```

```
#SecUploadKeepFiles RelevantOnly

# Uploaded files are by default created with permissions that do not allow
# any other user to access them. You may need to relax that if you want to
# interface ModSecurity to an external program (e.g., an anti-virus).
#
#SecUploadFileMode 0600


# -- Debug log configuration ---------------------------------------------

# The default debug log configuration is to duplicate the error, warning
# and notice messages from the error log.
#
#SecDebugLog /opt/modsecurity/var/log/debug.log
#SecDebugLogLevel 3


# -- Audit log configuration ---------------------------------------------

# Log the transactions that are marked by a rule, as well as those that
# trigger a server error (determined by a 5xx or 4xx, excluding 404,
# level response status codes).
#
SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?!04))"

# Log everything we know about a transaction.
SecAuditLogParts ABIJDEFHZ

# Use a single file for logging. This is much easier to look at, but
# assumes that you will use the audit log only ocassionally.
#
SecAuditLogType Serial
SecAuditLog "${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/modsec_audit.log"

# Specify the path for concurrent audit logging.
SecAuditLogStorageDir "${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs"
#Simple test
SecRule ARGS "\.\./" "t:normalisePathWin,id:99999,severity:4,msg:'Drive Access'"

# -- Miscellaneous -------------------------------------------------------

# Use the most commonly used application/x-www-form-urlencoded parameter
# separator. There's probably only one application somewhere that uses
# something else so don't expect to change this value.
#
SecArgumentSeparator &

# Settle on version 0 (zero) cookies, as that is what most applications
# use. Using an incorrect cookie version may open your installation to
# evasion attacks (against the rules that examine named cookies).
#
SecCookieFormat 0

# Specify your Unicode Code Point.
# This mapping is used by the t:urlDecodeUni transformation function
# to properly map encoded data to your language. Properly setting
# these directives helps to reduce false positives and negatives.
#
```

```
#SecUnicodeCodePage 20127
#SecUnicodeMapFile unicode.mapping
```

G

OHS Module Directives

This appendix describes the directives available in the Oracle-developed modules supported by OHS. It contains these sections:

- Section G.1, "mod_certheaders"
- Section G.2, "mod_ossl"
- Section G.3, "mod_plsql"

G.1 mod_certheaders

mod_certheaders accepts the following directives:

- AddCertHeader
- SimulateHttps

G.1.1 AddCertHeader

Specify which headers should be translated to CGI environment variables. This can be achieved by using the `AddCertHeader` directive. This directive takes a single argument, which is the CGI environment variable that should be populated from a HTTP header on incoming requests. For example, to populate the SSL_CLIENT_CERT CGI environment variable.

Category	Value
Syntax	`AddCertHeader` *`environment_variable`*
Example	`AddCertHeader SSL_CLIENT_CERT`
Default	None

G.1.2 SimulateHttps

mod_certheaders can be used to instruct Oracle HTTP Server to treat certain requests as if they were received through HTTPS even though they were received through HTTP. This is useful when Oracle HTTP Server is front-ended by a reverse proxy or load balancer, which acts as a termination point for SSL requests, and forwards the requests to Oracle HTTP Server through HTTPS.

Category	Value
Syntax	`SimulateHttps on|off`
Example	`SimulateHttps on`

Category	Value
Default	`off`

G.2 mod_ossl

To configure SSL for your Oracle HTTP Server, enter the `mod_ossl` directives you want to use in the `ssl.conf` file.

The following directives are described in subsequent sections:

- SSLAccelerator
- SSLCARevocationFile
- SSLCARevocationPath
- SSLCipherSuite
- SSLEngine
- SSLFIPS
- SSLInsecureRenegotiation
- SSLMutex
- SSLNZTraceLogLevel
- SSLOptions
- SSLPassPhraseDialog
- SSLProtocol
- SSLProxyCipherSuite
- SSLProxyEngine
- SSLProxyProtocol
- SSLProxyWallet
- SSLRequire
- SSLRequireSSL
- SSLSessionCache
- SSLSessionCacheTimeout
- SSLVerifyClient
- SSLWallet

G.2.1 SSLAccelerator

Specifies if SSL accelerator is used. Currently only nFast card is supported.

Category	Value
Syntax	`SSLAccelerator yes|no`
Example	SSLAccelerator yes
Default	`SSLAccelerator no`

Note: The `SSLAccelerator` directive has been deprecated. For information on enabling SSL acceleration support using a wallet, refer to the *Oracle Advanced Security Administrator's Guide* on `http://www.oracle.com/technology/documentation`.

G.2.2 SSLCARevocationFile

Specifies the file where you can assemble the Certificate Revocation Lists (CRLs) from CAs (Certificate Authorities) that you accept certificates from. These are used for client authentication. Such a file is the concatenation of various PEM-encoded CRL files in order of preference. This directive can be used alternatively or additionally to `SSLCARevocationPath`.

Category	Value
Syntax	`SSLCARevocationFile` *file_name*
Example	`SSLCARevocationFile ${`*ORACLE_INSTANCE*`}/config/fmwconfig/components/${`*COMPONENT_TYPE*`}/instances/${COMPONENT_NAME}/keystores/crl/ca_bundle.cr`
Default	None

G.2.3 SSLCARevocationPath

Specifies the directory where PEM-encoded Certificate Revocation Lists (CRLs) are stored. These CRLs come from the CAs (Certificate Authorities) that you accept certificates from. If a client attempts to authenticate itself with a certificate that is on one of these CRLs, then the certificate is revoked and the client cannot authenticate itself with your server.

Category	Value
Syntax	`SSLCARevocationPath` *path/to/CRL_directory/*
Example	`SSLCARevocationPath ${`*ORACLE_INSTANCE*`}/config/fmwconfig/components/${`*COMPONENT_TYPE*`}/instances/${COMPONENT_NAME}/keystores/crl`
Default	None

G.2.4 SSLCipherSuite

Specifies the SSL cipher suite that the client can use during the SSL handshake. This directive uses a colon-separated cipher specification string to identify the cipher suite. Table 11–2 shows the tags you can use in the string to describe the cipher suite you want. SSLCipherSuite accepts the following values:

- none: Adds the cipher to the list
- + : Adds the cipher to the list and place it in the correct location in the list
- - : Remove the cipher from the list (can be added later)
- ! : Remove the cipher from the list permanently

Tags are joined with prefixes to form a cipher specification string. Cipher suite tags are listed in Table G–1.

Category	Value
Example	`SSLCipherSuite` ALL:!MD5 In this example, all ciphers are specified except MD5 strength ciphers.
Syntax	`SSLCipherSuite` *`cipher-spec`*
Default	`ALL:!ADH:+HIGH:+MEDIUM:+LOW`

Table G–1 SSLCipher Suite Tags

Function	Tag	Meaning
Key exchange	`kRSA`	`RSA` key exchange
Key exchange	`kDHr`	Diffie-Hellman key exchange with RSA key
Authentication	`aNULL`	No authentication
Authentication	`aRSA`	`RSA` authentication
Authentication	`aDH`	Diffie-Hellman authentication
Encryption	`eNULL`	No encryption
Encryption	`DES`	`DES` encoding
Encryption	`3DES`	Triple `DES` encoding
Encryption	`RC4`	`RC4` encoding
Encryption	`ECC`	Elliptic curve cryptography encoding
Data Integrity	`MD5`	`MD5` hash function
Data Integrity	`SHA`	`SHA` hash function
Data Integrity	`SHA256`	SHA256 hash function
Data Integrity	`SHA384`	SHA384 hash function
Aliases	`SSLv3`	All SSL version 3.0 ciphers
Aliases	`TLSv1`	All TLS version 1 ciphers
Aliases	`TLSv1.1`	All TLS version 1.1 ciphers
Aliases	`TLSv1.2`	All TLS version 1.2 ciphers
Aliases	`LOW`	All low strength ciphers (export and single `DES`)
Aliases	`MEDIUM`	All ciphers with 128-bit encryption
Aliases	`HIGH`	All ciphers using triple `DES`
Aliases	`AES`	All ciphers using AES encryption.
Aliases	`RSA`	All ciphers using `RSA` key exchange
Aliases	`DH`	All ciphers using Diffie-Hellman key exchange

Table G–2 lists the Cipher Suites supported in Oracle Advanced Security 12.1.2.

Table G–2 Cipher Suites Supported in Oracle Advanced Security 12.1.2

Cipher Suite	Authentication	Encryption	Data Integrity	TLSv1	TLSv1.1	TLSv1.2
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4 (128)	MD5	Yes	Yes	Yes
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4 (128)	SHA	Yes	Yes	Yes
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES (168)	SHA	Yes	Yes	Yes
SSL_RSA_WITH_AES_128_CBC_SHA	RSA	AES (128)	SHA	Yes	Yes	Yes
SSL_RSA_WITH_AES_256_CBC_SHA	RSA	AES (256)	SHA	Yes	Yes	Yes
RSA_WITH_AES_128_CBC_SHA256	RSA	AES (128)	SHA256	No	No	Yes
RSA_WITH_AES_256_CBC_SHA256	RSA	AES (256)	SHA256	No	No	Yes
RSA_WITH_AES_128_GCM_SHA256	RSA	AES (128)	SHA256	No	No	Yes
RSA_WITH_AES_256_GCM_SHA384	RSA	AES (256)	SHA384	No	No	Yes
ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDSA	AES (128)	SHA	Yes	Yes	Yes
ECDHE_ECDSA_WITH_AES_256_CBC_SHA	ECDSA	AES (256)	SHA	Yes	Yes	Yes
ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDSA	AES (128)	SHA256	No	No	Yes
ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDSA	AES (256)	SHA384	No	No	Yes
ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDSA	AES (128)	SHA256	No	No	Yes
ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDSA	AES (256)	SHA384	No	No	Yes

G.2.5 SSLEngine

Toggles the usage of the SSL Protocol Engine. This is usually used inside a `<VirtualHost>` section to enable SSL for a particular virtual host. By default, the SSL Protocol Engine is disabled for both the main server and all configured virtual hosts. Example 11–1 is an example for using SSLEngine directive.

Category	Value
Syntax	`SSLEngine on|off`
Example	`SSLEngine on`
Default	`off`

G.2.6 SSLFIPS

This directive toggles the usage of the SSL library FIPS_mode flag. It must be set in the global server context and cannot be configured with conflicting settings (`SSLFIPS on` followed by `SSLFIPS off` or similar). The mode applies to all SSL library operations.

Note: FIPS is available only on the UNIX/Linux platform. It is not available on the Windows platform

Category	Value
Syntax	`SSLFIPS ON | OFF`
Example	`SSLFIPS ON`
Default	Off

Configuring an SSLFIPS change requires that the `SSLFIPS on/off` directive be set globally in ssl.conf. Virtual level configuration is disabled in SSLFIPS directive. Hence, setting SSLFIPS to virtual directive will result in an error.

The cipher suites supported the SSLFIPS mode are:

- `SSL_RSA_WITH_3DES_EDE_CBC_SHA`
- `SSL_RSA_WITH_AES_128_CBC_SHA`
- `SSL_RSA_WITH_AES_256_CBC_SHA`
- `RSA_WITH_AES_128_CBC_SHA256`
- `RSA_WITH_AES_256_CBC_SHA256`
- `RSA_WITH_AES_256_GCM_SHA384`
- `ECDHE_ECDSA_WITH_AES_128_CBC_SHA`
- `ECDHE_ECDSA_WITH_AES_256_CBC_SHA`
- `ECDHE_ECDSA_WITH_AES_128_CBC_SHA256`
- `ECDHE_ECDSA_WITH_AES_256_CBC_SHA384`
- `ECDHE_ECDSA_WITH_AES_128_GCM_SHA256`
- `ECDHE_ECDSA_WITH_AES_256_GCM_SHA384`

For instructions on how to implement these cipher suites, see Section G.2.4, "SSLCipherSuite".

G.2.7 SSLInsecureRenegotiation

As originally specified, all versions of the SSL and TLS protocols (up to and including TLS/1.2) were vulnerable to a Man-in-the-Middle attack (CVE-2009-3555) during a renegotiation. This vulnerability allowed an attacker to "prefix" a chosen plaintext to the HTTP request as seen by the web server. A protocol extension was developed which fixed this vulnerability if supported by both client and server.

For more information on Man-in-the-Middle attack (CVE-2009-3555), see:

`https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-3555`

Default mode

When the directive SSLInsecureRenegotion is not specified in the configuration, Oracle HTTP Server operates in compatibility mode.

In this mode, vulnerable peers that do not have RI/SCSV support are allowed to connect, but renegotiation is allowed only with those peers that have RI/SCSV support.

SSLInsecureRenegotiation ON

This option allows vulnerable peers that do not have RI/SCSV to perform renegotiation. Hence this option must be used with caution, as it leaves the server vulnerable to the renegotiation attack described in CVE-2009-3555.

SSLInsecureRenegotiation OFF

If this option is used, the behavior of Oracle HTTP Server is similar to that described in **Default mode**.

Category	Value
Syntax	`SSLInsecureRenegotiation ON | OFF`
Example	`SSLInsecureRenegotiation ON`
Default	Off

To configure SSLInsecureRenegotiation, edit ssl.conf file and set `SSLInsecureRenegotiation ON/OFF` globally or virtually to enable disable insecure renegotiation.

G.2.8 SSLMutex

Type of semaphore (lock) for SSL engine's mutual exclusion of operations that have to be synchronized between Oracle HTTP Server processes. Accepted values are:

- `file`:path/to/mutex: Uses a file for locking. The process ID (PID) of the Oracle HTTP Server parent process is appended to the filename to ensure uniqueness. If the filename does not begin with a slash (/), it is assumed to be relative to `ServerRoot`. This setting is not available on Windows.
- `none`: Uses no mutex at all. Not recommended, because the mutex synchronizes the write access to the SSL session cache. If you do not configure a mutex, the session cache can become garbled.
- `pthread`: This directive tells the SSL Module to use Posix thread mutexes. It is only available if the underlying platform and Apache Portable Runtime (APR) supports it.
- `sem`: Uses an operating system semaphore to synchronize writes. On UNIX, it would be a Sys V IPC semaphore; on Windows, it is a Windows Mutex. This is the best choice, if the operating system supports it.

Category	Value
Syntax	`SSLMutex none | file | pthread | sem`
Example	`SSLMutex sem`
Default	`pthread`

Notes:

- In the Oracle HTTP Server default `ssl.conf` template file, `pthread` is defined as the default value for the SSLMutex directive for non-Windows platforms and `none` is defined as a default value for the Windows platform as follows:

```
<IfModule mpm_winnt_module>
   SSLMutex "none"
</IfModule>
<IfModule !mpm_winnt_module>
   SSLMutex pthread
</IfModule>
```

  As new Oracle HTTP Server instances are created for non-Windows platforms, the default value for SSLMutex will continue to be `pthread` unless you explicitly modify your configuration. If you comment out these lines in the `ssl.conf` file and no value is specified for SSLMutex in the Oracle HTTP Server configuration files, then Oracle HTTP Server will use `none` as the default value for SSLMutex.

- The `none` value for the SSLMutex directive is not recommended for non-Windows platforms, because it can create a garbled session cache and can lead to core dumps.

G.2.9 SSLNZTraceLogLevel

`SSLNZTraceLogLevel` adjusts the verbosity of the messages recorded in the NZ library error logs. When a particular level is specified, messages from all other levels of higher significance will be reported as well. For example, when `SSLNZTraceLogLevel ssl` is set, messages with log levels of error, warn, user and debug will also be posted.

SSLNZTraceLogLevel accepts the following log levels:

- `none`: NZ Trace disable
- `fatal`: Fatal error; system is unusable.
- `error`: Error conditions.
- `warn`: Warning conditions.
- `user`: Normal but significant condition.
- `debug`: Debug-level condition
- `ssl`: SSL level debugging

Category	Value
Syntax	`SSLNZTraceLogLevel none | fatal | error | warn | user | debug | ssl`
Example	`SSLNZTraceLogLevel fatal`
Default	None

G.2.10 SSLOptions

Controls various runtime options on a per-directory basis. In general, if multiple options apply to a directory, the most comprehensive option is applied (options are not

merged). However, if all of the options in an SSLOptions directive are preceded by a plus ('+') or minus ('-') symbol, then the options are merged. Options preceded by a plus are added to the options currently in force, and options preceded by a minus are removed from the options currently in force.

Accepted values are:

- StdEnvVars: Creates the standard set of CGI/SSI environment variables that are related to SSL. This is disabled by default because the extraction operation uses a lot of CPU time and usually has no application when serving static content. Typically, you only enable this for CGI/SSI requests.
- ExportCertData: Enables the following additional CGI/SSI variables:

  SSL_SERVER_CERT

  SSL_CLIENT_CERT

  SSL_CLIENT_CERT_CHAIN_n (where n= 0, 1, 2...)

  These variables contain the Privacy Enhanced Mail (PEM)-encoded X.509 certificates for the server and the client for the current HTTPS connection, and can be used by CGI scripts for deeper certificate checking. All other certificates of the client certificate chain are provided. This option is "Off" by default because there is a performance cost associated with using it.

  SSL_CLIENT_CERT_CHAIN_n variables are in the following order: SSL_CLIENT_CERT_CHAIN_0 is the intermediate CA who signs SSL_CLIENT_CERT. SSL_CLIENT_CERT_CHAIN_1 is the intermediate CA who signs SSL_CLIENT_CERT_CHAIN_0, and so forth, with SSL_CLIENT_ROOT_CERT as the root CA.
- FakeBasicAuth: Translates the subject distinguished name of the client X.509 certificate into an HTTP basic authorization user name. This means that the standard HTTP server authentication methods can be used for access control. Note that no password is obtained from the user; the string 'password' is substituted.
- StrictRequire: Denies access when, according to SSLRequireSSL or directives, access should be forbidden. Without StrictRequire, it is possible for a 'Satisfy any' directive setting to override the SSLRequire or SSLRequireSSL directive, allowing access if the client passes the host restriction or supplies a valid user name and password.

  Thus, the combination of SSLRequireSSL or SSLRequire with SSLOptions +StrictRequire gives mod_ossl the ability to override a 'Satisfy any' directive in all cases.
- CompatEnvVars: Exports obsolete environment variables for backward compatibility to Apache SSL 1.x, mod_ssl 2.0.x, Sioux 1.0, and Stronghold 2.x. Use this to provide compatibility to existing CGI scripts.
- OptRenegotiate: This enables optimized SSL connection renegotiation handling when SSL directives are used in a per-directory context.

Category	Value
Syntax	SSLOptions [+-] StdEnvVars | ExportCertData | FakeBasicAuth | StrictRequire | CompatEnvVars | OptRenegotiate
Example	SSLOptions -StdEnvVars
Default	None

G.2.11 SSLPassPhraseDialog

Type of pass phrase dialog for wallet access. `mod_ossl` asks the administrator for a pass phrase to access the wallet. Accepted values are:

- `builtin`: when the server is started, `mod_ossl` prompts for a password for each wallet.
- `exec:`*`path/to/program`* - when the server is started, `mod_ossl` calls an external program configured for each wallet. This program is invoked with two arguments: *`servername`*:*`portnumber`* and `RSA` or `DSA`.

Category	Value
Syntax	`SSLPassPhraseDialog builtin | exec`
Example	`SSLPassPhraseDialog exec:/usr/local/sbin/pfilter`
Default	`builtin`

G.2.12 SSLProtocol

Specifies SSL protocol(s) for `mod_ossl` to use when establishing the server environment. Clients can only connect with one of the specified protocols. Accepted values are:

- `SSLv3`
- `TLSv1`
- `TLSv1.1`
- `TLSv1.2`
- `All`

You can specify multiple values as a space-delimited list. In the syntax for SSLProtocol, the "-" and "+" symbols have the following meaning:

- + : Adds the protocol to the list
- - : Removes the protocol from the list

In the current release `All` is defined as `+SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2` (`SSLv2` is disabled out-of-the-box. You must explicitly disable `SSLv3` in this case.)

Note: Because of security concerns, Oracle strongly recommends that you disable the `SSLv3` security protocol from Oracle HTTP Server. For instructions on how to disable SSL, see "Disable SSL Security Protocols" in *Oracle HTTP Server Release Notes*.

Note: The syntax for the `SSLProtocol` directive can use either `TLSv1` as a value or the `nzos_Version_1_0` syntax (or `TLSv1.1` and `nzos_Version_1_1`, or `TLSv1.2` and `nzos_Version_1_2`).

If you are using Oracle Fusion Middleware Control, security will be configured using the `nzos*` syntax. Both options represent TLS 1.0 protocol version. The `nzos_Version_1_0` syntax is the Oracle representation of TLS1.0 and `TLSv1` is an open source representation. Oracle HTTP Server supports both ways to represent SSL protocol in its config files.

Category	Value
Syntax	`SSLProtocol [+-] SSLv3 | TLSv1 | TLSv1.1 | TLSv1.2 | All`
Example	`SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2`
Default	`ALL`

G.2.13 SSLProxyCipherSuite

Specifies the SSL cipher suite that the proxy can use during the SSL handshake. This directive uses a colon-separated cipher specification string to identify the cipher suite. Table G–1 shows the tags to use in the string to describe the cipher suite you want. SSLProxyCipherSuite accepts the following values:

- none: Adds the cipher to the list
- + : Adds the cipher to the list and place it in the correct location in the list
- - : Remove the cipher from the list (can be added later)
- ! : Remove the cipher from the list permanently

Tags are joined with prefixes to form a cipher specification string. The SSLProxyCipherSuite directive uses the same tags as the SSLCipherSuite directive. For a list of supported suite tags, see Table G–1.

Category	Value
Example	`SSLProxyCipherSuite` ALL:!MD5 In this example, all ciphers are specified except MD5 strength ciphers.
Syntax	`SSLProxyCipherSuite` *cipher-spec*
Default	`ALL:!ADH:+HIGH:+MEDIUM:+LOW`

The SSLProxyCipherSuite directive uses the same cipher suites as the SSLCipherSuite directive. For a list of the Cipher Suites supported in Oracle Advanced Security 12.1.2, see Table G–2.

G.2.14 SSLProxyEngine

Enables or disables the SSL/TLS protocol engine for proxy. SSLProxyEngine is usually used inside a `<VirtualHost>` section to enable SSL/TLS for proxy usage in a particular virtual host. By default, the SSL/TLS protocol engine is disabled for proxy both for the main server and all configured virtual hosts.

SSLProxyEngine should not be included in a virtual host that will be acting as a forward proxy (by using `Proxy` or `ProxyRequest` directives). SSLProxyEngine is not required to enable a forward proxy server to proxy SSL/TLS requests.

Category	Value
Syntax	SSLProxyEngine ON | OFF
Example	SSLProxyEngine on
Default	`Disable`

G.2.15 SSLProxyProtocol

Specifies SSL protocol(s) for `mod_ossl` to use when establishing a proxy connection in the server environment. Proxies can only connect with one of the specified protocols. Accepted values are:

- `SSLv3`
- `TLSv1`
- `TLSv1.1`
- `TLSv1.2`
- `All`

You can specify multiple values as a space-delimited list. In the syntax, the "-" and "+" symbols have the following meaning:

- + : Adds the protocol to the list
- - : Removes the protocol from the list

In the current release `All` is defined as `+SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2` (`SSLv2` is disabled out of the box. You must explicitly disable `SSLv3` in this case.)

Note: Because of security concerns, Oracle strongly recommends that you disable the `SSLv3` security protocol from Oracle HTTP Server. For instructions on how to disable SSL, see "Disable SSL Security Protocols" in *Oracle HTTP Server Release Notes*.

Note: The syntax for the `SSLProxyProtocol` directive can use either `TLSv1` as a value or the `nzos_Version_1_0` syntax (or `TLSv1.1` and `nzos_Version_1_1`, or `TLSv1.2` and `nzos_Version_1_2`).

If you are using Oracle Fusion Middleware Control, security will be configured using the `nzos*` syntax. Both options represent TLS 1.0 protocol version. The `nzos_Version_1_0` syntax is the Oracle representation of TLS1.0 and `TLSv1` is an open source representation. Oracle HTTP Server supports both ways to represent SSL protocol in its config files.

Category	Value
Syntax	`SSLProxyProtocol [+-] SSLv3 | TLSv1 | TLSv1.1 | TLSv1.2 | All`
Example	`SSLProxyProtocol +TLSv1 +TLSv1.1 +TLSv1.2`
Default	`ALL`

G.2.16 SSLProxyWallet

Specifies the location of the wallet with its WRL, specified as a filepath, that a proxy connection needs to use.

Category	Value
Syntax	`SSLProxyWallet file:path to wallet`

Category	Value
Example	`SSLProxyWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/proxy"`
Default	None

G.2.17 SSLRequire

Denies access unless an arbitrarily complex boolean expression is true.

Category	Value
Syntax	`SSLRequire` *`expression`* (see Understanding the Expression)
Example	`SSLRequire word ">=" word |word "ge" word`
Default	None

Understanding the Expression

The *expression* must match the following syntax (given as a BNF grammar notation):

```
expr ::= "true" | "false"
"!" expr
expr "&&" expr
expr "||" expr
"(" expr ")"

comp ::=word "==" word | word "eq" word
word "!=" word |word "ne" word
word "<" word |word "lt" word
word "<=" word |word "le" word
word ">" word |word "gt" word
word ">=" word |word "ge" word
word "=~" regex
word "!~" regex
wordlist ::= word
wordlist "," word

word ::= digit
cstring
variable
function

digit ::= [0-9]+

cstring ::= "..."

variable ::= "%{varname}"
```

Table G–3 and Table G–4 list standard and SSL variables. These are valid values for `varname`.

```
function ::= funcname "(" funcargs ")"
```

For `funcname`, the following function is available:

`file(`*`filename`*`)`

The file function takes one string argument, the filename, and expands to the contents of the file. This is useful for evaluating the file's contents against a regular expression.

Table G–3 lists the standard variables for SSLRequire `varname`.

Table G–3 Standard Variables for SSLRequire Varname

Standard Variables	Standard Variables	Standard Variables
HTTP_USER_AGENT	PATH_INFO	AUTH_TYPE
HTTP_REFERER	QUERY_STRING	SERVER_SOFTWARE
HTTP_COOKIE	REMOTE_HOST	API_VERSION
HTTP_FORWARDED	REMOTE_IDENT	TIME_YEAR
HTTP_HOST	IS_SUBREQ	TIME_MON
HTTP_PROXY_CONNECTION	DOCUMENT_ROOT	TIME_DAY
HTTP_ACCEPT	SERVER_ADMIN	TIME_HOUR
HTTP:headername	SERVER_NAME	TIME_MIN
THE_REQUEST	SERVER_PORT	TIME_SEC
REQUEST_METHOD	SERVER_PROTOCOL	TIME_WDAY
REQUEST_SCHEME	REMOTE_ADDR	TIME
REQUEST_URI	REMOTE_USER	ENV:variablename
REQUEST_FILENAME		

Table G–4 lists the SSL variables for SSLRequire varname.

Table G–4 SSL Variables for SSLRequire Varname

SSL Variables	SSL Variables	SSL Variables
HTTPS	SSL_PROTOCOL	SSL_CIPHER_ALGKEYSIZE
SSL_CIPHER	SSL_CIPHER_EXPORT	SSL_VERSION_INTERFACE
SSL_CIPHER_USEKEYSIZE	SSL_VERSION_LIBRARY	SSL_SESSION_ID
SSL_CLIENT_V_END	SSL_CLIENT_M_SERIAL	SSL_CLIENT_V_START
SSL_CLIENT_S_DN_ST	SSL_CLIENT_S_DN	SSL_CLIENT_S_DN_C
SSL_CLIENT_S_DN_CN	SSL_CLIENT_S_DN_O	SSL_CLIENT_S_DN_OU
SSL_CLIENT_S_DN_G	SSL_CLIENT_S_DN_T	SSL_CLIENT_S_DN_I
SSL_CLIENT_S_DN_UID	SSL_CLIENT_S_DN_S	SSL_CLIENT_S_DN_D
SSL_CLIENT_I_DN_C	SSL_CLIENT_S_DN_Email	SSL_CLIENT_I_DN
SSL_CLIENT_I_DN_O	SSL_CLIENT_I_DN_ST	SSL_CLIENT_I_DN_L
SSL_CLIENT_I_DN_T	SSL_CLIENT_I_DN_OU	SSL_CLIENT_I_DN_CN
SSL_CLIENT_I_DN_S	SSL_CLIENT_I_DN_I	SSL_CLIENT_I_DN_G
SSL_CLIENT_I_DN_Email	SSL_CLIENT_I_DN_D	SSL_CLIENT_I_DN_UID
SSL_CLIENT_CERT	SSL_CLIENT_CERT_CHAIN_n	SSL_CLIENT_ROOT_CERT
SSL_CLIENT_VERIFY	SSL_CLIENT_M_VERSION	SSL_SERVER_M_VERSION
SSL_SERVER_V_START	SSL_SERVER_V_END	SSL_SERVER_M_SERIAL
SSL_SERVER_S_DN_C	SSL_SERVERT_S_DN_ST	SSL_SERVER_S_DN
SSL_SERVER_S_DN_OU	SSL_SERVER_S_DN_CN	SSL_SERVER_S_DN_O
SSL_SERVER_S_DN_I	SSL_SERVER_S_DN_G	SSL_SERVER_S_DN_T
SSL_SERVER_S_DN_D	SSL_SERVER_S_DN_UID	SSL_SERVER_S_DN_S

Table G–4 (Cont.) SSL Variables for SSLRequire Varname

SSL Variables	SSL Variables	SSL Variables
SSL_SERVER_I_DN	SSL_SERVER_I_DN_C	SSL_SERVER_S_DN_Email
SSL_SERVER_I_DN_L	SSL_SERVER_I_DN_O	SSL_SERVER_I_DN_ST
SSL_SERVER_I_DN_CN	SSSL_SERVER_I_DN_T	SSL_SERVER_I_DN_OU
SSL_SERVER_I_DN_G	SSL_SERVER_I_DN_I	

G.2.18 SSLRequireSSL

Denies access to clients not using SSL. This is a useful directive for absolute protection of a SSL-enabled virtual host or directories in which configuration errors could create security vulnerabilities.

Category	Value
Syntax	SSLRequireSSL
Example	SSLRequireSSL
Default	None

G.2.19 SSLSessionCache

Specifies the global/interprocess session cache storage type. The cache provides an optional way to speed up parallel request processing. The accepted values are:

- dc:UNIX:/path/to/socket: This makes use of the distcache distributed session caching libraries. The argument should specify the location of the server or proxy to be used using the distcache address syntax; for example, UNIX:/path/to/socket specifies a UNIX domain socket (typically a local dc_client proxy); IP:server.example.com:9001 specifies an IP address.
- none: disables the global/interprocess session cache. Produces no impact on functionality, but makes a major difference in performance.
- nonenotnull: This disables any global/inter-process Session Cache. However it does force OpenSSL to send a non-null session ID to accommodate buggy clients that require one.
- shmcb:/path/to/datafile[bytes]: Uses a high-performance Shared Memory Cyclic Buffer (SHMCB) session cache to synchronize the local SSL memory caches of the server processes. The performance of shmcb is more uniform in all environments when compared to shmht. Note: in this shm setting, no log files are created under /path/to/datafile on local disk.
- shmht:/path/to/datafile[bytes]: Uses a high-performance hash table (bytes specifies approximate size) inside a shared memory segment in RAM, which is established by the /path/to/datafile. This hash table synchronizes the local SSL memory caches of the server processes. Note: in this shm setting, no log files are created under /path/to/datafile on local disk.

Category	Value
Syntax	SSLSessionCache dc:UNIX:/path/to/socket | none |nonetonull | shmcb:/path/to/datafile[bytes] | shmht:/path/to/datafile[bytes]

Category	Value
Examples	SSLSessionCache "shmcb:${ORACLE_ INSTANCE}/servers/${COMPONENT_NAME}/logs/ssl_ scache(512000)"
Default	`SSLSessionCache` *none*

G.2.20 SSLSessionCacheTimeout

Specifies the number of seconds before a SSL session in the session cache expires.

Category	Value
Syntax	`SSLSessionCacheTimeout` *seconds*
Example	`SSLSessionCacheTimeout 120`
Default	300

G.2.21 SSLVerifyClient

Specifies whether a client must present a certificate when connecting. The accepted values are:

- `none`: No client certificate is required
- `optional`: Client can present a valid certificate
- `require`: Client must present a valid certificate

Category	Value
Syntax	`SSLVerifyClient none | optional | require`
Example	`SSLVerifyClient optional`
Default	None

Note: The level `optional_no_ca` included with `mod_ssl` (in which the client can present a valid certificate, but it need not be verifiable) is not supported in `mod_ossl`.

G.2.22 SSLWallet

Specifies the location of the wallet with its WRL, specified as a filepath.

Category	Value
Syntax	`SSLWallet` *file:path to wallet*
Example	`SSLWallet "${ORACLE_ INSTANCE}/config/fmwconfig/components/${COMPONENT_ TYPE}/instances/${COMPONENT_NAME}/keystores/default"`
Default	None

G.3 mod_plsql

The mod_plsql configuration parameters are described in these sections:

- Section G.3.1, "plsql.conf"
- Section G.3.2, "dads.conf"
- Section G.3.3, "cache.conf"

G.3.1 plsql.conf

The following parameters are used with the `plsql.conf` file:

- PlsqlDMSEnable
- PlsqlLogEnable
- PlsqlLogDirectory
- PlsqlIdleSessionCleanupInterval

G.3.1.1 PlsqlDMSEnable

Enables Dynamic Monitoring Service (DMS) for the mod_plsql module.

Category	Value
Syntax	`PlsqlDMSEnable On | Off`
Example	`PlsqlDMSEnable On`
Default	`On`

G.3.1.2 PlsqlLogEnable

Enables debug level logging for the mod_plsql module. Debug level logging is meant to be used for debugging purposes only.

When logging is enabled, Oracle HTTP Server log files are typically created in the `PlsqlCacheDirectory` *DOMAIN_HOME*/servers/*componentName*/ directory. However, the location specified in PlsqlLogDirectory determines the final location.

This parameter should be set to `Off` unless recommended by Oracle support to debug problems with the mod_plsql module.

To view more details about the internal processing of the mod_plsql module, set this directive to `On`. This causes the mod_plsql module to start logging every request that is processed. The log files are generated as specified by the `PlsqlLogDirectory` directive.

Category	Value
Syntax	`PlsqlLogEnable On | Off`
Example	`PlsqlLogEnable Off`
Default	`Off`

G.3.1.3 PlsqlLogDirectory

Specifies the directory where debug level logs are written.

Set the directory name of the location where log files should be generated when logging is enabled. To avoid possible confusion about the location of this directory, an absolute path is recommended.

On UNIX, this directory must have write permissions by the owner of the child `httpd` processes.

Category	Value
Syntax	`PlsqlLogDirectory` *directory*
Example	PlsqlLogDirectory "${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs"
Default	None

G.3.1.4 PlsqlIdleSessionCleanupInterval

Specifies the time (in minutes) in which the idle database sessions should be closed and cleaned by the mod_plsql module.

This directive is used with connection pooling of database connections and sessions in the mod_plsql module. When a session is not used for the specified amount of time, it is closed and freed. This is done so that unused sessions can be cleaned, and the memory is freed on the database side.

Setting this time to a low number helps in faster cleanup of unused database sessions. If this number is too low, then this may adversely affect the performance benefits of connection pooling in the mod_plsql module.

If the number of open database sessions is not a concern, you can increase the value of this parameter for best performance. In such a case, if the site is accessed frequently enough that the idle session cleanup interval is never reached for a session, then the DAD configuration parameter PlsqlMaxRequestsPerSession can be modified so that it is guaranteed that a pooled database session gets recycled on a regular basis.

For most installations, the default value is adequate.

Category	Value
Syntax	`PlsqlIdleSessionCleanupInterval` *number*
Example	`PlsqlIdleSessionCleanupInterval 10`
Default	15 (minutes)

G.3.2 dads.conf

The `dads.conf` file contains the configuration parameters for the PL/SQL database access descriptor. (See Table G–1 for the file location.) A DAD is a set of values that specifies how the mod_plsql module connects to a database server to fulfill a HTTP request.

The following parameters are used with the `dads.conf` file:

G.3.2.1 PlsqlAfterProcedure

Specifies the procedure to be invoked after calling the requested procedure. This enables you to put a hook point after the requested procedure is called. This is useful in doing SQL*Traces/SQL Profiles while debugging a problem with the requested procedure. This is also useful when you want to ensure that a specific call is made after running every procedure.

Category	Value
Syntax	`PlsqlAfterProcedure` *`string`*
Example	`PlsqlAfterProcedure portal.mypkg.myafterproc`
Default	None

> **Note:** This parameter should only be used for debugging purposes. In addition, you could use this parameter to stop SQL trace/SQL profiling.

G.3.2.2 PlsqlAlwaysDescribeProcedure

Specifies whether the mod_plsql module should describe a procedure before trying to run it. If this is set to `On`, then the mod_plsql module will always describe a procedure before invoking it. Otherwise, the mod_plsql module will only describe a procedure when its internal heuristics have interpreted a parameter type incorrectly.

Category	Value
Syntax	`PlsqlAlwaysDescribeProcedure On | Off`
Example	`PlsqlAlwaysDescribeProcedure On`
Default	`Off`

Note: This parameter should only be used for debugging purposes.

G.3.2.3 PlsqlAuthenticationMode

Specifies the authentication mode to use for allow access through the DAD. The accepted values for `PlsqlAuthenticationMode` are `Basic`, `SingleSignOn`, `GlobalOwa`, `CustomOwa`, `PerPackageOwa`.

Category	Value
Syntax	`PlsqlAuthenticationMode Basic | SingleSignOn | GlobalOwa | CustomOwa | PerPackageOwa`
Example	`PlsqlAuthenticationMode CustomOwa`
Default	`Basic`

- `Basic` is the default mode and determines whether to ask for username and password if they are not provided with `PlsqlDatabaseUsername` and `PlsqlDatabasePassword`. This setting is required for WebDB 2.x applications. If the DAD is not using the Basic authentication, then you must include a valid username/password in the DAD configuration.
- `SingleSignOn` specifies that you want to use Single Sign-On server. This is required for DADs using Oracle9iAS Portal. As already stated the provided username and password need to be the one from your single sign-on server.
- `GlobalOwa`, `CustomOwa`, and `PerPackageOwa` are used only by very few PL/SQL applications. Custom authentication enables applications to authenticate users within the application itself, not at the database level.

  Authorization is performed by invoking a user-written authorization function. Custom authentication uses a static username/password that is stored in the DAD. It cannot be combined with dynamic username/password authentication. To enable custom authentication, set the level of authentication for PlsqlAuthenticationMode and implement the authorize function.

You should also be aware of the following:

- If the DAD is not using the Basic authentication, then you must include a valid username/password in the DAD configuration. For the Basic mode, to perform dynamic authentication, the DAD username/password parameters must be omitted.
- The SingleSignOn mode is supported only for Oracle Fusion Middleware releases, and is used by Oracle Portal and Oracle Single Sign-On. Most customer applications use Basic authentication. Custom authentication modes (GlobalOwa, CustomOwa, and PerPackageOwa) are used by very few PL/SQL applications.

G.3.2.4 PlsqlBeforeProcedure

Specifies the procedure to be invoked before calling the requested procedure. This enables you to put a hook point before the requested procedure is called. This is useful in doing SQL*Traces/SQL Profiles while debugging a problem with the requested procedure. This is also useful when you want to ensure that a specific call be made before running every procedure.

Category	Value
Syntax	`PlsqlBeforeProcedure` *`string`*
Example	`PlsqlBeforeProcedure portal.mypkg.mybeforeproc`
Default	None

Note: This parameter should only be used for debugging purposes. In addition, you could use this parameter to start SQL Trace/SQL Profiling.

G.3.2.5 PlsqlBindBucketLengths

Note: This configuration property is rarely ever changed, and system defaults suffice in most cases.

Specifies the rounding size to use while binding the number of elements in a collection bind. While executing PL/SQL statements, the Oracle database maintains a cache of PL/SQL statements in the shared SQL area, and attempts to reuse the cached statement if the same statement is run again. Oracle's matching criteria requires that the statement texts be identical, and that the bind variable data types match. Unfortunately, the type match for strings is sensitive to the exact byte size specified, and for collection bindings is also sensitive to the number of elements in the collection. Since the mod_plsql module binds statements dynamically, the odds of hitting the shared cache are low, and it may fill up with near-duplicates and lead to contention for the latch on the shared area. This parameter reduces that effect by bucketing bind lengths to the nearest level.

All numbers specified should be in ascending order. After the last specified size, subsequent bucket sizes will be assumed to be twice the last one.

Category	Value
Syntax	`PlsqlBindBucketLengths` *`number multiline`*

Example	`PlsqlBindBucketLengths 4`
	`PlsqlBindBucketLengths 25`
	`PlsqlBindBucketLengths 125`

Default	4,20,100,400

- This parameter is relevant only if you are using procedures with array parameters, and passing varying number of parameters to the procedure.
- The default should be sufficient for most PL/SQL applications.
- To see if this parameter must be changed, check the number of versions of a SQL statement in the SQL area.
- After the higher configured value, mod_plsql starts auto-generating bucket sizes of larger values by doubling the last value, as needed. Therefore, after 400, the next bucket value becomes 800, then 1600, and so on.
- Consider using flexible parameter passing to reduce the problem.

G.3.2.6 PlsqlBindBucketWidths

Note: This configuration property is rarely ever changed, and system defaults suffice in most cases.

Specifies the rounding size to use while binding the number of elements in a collection bind. While executing PL/SQL statements, the Oracle database maintains a cache of PL/SQL statements in the shared SQL area, and attempts to reuse the cached statement if the same statement is run again. Oracle's matching criteria requires that the statement texts be identical, and that the bind variable data types match. Unfortunately, the type match for strings is sensitive to the exact byte size specified, and for collection bindings is also sensitive to the number of elements in the collection. Since the mod_plsql module binds statements dynamically, the odds of hitting the shared cache are low, and it may fill up with near-duplicates and lead to contention for the latch on the shared area. This parameter reduces that effect by bucketing bind widths to the nearest level.

All numbers specified should be in ascending order. After the last specified size, subsequent bucket sizes will be assumed to be twice the last one.

The last bucket width must be equal to or less than 4000. This is due to the restriction imposed by OCI where array bind widths cannot be greater than 4000.

Category	Value
Syntax	`PlsqlBindBucketWidths` *`number multiline`*

Example	`PlsqlBindBucketWidths 40`
	`PlsqlBindBucketWidths 400`
	`PlsqlBindBucketWidths 2000`

Default	`32,128,1450,2048,4000`

- This parameter is relevant only if you are using procedures with array parameters, and passing varying number of parameters to the procedure.
- The default should be sufficient for most PL/SQL applications.
- To see if this parameter must be changed, check the number of versions of a SQL statement in the SQL area.
- After the higher configured value, mod_plsql starts auto-generating bucket sizes of larger values by doubling the last value, as needed. Therefore, after 400, the next bucket value becomes 800, then 1600, and so on.
- Consider using flexible parameter passing to reduce the problem.

G.3.2.7 PlsqlCGIEnvironmentList

Specifies overrides and additions of CGI environment variables to the default set of environment variables passed to a PL/SQL procedure. This is a multi-line directive of name-value pairs to be added, overridden or removed. You can only specify one environment variable for each directive.

You can add CGI environment variables from the Oracle HTTP Server environment by specifying the variable name. To remove a CGI environment variable, set it equal to blank. To add your own name-value pair, use the syntax `myname=myvalue`.

Category	Value
Syntax	`PlsqlCGIEnvironmentList string` *`multiline`*
Default	None

Example	■ To add a new environment variable from the Oracle HTTP Server environment:
	`PlsqlCGIEnvironmentList DOCUMENT_ROOT`
	■ To remove an environment variable:
	`PlsqlCGIEnvironmentList MYENVAR2=`
	■ To override from the Oracle HTTP Server environment:
	`PlsqlCGIEnvironmentList REQUEST_PROTOCOL=HTTPS`
	■ To add your own environment variable:
	`PlsqlCGIEnvironmentList MY_VARNAME=MY_VALUE`

- Environment variables added here are available in the PL/SQL application through the function `owa_util.get_cgi_env`.

G.3.2.8 PlsqlConnectionTimeout

Specifies the timeout in milliseconds for testing a connection pool in the mod_plsql module.

Category	Value
Syntax	`PlsqlConnectionTimeout` *`number`*
Example	`PlsqlConnectionTimeout 5000`
Default	10000 (milliseconds)

When PlsqlConnectionValidation is set to `Automatic` or `AlwaysValidate`, the mod_plsql module attempts to test pooled database connections. This parameter specifies the maximum time the mod_plsql module should wait for the test request to complete before it assumes that the connection is not usable.

G.3.2.9 PlsqlConnectionValidation

Specifies the mechanism the mod_plsql module should use to detect terminated connections in its connection pool.

> **Note:** This configuration property is rarely ever changed, and system defaults suffice in most cases.

For performance reasons, the mod_plsql module pools database connections. If a database instance goes down, and the mod_plsql module was maintaining a pool of connections to the instance, then each pooled database connection results in an error when it is next used to service a request. This can be a concern in high availability configurations such as Oracle RAC where even if one node goes down, other nodes servicing the database might have been able to service the request successfully. The mod_plsql module provides for a mechanism whereby it can self-correct after it detects a failure that could be caused by a database node going down. This mechanism to self-correct is controlled by the parameter PlsqlConnectionValidation.

The following are the valid values for PlsqlConnectionValidation:

- `Automatic`: The mod_plsql module tests all pooled database connections which were created before the detection of a failure that could mean an instance failure.
- `ThrowAwayOnFailure`: The mod_plsql module throws away all pooled database connections which were created before the detection of a failure that could mean an instance failure.
- `AlwaysValidate`: The mod_plsql module always tests all pooled database connections which were created before issuing a request. Since this option has an associated performance overhead for each request, this should be used with caution.
- `NeverValidate`: The mod_plsql module never pings any pooled database connection.

Category	Value
Syntax	`PlsqlConnectionValidation Automatic | ThrowAwayOnFailure | AlwaysValidate | NeverValidate`
Example	`PlsqlConnectionValidation ThrowAwayOnFailure`
Default	`Automatic`

When the mod_plsql module encounters one of the following errors, it assumes that the database may have been down.

- `00443 – background process <string> did not start`
- `00444 – background process <string> failed while starting`
- `00445 – background process did not start after <x> seconds`
- `00447 – fatal error in background processes`
- `00448 – normal completion of background process`
- `00449 – background process <string> unexpectedly terminated with error`
- `00470 – LGWR process terminated with error`
- `00471 – DBWR process terminated with error`
- `00472 – PMON process terminated with error`
- `00473 – ARCH process terminated with error`
- `00474 – SMON process terminated with error`
- `00475 – TRWR process terminated with error`
- `00476 – RECO process terminated with error`
- `00480 – LCK* process terminated with error`
- `00481 – LMON process terminated with error`
- `00482 – LMD* process terminated with error`
- `00484 – LMS* process terminated with error`
- `00485 – DIAG process terminated with error`
- `01014 – ORACLE shutdown in progress`
- `01033 – ORACLE initialization or shutdown in progress`

- 01034 – ORACLE not available
- 01041 – internal error. hostdef extension doesn't exist
- 01077 – background process initialization failure
- 01089 – immediate shutdown in progress- no operations permitted
- 01090 – shutdown in progress- connection is not permitted
- 01091 – failure during startup force
- 01092 – ORACLE instance terminated. Disconnection forced
- 03106 – fatal two-task communication protocol error
- 03113 – end-of-file on communication channel
- 03114 – not connected to ORACLE
- 12570 – TNS: packet reader failure
- 12571 – TNS: packet writer failure

G.3.2.10 PlsqlDatabaseConnectString

Specifies the connection to an Oracle database.

Category	Value
Syntax	`PlsqlDatabaseConnectString` *`string`* `{ServiceNameFormat | SIDFormat | TNSFormat | NetServiceNameFormat}` The *string* parameter depends on the second argument: ■ If the second argument is `ServiceNameFormat`, *string* is *`HOST:PORT:SERVICE_NAME`*, where *`HOST`* is the host name running the database, *`PORT`* is the port number the TNS listener is listening at, and *`SERVICE_NAME`* is the database service name. An IPv6 address can be specified using the format *[IPv6_ADDRESS]:PORT:SERVICE_NAME*. ■ If the second argument is `SIDFormat`, *string* is *HOST:PORT:SID* where *`HOST`* is the host name running the database, *`PORT`* is the port number the TNS listener is listening at, and *`SID`* is the database SID. An IPv6 address can be specified using the format *[IPv6_ADDRESS]:PORT:SID*. ■ If the second argument is `TNSFormat`, *string* is a valid TNS alias that can be resolved using Oracle Net utilities like tnsping and SQL*Plus. ■ If the second argument is `NetServiceNameFormat`, *string* is a valid net service name that can be resolved to a connect descriptor. A connect descriptor is a specially formatted description of the destination for a network connection. A connect descriptor contains destination service and network route information. If the format argument is not specified, then the mod_plsql module assumes the *string* is either in the HOST:PORT:SID format, or resolvable by Oracle Net. The differentiation between the two is made by the presence of the colon in the specified string. It is recommended that newer DADs do not use the `SIDFormat` syntax. This exists only for backward compatibility reasons. Use the new two argument format for newly created DADs.

Category	Value
Example	■ `PlsqlDatabaseConnectString example.com:1521:myhost.iasdb.inst ServiceNameFormat` ■ `PlsqlDatabaseConnectString [2001:DB8:f1ff:f1ff]:1521:myhost.iasdb.inst ServiceNameFormat` ■ `PlsqlDatabaseConnectString example.com:1521:iasdb SIDFormat` ■ `PlsqlDatabaseConnectString [2001:DB8:ff1ff:f1ff]:1521:iasdb SIDFormat` ■ `PlsqlDatabaseConnectString myhost_tns TNSFormat` ■ `PlsqlDatabaseConnectString cn=oracle,cn=iasdb NetServiceNameFormat` ■ `PlsqlDatabaseConnectString (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(Host=`*`example.com`*`)(Port=1521))(CONNECT_DATA=(SID=iasdb))) TNSFormat` ■ `PlsqlDatabaseConnectString myhost_tns` ■ `PlsqlDatabaseConnectString example.com:1521:iasdb`
Default	None

- If the database is running in the same Oracle home, or the environment variable TWO_TASK is set, then this parameter need not be specified.
- If the database is running in a separate Oracle home, then this parameter is mandatory.
- If you have problems connecting to the database:
  - Check the username and password information in the DAD.
  - Make sure that you run `tnsping db_connect_string`, and commands such as:

    `sqlplus` *`DADUsername/DADPassword@db_connect_string`*

  - Ensure that TNS_ADMIN is configured properly.
  - Verify that the HOST:PORT:SERVICE_NAME format works correctly.
  - Ensure that the TNS listener and database are up and running.
  - Ensure that you can ping the host from this machine.
- From a the mod_plsql module perspective, `TNSFormat` and `NetServiceNameFormat` are synonymous and denote connect descriptors that are resolved by Oracle Net. The `TNSFormat` is provided as a convenience so that end-users use this to signify that the name resolution happens through the local tnsnames.ora. For situations where the resolution is through an LDAP lookup as configured in sqlnet.ora, it is recommended that the format specifier of `NetServiceNameFormat` be used.

  If your database supports high availability, for example, Oracle Real Application Clusters database, it is highly recommended that you use the `NetServiceNameFormat` such that the resolution for the net service name is through LDAP. This enables you to add or remove Oracle RAC nodes accessible through the mod_plsql module by changing Oracle Internet Directory with the new or deleted node information. In such situations, hard-coding database listener HOST:PORT information in dads.conf or in the local `tnsnames.ora` is not recommended.

G.3.2.11 PlsqlDatabasePassword

Specifies the password to use to log in to the database.

Category	Value
Syntax	`PlsqlDatabasePassword` *string*
Example	`PlsqlDatabasePassword tiger`
Default	None

- This is a mandatory parameter, except for a DAD that sets PlsqlAuthenticationMode to Basic and uses dynamic authentication.
- For DADs using SingleSignOn authentication, this parameter uses the name of the schema owner.

After making manual configuration changes to DAD passwords, you should obfuscate the DAD passwords by running the `dadTool.pl` script, located in `ORACLE_HOME/bin`.

To obfuscate DAD passwords:

1. If necessary, change the user to the Oracle software owner user, typically `oracle`, using the following command:

   ```
   $ su - oracle
   ```

2. Set the *ORACLE_HOME* environment variable to specify the path to the Oracle home directory for the current release, and set the PATH environment variable to include the directory containing the Perl executable and the location of the `dadTool.pl` script.

   Bourne, Bash, or Korn shell:

   ```
   $ ORACLE_HOME=new_ORACLE_HOME_path;export ORACLE_HOME
   $ PATH=ORACLE_HOME/bin:ORACLE_HOME/perl/bin:$PATH;export PATH
   ```

   C or tcsh shell:

   ```
   % setenv ORACLE_HOME new_ORACLE_HOME_PATH
   % setenv PATH ORACLE_HOME/bin:ORACLE_HOME/perl/bin:PATH
   ```

   On Microsoft Windows, set the PATH and PERL5LIB environment variable:

   ```
   set PATH=ORACLE_HOME\bin;ORACLE_HOME\perl\bin;%PATH%
   set PERL5LIB=ORACLE_HOME\perl\lib
   ```

3. On UNIX platforms, set the shared library path environment variable.

   Include the `ORACLE_HOME/lib` or `lib32` directory in your shared library path. Table G–5 shows the appropriate directory and environment variable for each platform.

Table G–5 Shared Library Path Environment Variable

Platform	Environment Variable	Include Directory
AIX Based Systems	LIBPATH	ORACLE_HOME/lib
HP-UX PA-RISC	SHLIB_PATH	ORACLE_HOME/lib
Solaris Operating System	LD_LIBRARY_PATH	ORACLE_HOME/lib32
Other UNIX platforms, including Linux and HP Tru64 UNIX	LD_LIBRARY_PATH	ORACLE_HOME/lib

   For example, on HP-UX PA-RISC systems, set the SHLIB_PATH environment to include the `ORACLE_HOME/lib` directory:

```
SHLIB_PATH=$ORACLE_HOME/lib:$SHLIB_PATH;export SHLIB_PATH
```

4. Change directory to the bin directory for the current release of Oracle HTTP Server:

   ```
   cd $ORACLE_HOME/ohs/bin
   ```

5. Invoke the following Perl script to obfuscate DAD password:

   ```
   perl dadTool.pl -f dadfilename
   ```

   where *dadfilename* is the filename for `dads.conf`, which includes the full path to the DAD file.

   For example:

   ```
   perl dadTool.pl -f
   /u01/app/oracle/user_projects/domains/base_
   domain/config/fmwconfig/components/OHS/ohs1/mod_plsql/dads.conf
   ```

G.3.2.12 PlsqlDatabaseUserName

Specifies the username to use to log in to the database.

Category	Value
Syntax	`PlsqlDatabaseUsername` *string*
Example	`PlsqlDatabaseUsername scott`
Default	None

- This is a mandatory parameter, except for a DAD that sets PlsqlAuthenticationMode to Basic and uses dynamic authentication.
- For DADs using SingleSignOn authentication, this parameter is the name of the schema owner.

G.3.2.13 PlsqlDefaultPage

Specifies the default procedure to call if none is specified in the URL.

Category	Value
Syntax	`PlsqlDefaultPage` *string*
Example	`PlsqlDefaultPage myschema.mypackage.home`
Default	None

You can also use Oracle HTTP Server Rewrite rules to achieve the same effect as you get by setting this configuration parameter.

G.3.2.14 PlsqlDocumentPath

Specifies a virtual path in the URL that initiates document download from the document table. For example, if this parameter is set to `docs`, then the following URLs will start the document downloading process for URLs of the format:

```
/pls/dad/docs
/pls/plsqlapp/docs
```

Category	Value
Syntax	`PlsqlDocumentPath` *`string`*
Example	`PlsqlDocumentPath docs`
Default	`docs`

Omit this parameter for applications that do not perform document uploads or downloads.

G.3.2.15 PlsqlDocumentProcedure

Specifies the procedure to call when a document download is initiated. This procedure is called to process the download.

Category	Value
Syntax	`PlsqlDocumentProcedure` *`string`*
Example	`PlsqlDocumentProcedure portal.wwdoc_process.process_download`
Default	None

Omit this parameter for applications that do not perform document uploads or downloads.

G.3.2.16 PlsqlDocumentTablename

Specifies the table in the database to which all documents are uploaded.

Category	Value
Syntax	`PlsqlDocumentTablename` *`string`*
Example	`PlsqlDocumentTablename myschema.document_table`
Default	None

Omit this parameter for applications that do not perform document uploads or downloads.

G.3.2.17 PlsqlErrorStyle

Specifies the error reporting mode for mod_plsql errors.

Category	Value

Syntax	`PlsqlErrorStyle {ApacheStyle | ModplsqlStyle | DebugStyle}`
	■ `ApacheStyle`: The mod_plsql module indicates to Oracle HTTP Server the HTTP error that was encountered. Oracle HTTP Server then generates the error page. This can be used with the Oracle HTTP Server `ErrorDocument` directive to produce customized error messages.
	■ `ModplsqlStyle`: The mod_plsql module generates the error pages, usually a short message indicating the PL/SQL error encountered and PL/SQL exception stack, if any. For example:
	`scott.foo PROCEDURE NOT FOUND`
	■ `DebugStyle`: This mode provides more details than `ModplsqlStyle`. The mod_plsql module provides more details about the URL and parameters, and also produces server configuration information. This mode is for debugging purposes only. Do not use this in a production system, since displaying internal server variables could be a security risk.

Example	`PlsqlErrorStyle ModplsqlStyle`
Default	`ApacheStyle`

G.3.2.18 PlsqlExclusionList

Specifies a pattern for procedures, packages, or schema names which are forbidden to be directly run from a browser. This is a multi-line directive in which each pattern is on a separate line. The pattern is not case sensitive and can accept a wildcard such as an asterisk (*). The default patterns disallowed from direct URL access are as follows:

- sys.*
- dbms_*
- utl_*
- owa_util*
- owa.*
- htp.*
- htf.*
- wpg_docload.*

Setting this directive to `#NONE#` will disable all protection. This is strongly discouraged for an active site and should not be done. It may be used for debugging purposes.

If this parameter is overridden, the defaults still apply, which means that you do not have to explicitly add the default list to the list of excluded patterns.

Category	Value
Syntax	`PlsqlExclusionList {string | "#NONE#" multiline}`

Category	Value

Example	`PlsqlExclusionList myschema.private.*`
	`PlsqlExclusionList myschema.private1.*`
	will disallow access to URLs which contain one of:
	`sys.*, dbms_*, utl_*, owa_util*, owa.*, htp.*, htf.*, wpg_docload.*, myschema.private.*, myschema.private1.*`
	`PlsqlExclusionList "#NONE#"`
	will disable all protection. Its use is strongly discouraged for an active site.

Default	sys.* dbms_* utl_* owa_util* owa.* htp.* htf.* wpg_docload.*

- In addition to the patterns specified with this parameter, the mod_plsql module disallows any procedure name which contains the following special characters:
  - tabs
  - new lines
  - carriage-return
  - single quotation mark
  - reverse slash
  - form feed
  - left parenthesis
  - right parenthesis
  - space

  This cannot be changed.

G.3.2.19 PlsqlFetchBufferSize

Specifies the number of rows of content to fetch from the database for each trip, using either owa_util.get_page or owa_util.get_page_raw.

By default, the mod_plsql module attempts to fetch 200 response lines of output where each line is of 255 bytes. In situations where the response bytes are single-bytes, the response buffer is populated to the maximum and can pack 255*200=51000 bytes for each round trip. For responses containing multibyte data, the byte packing for each row could be less than ideal resulting in lesser bytes getting transferred for each round trip. If your application generates large pages frequently and the response does not fit in one round trip, then consider setting this parameter higher. The memory usage for the mod_plsql module will increase.

Category	Value
Syntax	`PlsqlFetchBufferSize` *number*
Example	`PlsqlFetchBufferSize 256`
Default	200

- This parameter is changed only for performance reasons. The minimum value for this parameter is 28, but it is seldom reduced.
- Change this parameter only under the following circumstances:
  - The average response page is large and you want to reduce the number of round-trips the mod_plsql module makes to the database to fetch the response.
  - The character set in use is multibyte, and you want to compensate for the problem of `get_page` or `get_page_raw` fetching fewer bytes for each row. Calculations in the PL/SQL Web ToolKit are character-based and in the case of multibyte characters, OWA packages assume a worst-case character byte size and do not attempt to pack each row to its maximum.

G.3.2.20 PlsqlInfoLogging

Specifies what mode the mod_plsql module should use to do extra performance logging.

InfoDebug mode: This logs more information to the Apache's error_log. This is used with Apache's info logging level. If the Apache's logging level is not at least set to this high, this setting will be ignored.

Category	Value
Syntax	`PlsqlInfoLogging` *InfoDebug*
Example	`PlsqlInfoLogging InfoDebug`
Default	Empty

The logging setting is useful for debugging problems in your PL/SQL application.

G.3.2.21 PlsqlMaxRequestsPerSession

Specifies the maximum number of requests a pooled database connection should service before it is closed and re-opened.

Category	Value
Syntax	`PlsqlMaxRequestsPerSession` *number*
Example	`PlsqlMaxRequestsPerSession 500`
Default	1000

- This parameter helps relieve memory and resource problems that may occur due to prolonged session reuse by a PL/SQL application.
- This parameter should not need to be changed. The default is sufficient in most cases.

- Setting this parameter to a low number can degrade performance. A case for a lower value might be an infrequently-used DAD whose performance is not a concern, and for which limiting the number of requests provides some benefit.

G.3.2.22 PlsqlNLSLanguage

Specifies the NLS_LANG variable for this DAD. This parameter overrides the NLS_LANG environment variable. When this parameter is set, the PL/SQL Gateway uses the specified NLS_LANG to connect to the database. Once connected, an alter session command is issued to switch to the specified language and territory. If the middle tier character set matches that of the database, then no alter session call is issued by the mod_plsql module.

Category	Value
Syntax	`PlsqlNLSLanguage` *`string`*
Example	`PlsqlNLSLanguage America_America.UTF8`
Default	None

- Most applications have PlsqlTransferMode set to CHAR which means that the character set in PlsqlNLSLanguage needs to match the character set of the database. In one special case, where the database and the mod_plsql module are both using fixed-size character sets, and the character set width matches, the character set can be different. The response character set is always the mod_plsql module character set.
- If PlsqlTransferMode is set to RAW, then this parameter can be ignored.

G.3.2.23 PlsqlPathAlias

Specifies a virtual path alias to map to a procedure call. This is application-specific. This directive is used with PlsqlPathAliasProcedure.

Category	Value
Syntax	`PlsqlPathAlias` *`string`*
Example	`PlsqlPathAlias url`
Default	None

For applications that do not use path aliasing, this parameter may be omitted.

G.3.2.24 PlsqlPathAliasProcedure

Specifies the procedure to call when the virtual path in the URL matches the path alias as configured by PlsqlPathAlias.

Category	Value
Syntax	`PlsqlPathAliasProcedure` *`string`*
Example	`PlsqlPathAliasProcedure portal.wwpth_api_alias.process_download`
Default	None

For applications that do not use path aliasing, this parameter may be omitted.

G.3.2.25 PlsqlRequestValidationFunction

Specifies an application-defined PL/SQL function which gives you the opportunity to allow and disallow further processing of the requested procedure. This is useful in implementing tight security for your PL/SQL application by blocking out package and procedure calls that should not be allowed to run from a DAD.

The function defined by this parameter must have the following prototype:

```
boolean function_name (procedure_name IN varchar2)
```

The *procedure_name* parameter will contain the name of the procedure that the request is trying to run.

For example, if all the PL/SQL application procedures callable from a browser are inside the package mypkg, then an implementation of this function can be as follows:

```
boolean my_validation_check (procedure_name varchar2)
is
begin
  if (upper (procedure_name) like upper ('myschema.mypkg%')) then
    return TRUE
  else
    return FALSE
  end if;
end;
```

Category	Value
Syntax	`PlsqlRequestValidationFunction` *string*
Example	`PlsqlRequestValidationFunction myschema.mypkg.my_validation_check`
Default	none

- By default, the mod_plsql module already disallows direct URL access to certain schemas and packages. For more information, refer to PlsqlExclusionList.
- It is highly recommended that you provide an implementation for this function such that it only allows requests that belong to your application, and are callable from a browser.
- Since this function will be called for every request, be sure to make this function as optimized as possible. Suggested recommendations are:
  - Name your PL/SQL packages in a fashion such that the implementation of this function can be similar to the previous example.
  - If your implementation performs a table lookup to determine what packages and procedures should be allowed, then performance can be improved if you pin the cursor in the shared pool.

G.3.2.26 PlsqlSessionCookieName

Specifies the cookie name when PlsqlAuthenticationMode is set to SingleSignOn. This parameter is supported only for Oracle Fusion Middleware releases, and is used by Oracle Portal and Oracle Single Sign-On.

Category	Value
Syntax	`PlsqlSessionCookieName` *`cookie_name`*
Example	`PlsqlSessionCookieName mycookie`
Default	Same as DAD name

- For DADs not using SingleSignOn authentication, this parameter can be omitted. In most other cases, the session cookie name should be omitted (and this parameter automatically defaults to the DAD name).
- A session cookie name must be specified only for Oracle Portal instances that need to participate in a distributed Oracle Portal environment. For those Oracle Portal nodes you want to seamless participate as a federated cluster, ensure that the session cookie name for all the participating nodes is the same.
- Independent Oracle Portal nodes need to use distinct session cookie names.

G.3.2.27 PlsqlSessionStateManagement

Specifies how package and session state should be cleaned up at the end of each the mod_plsql request.

- `StatelessWithResetPackageState` causes the mod_plsql module to call `dbms_session.reset_package_state` at the end of each mod_plsql request. This is the default.
- `StatelessWithPreservePackageState` causes the mod_plsql module to call `htp.init` at the end of each mod_plsql request. This cleans up the state of session variables in the PL/SQL Web ToolKit. The PL/SQL application is responsible for cleaning up its own session state. Failure to do so causes erratic behavior, in which a request starts recognizing or manipulating state modified in previous requests.
- `StatelessWithFastResetPackageState` causes the mod_plsql module to call `dbms_session.modify_package_state(dbms_session.reinitialize)` at the end of each mod_plsql request. This API is faster than the mode of `StatelessWithResetPackageState`, and avoids some latch contention issues, but exists only in Oracle database releases 8.1.7.2 and later. This mode uses slightly more memory than the default mode.

Category	Value
Syntax	`PlsqlSessionStateManagement {StatelessWithResetPackageState | StatelessWithFastResetPackageState | StatelessWithPreservePackageState}`
Example	`PlsqlSessionStateManagement StatelessWithPreservePackageState`
Default	`StatelessWithResetPackageState`

- The earlier values of `stateful=no` or `stateful=STATELESS_RESET` corresponds to `StatelessWithResetPackageState`.
- The earlier value of `stateful=STATELESS_FAST_RESET` corresponds to `StatelessWithFastResetPackageState`.
- The earlier value of `stateful=STATELESS_PRESERVE` corresponds to `StatelessWithPreservePackageState`.

The mod_plsql module does not support stateful mode of operation. To allow PL/SQL applications stateful behavior, save the state in cookies and/or in the database.

G.3.2.28 PlsqlTransferMode

Specifies the transfer mode for data from the database back to the mod_plsql module. Most applications use the default value of `CHAR`.

Category	Value
Syntax	`PlsqlTransferMode {CHAR | RAW}`
Example	`PlsqlTransferMode CHAR`
Default	`CHAR`

This parameter only must be changed to enable sending back responses in different character sets from the same DAD. In such a case, the `CHAR` mode is useless, since it always converts the response data from the database character set to the mod_plsql character set.

G.3.2.29 PlsqlUploadAsLongRaw

Specifies the file extensions to be uploaded as LONGRAW data type, as opposed to using the default BLOB data type. The default can be overridden by specifying multi-line directives of file extensions for field. A value of asterisk (`*`) in this field causes all documents to be uploaded as LONGRAW.

Category	Value
Syntax	`PlsqlUploadAsLongRaw` *`string multiline`*
Example	`PlsqlUploadAsLongRaw jpg` `PlsqlUploadAsLongRaw gif`
Default	None

For applications that do not upload or download documents, this parameter may be omitted.

G.3.3 cache.conf

The `cache.conf` file contains the configuration settings for the file system caching functionality implemented in the mod_plsql module. This configuration file is relevant only if PL/SQL applications use the `OWA_CACHE` package to cache dynamically generated content in the file system.

The following parameters are specified in the `cache.conf` file:

- PlsqlCacheCleanupTime
- PlsqlCacheDirectory
- PlsqlCacheEnable
- PlsqlCacheMaxAge
- PlsqlCacheMaxSize
- PlsqlCacheTotalSize

G.3.3.1 PlsqlCacheCleanupTime

Specifies the time to start the cleanup of the cache storage.

This setting defines the exact day and time in which cleanup should occur. The frequency can be set as daily, weekly, and monthly.

- To define daily frequency, the keyword `Everyday` is used. The cleanup starts every day at the time defined. For example, `Everyday 2:00` causes the cleanup to happen everyday at 2:00 a.m. (local time).
- To define weekly frequency, the days of the week, (`Sunday`, `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`) are used. For example, `Wednesday 15:30` causes the cleanup to happen every Wednesday at 3:30 p.m. (local time).
- To define monthly frequency, the keyword `Everymonth` is used. The cleanup starts on the Saturday of the month at the time defined. For example, Saturday Everymonth 23:00 causes the cleanup to happen the first Saturday of every month at 11:00 p.m. (local time).

Category	Value
Syntax	`PlsqlCacheCleanupTime {Sunday-Saturday | Everyday | Everymonth} {hh:mm}`
Example	`PlsqlCacheCleanupTime Monday 20:00`
Default	Saturday 23:00

G.3.3.2 PlsqlCacheDirectory

Specifies the directory where cache files are written out by the mod_plsql module. This directory must exist or Oracle HTTP Server will not start.

On UNIX, this directory must have write permissions by the owner of the child httpd processes.

Category	Value
Syntax	`PlsqlCacheDirectory` *directory*
Example	PlsqlCacheDirectory "${ORACLE_ INSTANCE}/servers/${COMPONENT_NAME}"
Default	none

G.3.3.3 PlsqlCacheEnable

Enables mod_plsql caching.

Category	Value
Syntax	`PlsqlCacheEnable {On | Off}`
Example	`PlsqlCacheEnable On`
Default	`Off`

If an application does not make use of the `OWA_CACHE` package in the PL/SQL Web Toolkit, then you can choose to disable caching. In such situations, there will be a minor performance benefit.

G.3.3.4 PlsqlCacheMaxAge

Specifies the maximum time, in days, a cache file can reside in a file system cache, after which the cached file will be removed for cache maintenance.

This setting is to ensure that the cache system does not contain old content. This setting removes old cache files and makes space for new ones.

Category	Value
Syntax	`PlsqlCacheMaxAge` *number*
Example	`PlsqlCacheMaxAge 20`
Default	30 (days)

G.3.3.5 PlsqlCacheMaxSize

Specifies the maximum possible size of a cache file.

This setting prevents the case in which one file can fill up the entire cache. In general, it is recommended that this be set to about 1-3 percent of the total cache size, which is specified by PlsqlCacheTotalSize.

Category	Value
Syntax	`PlsqlCacheMaxSize` *number*
Example	`PlsqlCacheMaxSize 1048576`
Default	1048576

G.3.3.6 PlsqlCacheTotalSize

Specifies the total size of the cache directory. The default is 20 MB.

This setting limits the amount of space the cache is allowed to use. Both PL/SQL cache and Session Cookie cache share this cache space. This setting is not a hard limit. It might exceed the limit temporarily during normal processing. This is normal behavior.

The cleanup algorithm uses this setting to determine how much to reduce the cache files. Therefore, the real space limit is the physical storage's available size.

This parameter takes bytes as values:

- 1 megabytes = 1048576 bytes
- 10 megabytes = 10485760 bytes

Category	Value
Syntax	`PlsqlCacheTotalSize` *number*
Example	`PlsqlCacheTotalSize 20971520`
Default	20971520 (bytes)

Glossary

Apache HTTP Server

Apache HTTP Server is an open source web server originally derived from the National Center for Supercomputing Applications (NCSA).

authentication

The process of verifying the identity of a user, device, or other entity in a host system, often as a prerequisite to granting access to resources in a system. A recipient of an authenticated message can be certain of the message's origin (its sender). Authentication is presumed to preclude the possibility that another party has impersonated the sender.

availability

The percentage or amount of scheduled time that a computing system provides application service.

certificate

Also called a **digital certificate**. An ITU x.509 v3 standard data structure that securely binds an identity to a public key.

A certificate is created when an entity's public key is signed by a trusted identity, a **certificate authority** The certificate ensures that the entity's information is correct and that the public key actually belongs to that entity.

A certificate contains the entity's name, identifying information, and public key. It is also likely to contain a serial number, expiration date, and information about the rights, uses, and privileges associated with the certificate. It also contains information about the certificate authority that issued it.

certificate authority

A trusted third party that certifies that other entities—users, databases, administrators, clients, servers—are who they say they are. When it certifies a user, the certificate authority first seeks verification that the user is not on the certificate revocation list (CRL), then verifies the user's identity and grants a certificate, signing it with the certificate authority's private key. The certificate authority has its own certificate and public key which it publishes. Servers and clients use these to verify signatures the certificate authority has made. A certificate authority might be an external company that offers certificate services, or an internal organization such as a corporate MIS department.

CGI

Common Gateway Interface (CGI) is the industry-standard technique for transferring information between a Web server and any program designed to accept and return data that conforms to the CGI specifications.

ciphertext

Data that has been encrypted. Ciphertext is unreadable until it has been converted to plain text (decrypted) with a key. See **decryption**.

cleartext

See **plaintext**.

cryptography

The art of protecting information by transforming it (encrypting) into an unreadable format. See **encryption**.

DAD

See **database access descriptor**.

database access descriptor

A database access descriptor (DAD) is a set of values that specify how an application connects to an Oracle database to fulfill an HTTP request. The information in the DAD includes the username (which also specifies the schema and the privileges), password, connect-string, error log file, standard error message, and national language support (NLS) parameters such as NLS language, NLS date format, NLS date language, and NLS currency.

decryption

The process of converting the contents of an encrypted message (**ciphertext**) back into its original readable format (**plaintext**).

digital certificate

See **certificate**.

digital wallet

See **wallet**.

encryption

The process of converting a message thereby rendering it unreadable to any but the intended recipient. Encryption is performed by converting data into code that cannot be understood by unauthorized people or systems. There are two main types of encryption: **public-key encryption** (also known as asymmetric-key encryption) and symmetric-key encryption.

entry

In the context of a directory service, entries are the building blocks of a directory. An entry is a collection of information about an object in the directory. Each entry is composed of a set of attributes that describe one particular trait of the object. For example, if a directory entry describes a person, that entry can have attributes such as first name, last name, telephone number, or e-mail address.

Execution Context ID

Execution Context ID (or ECID) is a unique identifier that can be used to correlate events in different components of Fusion Middleware or in different log files as being part of the same request execution flow.

failover

The ability to reconfigure a computing system to use an alternate active component when a similar component fails.

Fusion Middleware Control

See **Oracle Enterprise Manager Fusion Middleware Control**.

HTTP

See **Hypertext Transfer Protocol**.

Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) is the underlying format used by the Web to format and transmit messages and determine what actions Web servers and browsers should take in response to various commands. HTTP is the protocol used between Oracle Fusion Middleware and clients.

LDAP

See **Lightweight Directory Access Protocol**.

Lightweight Directory Access Protocol

A standard, extensible directory access protocol. It is a common language that LDAP clients and servers use to communicate. The framework of design conventions supporting industry-standard directory products, such as the Oracle Internet Directory.

modules

Modules extend the basic functionality of a Web server, and support integration between Oracle HTTP Server and other Oracle Fusion Middleware components.

Oracle Enterprise Manager Fusion Middleware Control

Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control) provides Web-based management tools designed specifically for Oracle Fusion Middleware. Using Fusion Middleware Control, you can monitor and configure the components of your application server, such as deploy applications, manage security, and create and manage Oracle Fusion Middleware clusters.

PEM

Privacy-enhanced Electronic Mail. An **encryption** technique that provides encryption, authentication, message integrity, and key management.

PL/SQL

PL/SQL is the Oracle proprietary extension to the SQL language. PL/SQL adds procedural and other constructs to SQL that make it suitable for writing applications.

plaintext

Also called cleartext. Unencrypted data in ASCII format.

plug-in

A module that adds a specific feature or service to a larger system.

port

A port is a number that TCP uses to route transmitted data to and from a particular program.

private key

In **public-key cryptography**, this key is the secret key. It is primarily used for decryption but is also used for encryption with digital signatures. See **public/private key pair**.

proxy server

A proxy server typically resides on a network firewall and allows clients behind the firewall to access Web resources. All requests from clients go to the proxy server rather than directly to the destination server. The proxy server forwards the request to the destination server and passes the received information back to the client. The proxy server channels all Web traffic at a site through a single, secure port; this allows an organization to create a secure firewall by preventing Internet access to internal systems, while allowing Web access.

public key

In **public-key cryptography**, this key is made public to all. It is primarily used for encryption but can be used for verifying signatures. See **public/private key pair**.

public-key cryptography

Encryption method that uses two different random numbers (keys). See **public key** and **public-key encryption**.

public-key encryption

The process where the sender of a message encrypts the message with the public key of the recipient. Upon delivery, the message is decrypted by the recipient using its private key.

public/private key pair

A set of two numbers used for **encryption** and **decryption**, where one is called the **private key** and the other is called the **public key**. Public keys are typically made widely available, while private keys are held by their respective owners. Though mathematically related, it is generally viewed as computationally infeasible to derive the private key from the public key. Public and private keys are used only with asymmetric encryption algorithms, also called **public-key encryption** algorithms, or public-key cryptosystems. Data encrypted with either a public key or a private key from a key pair can be decrypted with its associated key from the key-pair. However, data encrypted with a public key cannot be decrypted with the same public key, and data encrypted with a private key cannot be decrypted with the same private key.

RSA

A **public-key encryption** technology developed by RSA Data Security. The RSA algorithm is based on the fact that it is laborious to factor very large numbers. This makes it mathematically unfeasible, because of the computing power and time required to decode an RSA key.

scalability

A measure of how well the software or hardware product is able to adapt to future business needs.

Secure Sockets Layer

Secure Sockets Layer (SSL) is a standard for the secure transmission of documents over the Internet using HTTPS (secure HTTP). SSL uses digital signatures to ensure that transmitted data is not tampered with.

single sign-on

Single sign-on enables a you to authenticate once, combined with strong authentication occurring transparently in subsequent connections to other databases or applications. It lets you access multiple accounts and applications with a single password, entered during a single connection.

SSL

See **Secure Sockets Layer**.

wallet

Also called a digital wallet. A wallet is a data structure used to store and manage security credentials for an individual entity. It implements the storage and retrieval of credentials for use with various cryptographic services. A **Wallet Resource Locator** (WRL) provides the necessary information to locate the wallet.

Wallet Resource Locator

A wallet resource locator (WRL) provides all necessary information to locate a wallet. It is a path to an operating system directory that contains a wallet.

WRL

See **Wallet Resource Locator**.

X.509

A standard for creating digital certificates.

Index

I

L

M

N

O

P

R

S

T

U

W

X