Oracle® Fusion Middleware

Administering Oracle HTTP Server

12c (12.2.1.4.0)
E96370-11
November 2023

Oracle Fusion Middleware Administering Oracle HTTP Server, 12c (12.2.1.4.0)

E96370-11

Primary Author: Oracle Corporation

Contents

5 Working with Oracle HTTP Server

6 Configuring High Availability for Web Tier Components

9 Managing Oracle HTTP Server Logs

10 Managing Application Security

A Oracle HTTP Server WLST Custom Commands

E Troubleshooting Oracle HTTP Server

F Configuration Files

G Property Files

H Oracle HTTP Server Module Directives

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.

Accessible Access to Oracle Support

Oracle customers who 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.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and

partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documents

See the following documents in the Oracle Fusion Middleware documentation set:

- *Understanding Oracle Fusion Middleware*
- *Administering Oracle Fusion Middleware*
- *Tuning Performance*
- *High Availability Guide*
- *Using Oracle WebLogic Server Proxy Plug-Ins*
- Apache documentation included in this library. See: `http://httpd.apache.org/docs/2.4/`

> **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.

Part I
Understanding Oracle HTTP Server

Oracle HTTP Server is the web server component for Oracle Fusion Middleware. It includes several Oracle-provided and third-party modules to extend its basic functionality. It also includes Apache HTTP Server.

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

- Introduction to Oracle HTTP Server
- Understanding Oracle HTTP Server Modules
- Understanding Oracle HTTP Server Management Tools

- Introduction to Oracle HTTP Server
  Oracle HTTP Server is the web server component for Oracle Fusion Middleware, and provides a listener for Oracle WebLogic Server and the framework for hosting static pages, dynamic pages, and applications over the web.
- Understanding Oracle HTTP Server Modules
  Modules extend the basic functionality of Oracle HTTP Server and support integration between Oracle HTTP Server and other Oracle Fusion Middleware components. Oracle HTTP Server uses both Oracle developed modules or “plug-ins” and Apache and third party-developed modules.
- Understanding Oracle HTTP Server Management Tools
  Oracle HTTP Server can be managed using tools such as the Configuration Wizard, Fusion Middleware Control, and WebLogic Scripting tool.

1

Introduction to Oracle HTTP Server

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

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

This chapter includes the following sections:

- What is Oracle HTTP Server?

- Oracle HTTP Server 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 other Fusion Middleware components. 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-Ins to route dynamic content requests to Managed Server instances.

- Key Features of Oracle HTTP Server

  Oracle HTTP Server includes a web server proxy plug-in for Oracle WebLogic Server, components for boosting web application performance, an installation mode that does not require a database connection, multiple security configuration options, and more.

- Domain Types

  You can install Oracle HTTP Server on two types of domains: *WebLogic Server domain* and *standalone domain*. In the *WebLogic Server domain*, Oracle HTTP Server can be collocated with Oracle WebLogic Server in full or Restricted-JRF mode. Standalone domain has restricted functionality.

- Understanding Oracle HTTP Server Directory Structure

  When Oracle HTTP Server is installed in a domain, a directory tree is created that contains the files that are required by Oracle HTTP server to support that domain type.

- Understanding Configuration Files
  Oracle HTTP Server contains several configuration files that are similar to those used in Apache HTTP Server. Most of these files end with the `.conf` file type.
- Upgrading from Earlier Releases of Oracle HTTP Server
  You can use the Upgrade Assistant to upgrade and configure supported Fusion Middleware and Oracle HTTP Server domains from an earlier release to 12c (12.2.1.4.0) and perform a readiness check prior to an upgrade.
- Oracle HTTP Server Support
  Oracle provides technical support for Oracle HTTP Server features.

What is Oracle HTTP Server?

Oracle HTTP Server is a web server based on Apache HTTP Server infrastructure and includes additional modules developed specifically by Oracle. Oracle HTTP Server can also be a proxy server. 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, which allows Oracle HTTP Server to be set up as a reverse proxy through the fcgi protocol to a persistent Perl runtime environment using mod_proxy_fcgi.

  Although Oracle HTTP Server contains a Perl interpreter, it is internal to the product. You cannot use this interpreter for hosting Perl under a FastCGI environment. You must provide your own Perl environment.
- Oracle WebLogic Server Proxy Plug-In, which enables Oracle HTTP Server to front-end WebLogic Servers and other Fusion Middleware-based applications.

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

- Perl (through mod_proxy_fcgi, CGI and FastCGI)
- C and C++ (through mod_proxy_fcgi, CGI and FastCGI)
- Java, Ruby and Python (through mod_proxy_fcgi, CGI and FastCGI)

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 Oracle Fusion Middleware concepts, see *Understanding Oracle Fusion Middleware*.

Accessibility Tips for Oracle HTTP Server

Oracle HTTP Server can be managed using the Oracle Fusion Middlware Control in collocated mode. Oracle HTTP Server uses Fusion Middleware Control as its graphical user interface.

See Accessibility Features and Tips for Fusion Middleware Control in *Accessibility Guide*.

Oracle HTTP Server 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 other Fusion Middleware components. 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-Ins to route dynamic content requests to Managed Server instances.

There are multiple ways of implementing Oracle HTTP Server, depending on your requirements. Table 1-1 describes the major implementations, or "topologies."

Table 1-1 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 1-1. To obtain this topology, install Oracle HTTP Server in standalone mode. Can be paired with Oracle HTTP Server Collocated mode by using the Pack or UnPack commands.	See Standard Installation Topology for Oracle HTTP Server in a Standalone Domain in *Installing and Configuring Oracle HTTP Server*.
Standard Installation Topology for Oracle HTTP Server in a WebLogic Server Domain (Restricted-JRF)	This topology is similar to the Full-JRF (Java Required Files) topology, except that it does not require a backing database. The Restricted-JRF mode offers all of the functionality as the Full-JRF mode, except cross component wiring is not available. To obtain this topology, install Oracle HTTP Server in Collocated mode, then choose the Oracle HTTP Server Restricted-JRF domain template for provisioning this domain. This topology handles most use cases except for cross-component wiring.	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 WebLogic Server Domain (Full-JRF)	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 1-2. To obtain this topology, install Oracle HTTP Server in Collocated mode, then choose the Oracle HTTP Server Full-JRF domain template. Note that this topology, requires a database in back-end and can support cross-component wiring.	See Standard Installation Topology for Oracle HTTP Server in a WebLogic Server Domain in *Installing and Configuring Oracle HTTP Server*.

Figure 1-1 illustrates the standard installation topology for Oracle HTTP Server in a standalone domain.

Figure 1-1 Standard Installation Topology for Oracle HTTP Server in a Standalone Domain

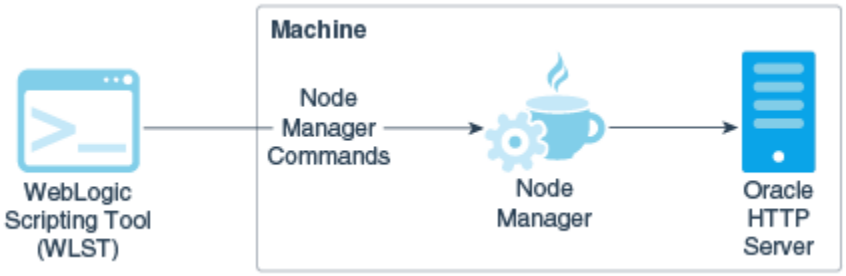
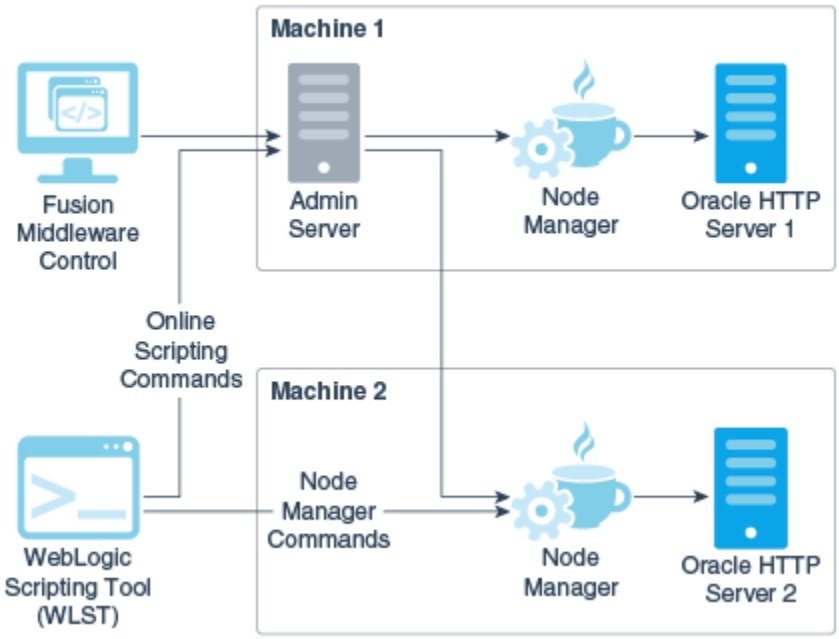


Figure 1-2 illustrates the standard installation topology for Oracle HTTP Server in a WebLogic Server domain.

Figure 1-2 Standard Installation Topology for Oracle HTTP Server in a WebLogic Server Domain



Key Features of Oracle HTTP Server

Oracle HTTP Server includes a web server proxy plug-in for Oracle WebLogic Server, components for boosting web application performance, an installation mode that does not require a database connection, multiple security configuration options, and more.

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

- Restricted-JRF Mode
- Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)
- CGI and Fast CGI Protocol (mod_proxy_fcgi)
- Security Features
- URL Rewriting and Proxy Server Capabilities

- Restricted-JRF Mode
- Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)
- CGI and Fast CGI Protocol (mod_proxy_fcgi)
- Security Features
- URL Rewriting and Proxy Server Capabilities

Restricted-JRF Mode

To install Oracle HTTP Server in a Oracle WebLogic Server domain in the Restricted-JRF mode, then a connection to an external database is not required. All of the Oracle HTTP

Server functionality through Fusion MiddleWare Control and WLST described in this documentation is still available, with the exception of cross component wiring.

Lack of support for cross component wiring means that:

- There are changes to the Oracle Fusion Middleware Control menu options. Some of the menu options which support cross component wiring are removed or disabled.
- Any database dependencies are completely removed.

> **See Also:**
>
> Wiring Components to Work Together in *Administering Oracle Fusion Middleware*.

The management of keys and certificates for an Oracle HTTP Server instance in a Restricted-JRF domain continue to be keystore services (KSS). In a Restricted-JRF domain, the database persistency of KSS is replaced with file persistency. To an end user, there are no visible change in basic KSS APIs to manage keys or certificates.

Oracle HTTP Server continues to support multiple Oracle wallets for complex virtual server configurations both in Restricted-JRF and full JRF mode.

Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)

The Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs) enables requests to be proxied from Oracle HTTP Server to Oracle WebLogic Server. This plug-in enhances an Oracle HTTP server installation by allowing Oracle WebLogic Server to handle requests that require dynamic functionality. In other words, you typically use a plug-in where the HTTP server serves static pages such as HTML pages, while Oracle WebLogic Server serves the J2EE dynamic pages such as Servlets, Java Server Pages (JSPs), and Enterprise Java Bean (EJB).

See Configuring the Plug-In for Oracle HTTP Server.

CGI and Fast CGI Protocol (mod_proxy_fcgi)

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 by using the mod_proxy_fcgi module.

The mod_proxy_fcgi module is the Oracle replacement for the deprecated mod_fastcgi module. The mod_proxy_fcgi module requires the service of the mod_proxy module and provides support for the FastCGI protocol.

For information on configuring the mod_proxy_fcgi module, see About Configuring mod_proxy_fcgi. For information on migrating from the mod_fastcgi module to mod_proxy_fcgi, see Migrating to the mod_proxy_fcgi and mod_authnz_fcgi Modules .

Security Features

Oracle HTTP Server employs many security features. Key among them are:

- Oracle Secure Sockets Layer (mod_ossl)
- Security: Encryption with Secure Sockets Layer
- Security: Single Sign-On with WebGate

Oracle Secure Sockets Layer (mod_ossl)

The mod_ossl module, the Oracle Secure Sockets Layer (SSL) implementation used in 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. The mod_ossl module supports TLS version 1.0, 1.1 , and 1.2.

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*

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*

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.

Domain Types

You can install Oracle HTTP Server on two types of domains: *WebLogic Server domain* and *standalone domain*. In the *WebLogic Server domain*, Oracle HTTP Server can be collocated with Oracle WebLogic Server in full or Restricted-JRF mode. Standalone domain has restricted functionality.

You can select which environment you want to use during server configuration.

- WebLogic Server Domain (Full-JRF Mode)
- WebLogic Server Domain (Restricted-JRF Mode)
- Standalone Domain

- WebLogic Server Domain (Full-JRF Mode)
- WebLogic Server Domain (Restricted-JRF Mode)
- 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.

WebLogic Server Domain (Full-JRF Mode)

A WebLogic Server Domain in Full-JRF mode 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, which requires installing a complete Oracle Fusion Middleware infrastructure before you install Oracle HTTP Server.

- For more information about installing a WebLogic Server Domain, see *Installing and Configuring the Oracle Fusion Middleware Infrastructure*.
- For information about installing Oracle HTTP Server either as part of a Oracle Fusion Middleware infrastructure or as standalone component, see *Installing and Configuring Oracle HTTP Server*.

WebLogic Server Domain (Restricted-JRF Mode)

The Weblogic Server Domain in Restricted-JRF mode is similar in architecture and functionality to Weblogic Server Domain in Full mode, except it does not define a connection to an external database. There are no database dependencies in Restricted-JRF mode.

This lack of a backing database means that cross component wiring is not supported by Oracle HTTP Server in a Restricted-JRF domain; this is the major differentiating factor between a Full JRF- and a Restricted-JRF-based domain.

Like the Full -JRF domain, the management of keys and certificates of an Oracle HTTP Server instance in a Restricted-JRF domain continues to be keystore service (KSS). In a Restricted-JRF domain, the database persistency of KSS is replaced with file persistency, although to an end user there is no visible change in basic KSS APIs to manage keys and certificates.

Like the Full -JRF domain, Oracle HTTP Server in a Restricted-JRF domain supports multiple Oracle wallets for complex virtual server configurations.

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 the following tools:

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

- Configuration Wizard
- Pack or Unpack

Generally, you would use a standalone domain when you do not want your Oracle HTTP Server implementation installed with a WebLogic Server domain and do not need the management functionality provided by Oracle 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 Node Manager.

Understanding Oracle HTTP Server Directory Structure

When Oracle HTTP Server is installed in a domain, a directory tree is created that contains the files that are required by Oracle HTTP server to support that domain type.

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 Understanding the Directory Structures in *Installing and Configuring Oracle HTTP Server*.

Understanding Configuration Files

Oracle HTTP Server contains several configuration files that are similar to those used in Apache HTTP Server. Most of these files end with the `.conf` file type.

The following topics explain the layout of the configuration file directories, mechanisms for editing the files, and more about the files themselves.

- Staging and Run-time Configuration Directories
- Oracle HTTP Server Configuration Files
- Modifying an Oracle HTTP Server Configuration File
- Staging and Run-time Configuration Directories
- Oracle HTTP Server Configuration Files
- Modifying an Oracle HTTP Server Configuration File

Staging and Run-time Configuration Directories

Two configuration directories are associated with each Oracle HTTP Server instance: a staging directory and a run-time directory.

- 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 contain the complete Oracle HTTP Server configuration -- httpd.conf, admin.conf, auditconfig.xml, and so on.

Modifications to the configuration are made in the staging directory. These modifications are automatically propagated to the run-time directory during the following operations:

> **Note:**
>
> Before making any changes to the files in the staging directory manually (that is, without using Fusion Middleware Control or WLST), stop the Administration Server.

- 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 Oracle HTTP Server 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, the user 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.

Oracle HTTP Server Configuration Files

The default Oracle HTTP Server configuration contains the files described in 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/2.4/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.

Modifying an Oracle HTTP Server Configuration File

For instances that are part of a WebLogic Server Domain, Fusion Middleware Control and the management infrastructure manages the Oracle HTTP Server configuration. 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 stopped. When the administration server is subsequently started (with start or restart), the results of any manual edits will be replicated to the run-time directory on the node of the managed instance. See About Editing Configuration Files.

> **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.

Upgrading from Earlier Releases of Oracle HTTP Server

You can use the Upgrade Assistant to upgrade and configure supported Fusion Middleware and Oracle HTTP Server domains from an earlier release to 12c (12.2.1.4.0) and perform a readiness check prior to an upgrade.

To upgrade Oracle HTTP Server, see *Upgrading with the Upgrade Assistant*.

Oracle HTTP Server Support

Oracle provides technical support for Oracle HTTP Server features.

The following Oracle HTTP Server features and conditions are supported:

- 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 non-Oracle-provided modules are suspect of 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.

2

Understanding Oracle HTTP Server Modules

Modules extend the basic functionality of Oracle HTTP Server and support integration between Oracle HTTP Server and other Oracle Fusion Middleware components. Oracle HTTP Server uses both Oracle developed modules or “plug-ins” and Apache and third party-developed modules.

This chapter includes the following sections:

Oracle-Developed Modules for Oracle HTTP Server

Oracle has developed modules that Oracle HTTP Server can use specifically to extend its basic functionality.

The following sections describe these modules:

- mod_certheaders Module—Enables Reverse Proxies
- mod_context Module—Creates or Propagates ECIDs
- mod_dms Module—Enables Access to DMS Data
- mod_odl Module—Enables Access to ODL
- mod_ora_audit—Supports Authentication and Authorization Auditing
- mod_ossl Module—Enables Cryptography (SSL)
- mod_webgate Module—Enables Single Sign-on
- mod_wl_ohs Module—Proxies Requests to Oracle WebLogic Server

mod_certheaders Module—Enables Reverse Proxies

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 mod_certheaders Module for a list and description of the directives accepted by mod_certheaders.

mod_context Module—Creates or Propagates ECIDs

The mod_context module creates or propagates Execution Context IDs, 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 enables loading ECIDs 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.

mod_dms Module—Enables Access to DMS Data

The mod_dms module provides FMW infrastructure access to the Oracle HTTP Server Dynamic Monitoring Service (DMS) data.

> **See Also:**
>
> Oracle Dynamic Monitoring Service in *Tuning Performance*.

mod_odl Module—Enables Access to 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.

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 Managing Oracle HTTP Server Logs.

> **See Also:**
>
> Managing Log Files and Diagnostic Datain *Administering Oracle Fusion Middleware*.

mod_ora_audit—Supports Authentication and Authorization Auditing

This module provides the OraAuditEnable directive to support authentication and authorization auditing by using the FMW Common Audit Framework. Previously the code for Audit was integrated in Oracle HTTP Server binary itself. In the current release, this is provided as a separate loadable module. See Support for FMW Audit Framework.

mod_ossl Module—Enables Cryptography (SSL)

The `mod_ossl` module enables strong cryptography for Oracle HTTP Server. It is a plug-in to Oracle HTTP Server that enables the server to use SSL. The functionality of this module is similar to the functionality of Apache's `mod_ssl` module. However, the cryptography engine used in the `mod_ossl` module differs from that of the `mod_ssl` module. The `mod_ossl` module uses Oracle's Secure Socket Layer, which is based on RSA security technology, whereas the `mod_ssl` module relies on OpenSSL to provide the cryptography engine.

> **Note:**
>
> Oracle HTTP server distributes OpenSSL libraries for usage with `mod_security2` module. As stated above, the `mod_ossl` module does not use OpenSSL libraries.

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. See SSLFIPS Directive.

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

The `mod_ossl` modules provides these features:

- 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 the `mod_ssl` module.

mod_ossl Module Directives

See mod_ossl Module for a list and descriptions of directives accepted by the `mod_ossl` module.

> **Note:**
>
> See Configuring SSL for the Web Tier in *Administering Oracle Fusion Middleware*.

mod_webgate Module—Enables Single Sign-on

The `mod_webgate` module is included with Oracle HTTP Server to enable single sign-on features from Oracle Access Manager (OAM). OAM's WebGate feature examines incoming requests and determines whether the requested resource is protected, and if so, retrieves the session information for the user. See Authenticating Users with WebGate and Security: Single Sign-On with WebGate.

`mod_webgate` is generally integrated with `mod_ossl` and `mod_wl_ohs`, and has a dependency on cURL and OpenSSL libraries. These libraries are also included in the Oracle HTTP Server installation. For information about configuring WebGate, see Configuring WebGate for Oracle Access Manager in *Installing and Configuring Oracle HTTP Server*.

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

mod_wl_ohs Module—Proxies Requests to Oracle WebLogic Server

The mod_wl_ohs module is a key feature of Oracle HTTP Server that enables requests to be proxied from Oracle HTTP Server to Oracle WebLogic Server. This

module is generally referred to as the Oracle WebLogic Server Proxy Plug-In. This plug-in enhances an Oracle HTTP server installation by allowing Oracle WebLogic Server to handle requests that require dynamic functionality. In other words, you typically use a plug-in where the HTTP server serves static pages such as HTML pages, while Oracle WebLogic Server serves dynamic pages such as HTTP Servlets and Java Server Pages (JSPs).

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.* Directives for this module are listed in Parameters for Oracle WebLogic Server Proxy Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins.*

> **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.

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

Oracle HTTP Server includes Apache and third-party modules. These modules are not developed by Oracle.

Table 2-1 lists these modules.

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

Module	Enabled by Default?	For more information, see:
mod_access_compat	No	http://httpd.apache.org/docs/2.4/mod/mod_access_compat.html
mod_actions	Yes	http://httpd.apache.org/docs/2.4/mod/mod_actions.html
mod_alias	Yes	http://httpd.apache.org/docs/2.4/mod/mod_alias.html
mod_asis	Yes	http://httpd.apache.org/docs/2.4/mod/mod_asis.html
mod_auth_basic	Yes	http://httpd.apache.org/docs/2.4/mod/mod_auth_basic.html
mod_authn_anon	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authn_anon.html
mod_authn_core	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authn_core.html
mod_authn_file	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authn_file.html
mod_authz_core	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authz_core.html

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

Module	Enabled by Default?	For more information, see:
mod_authnz_fcgi	No	http://httpd.apache.org/docs/2.4/mod/mod_authnz_fcgi.html
mod_authz_groupfile	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authz_groupfile.html
mod_authz_host	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authz_host.html
mod_authz_owner	No	http://httpd.apache.org/docs/2.4/mod/mod_authz_owner.html
mod_authz_user	Yes	http://httpd.apache.org/docs/2.4/mod/mod_authz_user.html
mod_autoindex	Yes	http://httpd.apache.org/docs/2.4/mod/mod_autoindex.html
mod_cache (Windows only)	No	http://httpd.apache.org/docs/2.4/mod/mod_cache.html
mod_cache_disk	No	http://httpd.apache.org/docs/2.4/mod/mod_cache_disk.html
mod_disk_cache (Windows only)	No	http://httpd.apache.org/docs/2.2/mod/mod_disk_cache.html
mod_cern_meta	Yes	http://httpd.apache.org/docs/2.4/mod/mod_cern_meta.html
mod_cgi	Yes	http://httpd.apache.org/docs/2.4/mod/mod_cgi.html
mod_cgid (UNIX only)	Yes	http://httpd.apache.org/docs/2.4/mod/mod_cgid.html
mod_deflate	No	http://httpd.apache.org/docs/2.4/mod/mod_deflate.html **Note:** To enable `mod_deflate`, you must first upload `mod_filter`. In Apache HTTP Server Version 2.4, the command *AddOutputFilterByType* directive is moved to `mod_filter` module. See https://httpd.apache.org/docs/current/upgrading.html#commonproblems.
mod_dir	Yes	http://httpd.apache.org/docs/2.4/mod/mod_dir.html
mod_dumpio	No	http://httpd.apache.org/docs/2.4/mod/mod_dumpio.html
mod_env	Yes	http://httpd.apache.org/docs/2.4/mod/mod_env.html
mod_expires	Yes	http://httpd.apache.org/docs/2.4/mod/mod_expires.html
mod_file_cache	Yes	http://httpd.apache.org/docs/2.4/mod/mod_file_cache.html
mod_filter	No	http://httpd.apache.org/docs/2.4/mod/mod_filter.html **Note:** The syntax of the `FilterProvider` directive under `mod_filter` has changed in Apache 2.4. This directive must be upgraded manually. See http://httpd.apache.org/docs/2.4/upgrading.html

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

Module	Enabled by Default?	For more information, see:
mod_headers	Yes	http://httpd.apache.org/docs/2.4/mod/mod_headers.html
mod_imagemap	Yes	http://httpd.apache.org/docs/2.4/mod/mod_imagemap.html
mod_include	Yes	http://httpd.apache.org/docs/2.4/mod/mod_include.html
mod_info	Yes	http://httpd.apache.org/docs/2.4/mod/mod_info.html
mod_lbmethod_bybusyness	No	http://httpd.apache.org/docs/2.4/mod/mod_lbmethod_bybusyness.html
mod_lbmethod_byrequests	No	http://httpd.apache.org/docs/2.4/mod/mod_lbmethod_byrequests.html
mod_lbmethod_bytraffic	No	http://httpd.apache.org/docs/2.4/mod/mod_lbmethod_bytraffic.html
mod_log_config	Yes	http://httpd.apache.org/docs/2.4/mod/mod_log_config.html
mod_log_forensic	Yes	http://httpd.apache.org/docs/2.4/mod/mod_log_forensic.html
mod_logio	No	http://httpd.apache.org/docs/2.4/mod/mod_logio.html
mod_macro	No	http://httpd.apache.org/docs/2.4/mod/mod_macro.html
mod_mime	Yes	http://httpd.apache.org/docs/2.4/mod/mod_mime.html
mod_mime_magic	Yes	http://httpd.apache.org/docs/2.4/mod/mod_mime_magic.html
mod_mpm_event	Yes (Linux only)	http://httpd.apache.org/docs/2.4/mod/event.html
mod_mpm_prefork	No	http://httpd.apache.org/docs/2.4/mod/prefork.html
mod_mpm_winnt (Windows only)	Yes	http://httpd.apache.org/docs/2.4/mod/mpm_winnt.html
mod_mpm_worker	Yes (on Non-Windows and non-Linux platforms)	http://httpd.apache.org/docs/2.4/mod/worker.html
mod_negotiation	Yes	http://httpd.apache.org/docs/2.4/mod/mod_negotiation.html
mod_proxy	Yes	http://httpd.apache.org/docs/2.4/mod/mod_proxy.html
mod_proxy_balancer	Yes	http://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html
mod_proxy_connect	Yes	http://httpd.apache.org/docs/2.4/mod/mod_proxy_connect.html
mod_proxy_fcgi	No	http://httpd.apache.org/docs/2.4/mod/mod_proxy_fcgi.html

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

Module	Enabled by Default?	For more information, see:
mod_proxy_ftp	Yes	http://httpd.apache.org/docs/2.4/mod/mod_proxy_ftp.html
mod_proxy_http	Yes	http://httpd.apache.org/docs/2.4/mod/mod_proxy_http.html
mod_remoteip	No	http://httpd.apache.org/docs/2.4/mod/mod_remoteip.html
mod_reqtimeout	No	http://httpd.apache.org/docs/2.4/mod/mod_reqtimeout.html
mod_rewrite	Yes	http://httpd.apache.org/docs/2.4/mod/mod_rewrite.html
mod_security2	No	http://www.modsecurity.org/documentation.html Also, for Oracle HTTP Server-specific information regarding mod_security, see Configuring mod_security in the httpd.conf File..
mod_sed	No	http://httpd.apache.org/docs/2.4/mod/mod_sed.html
mod_setenvif	Yes	http://httpd.apache.org/docs/2.4/mod/mod_setenvif.html
mod_slotmem_shm	Yes	http://httpd.apache.org/docs/2.4/mod/mod_slotmem_shm.html
mod_socache_shmcb	Yes	http://httpd.apache.org/docs/2.4/mod/mod_socache_shmcb.html
mod_speling	Yes	http://httpd.apache.org/docs/2.4/mod/mod_speling.html
mod_status	Yes	http://httpd.apache.org/docs/2.4/mod/mod_status.html
mod_substitute	No	http://httpd.apache.org/docs/2.4/mod/mod_substitute.html
mod_unique_id	Yes	http://httpd.apache.org/docs/2.4/mod/mod_unique_id.html
mod_unixd	Yes	http://httpd.apache.org/docs/2.4/mod/mod_unixd.html
mod_userdir	Yes	http://httpd.apache.org/docs/2.4/mod/mod_userdir.html
mod_usertrack	Yes	http://httpd.apache.org/docs/2.4/mod/mod_usertrack.html
mod_version	Yes	http://httpd.apache.org/docs/2.4/mod/mod_version.html
mod_vhost_alias	Yes	http://httpd.apache.org/docs/2.4/mod/mod_vhost_alias.html
mod_proxy_wstunnel	No	http://httpd.apache.org/docs/2.4/mod/mod_proxy_wstunnel.html

3

Understanding Oracle HTTP Server Management Tools

Oracle HTTP Server can be managed using tools such as the Configuration Wizard, Fusion Middleware Control, and WebLogic Scripting tool.

The following sections describe the management tools, how to access Fusion Middleware Control and the Oracle HTTP Server home page, and how to use the WebLogic Scripting Tool (WLST)

- Configuration Wizard, which enables you to create and delete Oracle HTTP Server instances. See *Installing and Configuring Oracle HTTP Server*.
- Fusion Middleware Control, which is a browser-based management tool. See *Administering Oracle Fusion Middleware*.
- WebLogic Scripting Tool, which is a command-driven scripting tool. 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. See Domain Types.
> - 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.

This chapter includes the following sections:

- Administering Oracle HTTP Server Using Fusion Middleware Control
- Administering Oracle HTTP Server Using WLST
- Administering Oracle HTTP Server Using Fusion Middleware Control
  Fusion Middleware Control is the main tool for managing Oracle HTTP Server. This tool is browser-based and helps to administer and monitor the Oracle Fusion Middleware environment.
- Administering Oracle HTTP Server Using WLST
  The WebLogic Scripting Tool (WLST) is a command-driven scripting tool that provides specific commands to manage Oracle HTTP Server.

Administering Oracle HTTP Server Using Fusion Middleware Control

Fusion Middleware Control is the main tool for managing Oracle HTTP Server. This tool is browser-based and helps to administer and monitor the Oracle Fusion Middleware environment.

The following sections describe some of the basic Oracle HTTP Server administration tasks you can perform with Fusion Middleware Control.

- Accessing Fusion Middleware Control
- Accessing the Oracle HTTP Server Home Page
- About the Oracle HTTP Server Home Page
- Editing Configuration Files Using Fusion Middleware Control

> **See Also:**
>
> *Administering Oracle Fusion Middleware*

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.

Accessing the Oracle HTTP Server Home Page

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 the target's menu is displayed at the top of the page, in the context pane.

To display the Oracle HTTP Server home page and the server menu, select an Oracle HTTP Server component from the HTTP Server folder. You can also display the Oracle HTTP Server menu by right-clicking the Oracle HTTP Server target in the navigation pane.

About the Oracle HTTP Server Home Page describes the target navigation pane and the home page of Oracle HTTP Server.

About 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.

The Oracle HTTP Server home page contains the following regions:

- **General Region**: Shows the name of the component, its state, host, port, and machine name, and the location of the Oracle Home.
- **Key Statistics Region**: Shows the processes and requests statistics.
- **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.

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

Editing Configuration Files Using Fusion Middleware Control

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. See 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. For instructions on how to edit a configuration file from Fusion Middleware Control, see Editing a Configuration File for a WebLogic Server Domain.

Administering Oracle HTTP Server Using WLST

The WebLogic Scripting Tool (WLST) is a command-driven scripting tool that provides specific commands to manage Oracle HTTP Server.

This section contains information on WLST commands and how to use WLST in a standalone environment.

- Oracle HTTP Server-Specific WLST Commands
- Using WLST in a Standalone Environment

For detailed information on WLST, see *Understanding the WebLogic Scripting Tool*

For more information on the WLST custom commands that are available for Oracle HTTP Server, see Oracle HTTP Server WLST Custom Commands.

- Oracle HTTP Server-Specific WLST Commands
- Using WLST in a Standalone Environment

Oracle HTTP Server-Specific WLST Commands

WLST provides Oracle HTTP Server-specific commands for server management in WebLogic Server Domains. See Oracle HTTP Server WLST Custom Commands.

The following are online commands, which require a connection between WLST and the administration server for the domain.

- `ohs_createInstance`
- `ohs_deleteInstance`
- `ohs_addAdminProperties`
- `ohs_addNMProperties`
- `ohs_exportKeyStore`
- `ohs_updateInstances`

Oracle recommends that you use the `ohs_createInstance` and `ohs_deleteInstance` commands to create and delete Oracle HTTP Server instances instead of using the Configuration Wizard. These commands perform additional error checking and, in the case of instance creation, automatic port assignment.

Using WLST in a Standalone Environment

If your Oracle HTTP Server instance is running in a standalone environment, you can use WLST but must use the offline, or "agent", commands that route tasks through. The specific WLST commands are described in Running Oracle HTTP Server, in the context of the task they perform (for example, the WLST command for starting a standalone Oracle HTTP Server instance is documented in Starting Oracle HTTP Server Instances Using WLST); however, you must use the `nmConnect()` command to actually connect to offline WLST. For both Linux and Windows, the format of the command is the same:

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

For example:

```
nmConnect('weblogic','<yourpassword>','localhost','5556','myDomain')
```

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.

Part II
Managing Oracle HTTP Server

There are many management tasks to consider when running Oracle HTTP Server. These tasks include managing and monitoring the server processes, application security, connectivity, and more.

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

- Running Oracle HTTP Server
- Working with Oracle HTTP Server
- Managing and Monitoring Server Processes
- Managing Connectivity
- Managing Oracle HTTP Server Logs
- Managing Application Security
- Running Oracle HTTP Server
  To run Oracle HTTP Server, create and manage an Oracle HTTP Server instance in a WebLogic or standalone environment.
- Working with Oracle HTTP Server
  When working with an installed version of Oracle HTTP Server, there are some common tasks that you have to perform, such as editing configuration files, specifying server properties, and more.
- Configuring High Availability for Web Tier Components
  Use the instructions in this chapter to configure an Oracle HTTP Server highly available deployment in which Oracle HTTP Servers and WebLogic Managed Servers reside on different hosts, behind a load balancer.
- Managing and Monitoring Server Processes
  You have tools and procedures that help to manage and monitor the performance of Oracle HTTP Server.
- Managing Connectivity
  You can manage and monitor the performance of Oracle HTTP Server connectivity by creating ports, viewing port number usage, and configuring virtual hosts.
- Managing Oracle HTTP Server Logs
  Managing Oracle HTTP Server logs includes configuring the server logs, viewing the cause of an error and its corrective action, and more.
- Managing Application Security
  Oracle HTTP Server supports three main categories of security, namely, authentication, authorization, and confidentiality.

4

Running Oracle HTTP Server

To run Oracle HTTP Server, create and manage an Oracle HTTP Server instance in a WebLogic or standalone environment.

This chapter describes how to create an instance, perform basic Oracle HTTP Server tasks, and remotely administer Oracle HTTP Server. It includes the following sections:

- Before You Begin
- Creating an Oracle HTTP Server Instance
- Performing Basic Oracle HTTP Server Tasks
- Remotely Administering Oracle HTTP Server
- Configuring SSL for Admin Port

- Before You Begin
  Before running Oracle HTTP Server, there are prerequisite tasks that are to be completed. These tasks include installing and configuring the server, and starting WebLogic Server and Node Manager.
- Creating an Oracle HTTP Server Instance
  The Configuration Wizard enables you to simultaneously create multiple Oracle HTTP Server instances when you create a domain.
- Performing Basic Oracle HTTP Server Tasks
  You can use WLST or Fusion Middleware Control to perform basic Oracle HTTP Server administration tasks.
- Remotely Administering Oracle HTTP Server
  You can remotely manage an Oracle HTTP Server instance running in a standalone environment from a collocated Oracle HTTP Server implementation running on a separate machine. Use WLST or Fusion Middleware Control to start, stop, and configure the server from the remote machine.
- Configuring SSL for Admin Port
  Admin port is used internally by Oracle HTTP Server (OHS) to communicate with the OHS plugin for Node Manager. The OHS plugin for Node Manager has been enhanced to use SSL for its communication with the Node Manager.

Before You Begin

Before running Oracle HTTP Server, there are prerequisite tasks that are to be completed. These tasks include installing and configuring the server, and starting WebLogic Server and Node Manager.

1. Install and configure Oracle HTTP Server as described in *Installing and Configuring Oracle HTTP Server*.
2. SSL is enabled by default on Oracle HTTP Server admin host. The admin host of the newly created instance is configured to use the `default` wallet which has a self-signed certificate. You must change the admin host after configuration to use a CA-signed

certificate for security reasons using the instructions described in "Configuring SSL for Admin Host".

3. 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 see many warning messages. Despite these messages, WebLogic Server should start normally.
> - On the Windows platform, Oracle HTTP Server requires Microsoft Visual C++ run-time libraries to be installed on the system to function. See *Installing and Configuring Oracle HTTP Server*.

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

Creating an Oracle HTTP Server Instance

The Configuration Wizard enables you to simultaneously create multiple Oracle HTTP Server instances when you create a domain.

If you are creating a WebLogic Server Domain (Full or Restricted JRF domain types), you are not required to create any instances. If you elect *not* to create any instances, a warning appears; however, you are allowed to proceed with the configuration process.

If you are creating a standalone domain, an Oracle HTTP Server instance is created by default.

This section contains the following information:

- Creating an Oracle HTTP Server Instance in a WebLogic Server Domain
- Creating an Oracle HTTP Server Instance in a Standalone Domain

> **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. See Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only).

- Creating an Oracle HTTP Server Instance in a WebLogic Server Domain
- Creating an Oracle HTTP Server Instance in a Standalone Domain

Creating an Oracle HTTP Server Instance in a WebLogic Server Domain

You can create a managed Oracle HTTP Server instance in a WebLogic Server Domain by using either the WLST custom command `ohs_createInstance()` or from Fusion Middleware Control installed as part of a Oracle Fusion Middleware infrastructure. The following sections describe these procedures.

- Creating an Instance by Using WLST
- Associating Oracle HTTP Server Instances With a Keystore Using WLST
- Creating an Instance by Using Fusion Middleware Control
- About Instance Provisioning

> **Note:**
>
> If you are working with a WebLogic Server Domain, it is recommended to use the Oracle HTTP Server WLST custom commands as described in Administering Oracle HTTP Server Using WLST. These commands offer superior error checking, provide automatic port management, and so on.

- Creating an Instance by Using WLST
- Associating Oracle HTTP Server Instances With a Keystore Using WLST
- Creating an Instance by Using Fusion Middleware Control
- About Instance Provisioning

Creating an Instance by Using WLST

You can create an Oracle HTTP Server instance in a WebLogic Server Domain by using WLST. Follow these steps.

1. From the command line, launch WLST.

   Linux or UNIX: `$ORACLE_HOME/oracle_common/common/bin/wlst.sh`

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

2. Connect to WLST:

   - In a WebLogic Server Domain:

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

     For example:

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

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

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

> **Note:**
>
> If Node Manager is down, the create command takes place partially. The master copy of the config files appear at OHS/*componentName*. Once Node Manager comes back up, the system syncs again and the runtime copy of the files appear at OHS/instances/*componentName*.

For example:

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

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

> **Note:**
>
> For information about using the WebLogic Scripting Tool (WLST), see *Understanding the WebLogic Scripting Tool*.

Associating Oracle HTTP Server Instances With a Keystore Using WLST

After using the Configuration Wizard to create Oracle HTTP Server instances in collocated mode, use the `ohs_updateInstances` WLST custom command to associate the instances with a keystore.

This command parse across all of the Oracle HTTP Server instances in the domain and perform the following tasks:

- Create a new keystore with the name `<instanceName>_default` if one does not exist.
- Put a demonstration certificate, `demoCASignedCertificate` in the newly created keystore.
- Export the keystore to the instance location.

See ohs_updateInstances.

To associate Oracle HTTP Server instances with a keystore:

1. Launch WLST from the command line.

   Linux or UNIX: `$ORACLE_HOME/oracle_common/common/bin/wlst.sh`

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

2. Connect to the Administration Server instance:

   ```
   connect('<userName', '<password>', '<host>:<port>')
   ```

3. Issue the `ohs_updateInstances` WLST custom command, for example:

```
ohs_updateInstances()
```

Creating an Instance by Using Fusion Middleware Control

You can create an Oracle HTTP Server instance in a WebLogic Server Domain by using Fusion Middleware Control installed as part of the Oracle Fusion Middleware infrastructure. Follow these 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 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 appear only 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**, enter a unique name for the Oracle HTTP Server instance; for example, `ohs_2`.
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. The port number is automatically assigned.

After creating the instance, 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 Starting Oracle HTTP Server Instances. Once started, the arrow will point up.

About Instance Provisioning

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

- The master (staging) copy will be in:

  DOMAIN_HOME/config/fmwconfig/components/OHS/*componentName*
- The runtime will be in:

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

  Node Manager must be running to provision an instance in runtime.

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

- If `ohs_createInstance()` 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.

Creating an Oracle HTTP Server Instance in a Standalone Domain

If you select Standalone as your domain during server configuration, the Configuration Wizard will create the domain, and during this process an Oracle HTTP Server instance will also be created. See *Installing and Configuring Oracle HTTP Server*.

Performing Basic Oracle HTTP Server Tasks

You can use WLST or Fusion Middleware Control to perform basic Oracle HTTP Server administration tasks.

For detailed information on the process ID (PID) file, and how to use WLST or Fusion Middleware Control to perform basic administration tasks, see the following tasks:

- About Using the WLST Commands
- Understanding the PID File
- 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
- Changing the Default Node Manager Port Number
- Understanding the PID File
- Starting Oracle HTTP Server Instances
- Stopping Oracle HTTP Server Instances
- About Using the WLST Commands
- Restarting Oracle HTTP Server Instances
- Checking the Status of a Running Oracle HTTP Server Instance
- Deleting an Oracle HTTP Server Instance
- Changing the Default Node Manager Port Number
- Updating the Node Manager Username and Password in a Standalone Domain
  You can update username and password of the Node Manager in a standalone domain using WLST commands:

Understanding the PID File

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.

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

> **See Also:**
>
> PidFile directive in the Apache HTTP Server documentation.

Starting Oracle HTTP Server Instances

This section contains information on how to start Oracle HTTP Server using Fusion Middleware Control and WLST.

> **Note:**
>
> On the Windows platform, Oracle HTTP Server requires Microsoft Visual C++ run-time libraries to be installed on the system to function. See *Installing and Configuring Oracle HTTP Server*.

This section includes the following topics:

- Starting Oracle HTTP Server Instances Using Fusion Middleware Control
- Starting Oracle HTTP Server Instances Using WLST
- Starting Oracle HTTP Server Instances from the Command Line
- Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only)
- Starting Oracle HTTP Server Instances as a Different User (UNIX Only)

Starting Oracle HTTP Server Instances Using Fusion Middleware Control

In Fusion Middleware Control, you start the Oracle HTTP Server from the Oracle HTTP Server home page. Navigate to the 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.

Starting Oracle HTTP Server Instances Using WLST

To start an Oracle HTTP Server instance by using WLST, use the `start()` command in a WebLogic Server Domain or `nmStart()` for a standalone domain. The commands are illustrated in the following table.

> **Note:**
>
> - 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`.

These commands assume you have created an Oracle HTTP Server instance, as described in Creating an Oracle HTTP Server Instance and WLST is running.

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

Starting Oracle HTTP Server Instances from the Command Line

You can start Oracle HTTP Server instances from the command line by invoking the `startComponent` script from the host that contains the Administration Server.

1. Ensure that Node Manager is running.
2. Enter the following command:

   Linux or UNIX: `$DOMAIN_HOME/bin/startComponent.sh componentName`

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

   For example:

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

The `startComponent` script contacts Node Manager and runs the `nmStart()` command.

3. When prompted, enter your Node Manager password. The system responds with these messages:

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

Exiting WebLogic Scripting Tool.
```

> **Note:**
>
> You can also use this script to start Oracle HTTP Server instances remotely. In that case, the scripts read the configuration to determine the location of the component. You must run this script from the same system as the Administration Server. See Remotely Administering Oracle HTTP Server.

- Storing Your Node Manager Password

Storing Your Node Manager Password

You can avoid having to enter your Node Manager password every time you launch the server with `startComponent` command 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. Enter your password.

The system responds with this message:

```
Creating the key file can reduce the security of your system if it is not kept
in a secured location after it is created. Creating new key...
The username and password that were used for this WebLogic NodeManager
connection are stored in $HOME/.wlst/nm-cfg-myDomainName.props and
$HOME /.wlst/nm-key-myDomainName.props.
```

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 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) use the following one-time setup on each Oracle HTTP Server machine:

1. Update the ORACLE_HOME/ohs/bin/launch file by performing the following steps as the super user (if you do not have access to super user privileges, have your system administrator perform these steps):
   a. Change ownership of the file to root:

      ```
      chown root $ORACLE_HOME/ohs/bin/launch
      ```

   b. Change the permissions on the file as follows:

      ```
      chmod 4750 $ORACLE_HOME/ohs/bin/launch
      ```

      The steps that require root permissions are now complete.

   c. Modify the port settings for Oracle HTTP Server as described in Managing Ports.
2. Configure the User and Group directive in httpd.conf.

   The configured user ID for User should be the same user ID that created the instance. The configured group ID for Group must be the same group ID used to create the instance. See Oracle HTTP Server Configuration Files. To configure Oracle HTTP Server to run as a different user id see Starting Oracle HTTP Server Instances as a Different User (UNIX Only).
3. Stop the instance if it is running by using any of the stop methods described in Stopping Oracle HTTP Server Instances.
4. Start the instance by using any of the start-up methods described in Starting Oracle HTTP Server Instances.

Starting Oracle HTTP Server Instances as a Different User (UNIX Only)

On UNIX systems, the Oracle HTTP Server worker processes (the processes that accept connections and handle requests) may be configured to run as a different user id than the user id used to create the instance.

Follow the directions in Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only) and configure the User directive with the desired user id. The configured user id must be in the same group as the group that owns the instance directory. The Group directive must also be configured and set to the same group id used to create the instance.

> **Note:**
>
> - The parent process and logging processes of the Oracle HTTP Server will run as root—these processes neither accept connections nor handle requests.
> - If Node Manager is configured to use the SSL listener, then ensure that other users have the appropriate permissions to access the SSL trust store used by NodeMmanager so that the startComponent.sh or nmConnect commands can run successfully as a different user.
>
> See Node Manager Overview in *Administering Node Manager for Oracle WebLogic Server*.

Stopping Oracle HTTP Server Instances

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

This section includes the following topics:

Stopping Oracle HTTP Server Instances Using Fusion Middleware Control

In Fusion Middleware Control, you can stop Oracle HTTP Server from the Oracle HTTP Server home page. Navigate to the Oracle HTTP Server home page and do one of the following:

- From the Oracle HTTP Server home page:
  1. Select the server instance you want to stop.
  2. Select **Control** then **Shut Down** from the Oracle HTTP Server drop-down menu on the server instance home page.
- 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 on the server instance home page, select **Shut Down**.

Stopping Oracle HTTP Server Instances Using WLST

You can stop Oracle HTTP Server by using WLST. From within the scripting tool, use one of the following commands:

> **Note:**
>
> - 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')`[1]	`nmKill(serverName='ohs1', serverType='OHS')`

1 nmKill() will also work in a WebLogic domain.

> **WARNING:**
>
> If you run shutdown() without specifying any parameters, WebLogic Server will terminate and exit WLST. Oracle HTTP Server will continue running. To recover, restart WebLogic Server, launch WLST, and reconnect to the AdminServer. Then re-run the shutdown with the Oracle HTTP Server instance name.

Stopping Oracle HTTP Server Instances from the Command Line

You can stop Oracle HTTP Server instances from the command line by invoking the `stopComponent` script from the host that contains the Administration Server.

1. Enter the following command:

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

   For example:

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

   This command invokes WLST and executes the `nmKill()` command. The `stopComponent` command will not function if Node Manager is not running.

2. When prompted, enter your Node Manager password.

If you started Oracle HTTP Server instance with the `storeUserConfig` option as described in Storing Your Node Manager Password, you will not be prompted.

Once the server is stopped, the system will respond:

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

Exiting WebLogic Scripting Tool.
```

> **Note:**
>
> You can also use this script to stop Oracle HTTP Server instances remotely. In that case, the scripts read the configuration to determine the location of the component. You must run this script from the same system as the Administration Server. See Remotely Administering Oracle HTTP Server.

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 restriction on WLST:

If you run a standalone version of Oracle HTTP Server, you must use the offline, or "agent", WLST commands. These commands are described in their appropriate context.

See Getting Started Using the Oracle WebLogic Scripting Tool (WLST) in *Oracle® Fusion Middleware Administrator's Guide*.

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 contain information on how to restart Oracle HTTP Server using Fusion Middleware Control and WLST.

- Restarting Oracle HTTP Server Instances Using Fusion Middleware Control
- Restarting Oracle HTTP Server Instances Using WLST
- Restarting Oracle HTTP Server Instances from Command Line
- Restarting Oracle HTTP Server Instances Using Fusion Middleware Control
- Restarting Oracle HTTP Server Instances Using WLST
- Restarting Oracle HTTP Server Instances from Command Line
  To restart the Oracle HTTP Server instances from the command line, use the `restartComponent` script.

Restarting Oracle HTTP Server Instances Using Fusion Middleware Control

In Fusion Middleware Control you restart Oracle HTTP Server from the Oracle HTTP Server home page. Navigate to the Oracle HTTP Server home page and do one of the following:

- From the Oracle HTTP Server home page:
  1. Select the server instance you want to restart. Select **Control**.
  2. Click **Start Up** on the instance home page, or select **Control** then **Restart** from the Oracle HTTP Server drop-down menu.
- From the Target Navigation tree:
  1. Right-click the Oracle HTTP Server instance you want to restart.
  2. Select **Control**.
  3. Select **Restart** from the Control menu.

Restarting Oracle HTTP Server Instances Using WLST

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

> **Note:**
>
> - For the WebLogic and the Standalone domains, Node Manager must be running (that is, state is `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`.
> - The `nmSoftRestart` command can also be used in WebLogic domains. To do this, you must first connect to Node Manager by using the `nmConnect` command.

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

Restarting Oracle HTTP Server Instances from Command Line

To restart the Oracle HTTP Server instances from the command line, use the `restartComponent` script.

Run the following command:

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

For example:

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

This command invokes WLST and executes the `nmSoftRestart()` command. The `restartComponent` command will not function if the Node Manager is not running. When prompted, enter your Node Manager password.

If you had started the instance with `storeUserConfig` option as described in Storing Your Node Manager Password, you will not be prompted for the Node Manager password.

Once the server is restarted, the system responds with the following message:

```
Successfully restarted server componentName...
Successfully disconnected from Node Manager...
Exiting WebLogic Scripting Tool.
```

Checking the Status of a Running Oracle HTTP Server Instance

This section contains information on how to check the status of a running Oracle HTTP Server instance. You can check this information from either Fusion Middleware Control installed as part of an Oracle Fusion Middleware infrastructure or by using WLST.

This section includes the following topics:

- Checking Server Status by Using Fusion Middleware Control
- Checking Server Status Using WLST
- Checking Server Status by Using Fusion Middleware Control
- Checking Server Status Using WLST

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. The up arrow indicates that the server instance, in this case, `ohs_2`, is running.

The down arrow indicates that the server instance, in this case, `ohs_2`, is not running.

Checking Server Status Using WLST

In a WebLogic Server Domain, if you used `ohs_createInstance()` 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:

Note:

- 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 are not included, then an error will be thrown referencing an inability to find `startWebLogic`.
- The `nmServerStatus` command can also be used in WebLogic domains. To do this, you must first connect to the Node Manager by using the `nmConnect` command.

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, you made a typo), a state of UNKNOWN will be returned.

Deleting an Oracle HTTP Server Instance

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

This section includes the following topics:

Deleting an Oracle HTTP Server Instance in a WebLogic Server Domain

In a WebLogic Server Domain, you can use either the WLST custom command `ohs_deleteInstance()` or from Fusion Middleware Control installed as part of an Oracle Fusion Middleware infrastructure. The following topics describe these procedures.

- Deleting an Instance Using WLST
- Deleting an Instance Using Fusion Middleware Control

- Deleting an Instance Using WLST
- Deleting an Instance Using Fusion Middleware Control

Deleting an Instance Using WLST

If you are in a WebLogic Server Domain, you can delete an Oracle HTTP Server instance by using the WLST custom command `ohs_deleteInstance()`. When you use this command, the following happens:

- The selected instance information is removed from config.xml.
- All Oracle HTTP Server configuration directories and their contents are deleted; for example, OHS/*instanceName* and OHS/instances/*instanceName*. These paths refer to both the runtime and master copies of the configuration.
- 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 `ohs_deleteInstance()` if Node Manager is down.

To delete an instance using WLST:

1. From the command line, launch WLST:

   Linux or UNIX: `$ORACLE_HOME/oracle_common/common/bin/wlst.sh`

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

2. Connect to WLST:

   - In a WebLogic Server Domain:

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

     For example:

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

3. At the command prompt, enter:

   ```
   ohs_deleteInstance(instanceName='instanceName')
   ```

   For example, to delete an Oracle HTTP Server instance named `ohs1` use the following command:

```
ohs_deleteInstance(instanceName='ohs1')
```

You cannot delete an Oracle HTTP Server instance in either an UNKNOWN or a RUNNING state.

> **Note:**
>
> For newly created Oracle HTTP Server instances in state UNKNOWN (for example, created with config wizard), one can start and stop the instance to move the state to SHUTDOWN. It can then be deleted successfully.
>
> For instances in state RUNNING, first stop the instance to move it to state SHUTDOWN and then it can be deleted successfully.

Deleting an Instance 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 Stopping Oracle HTTP Server Instances and then proceed with the following steps.

1. Log in to Fusion Middleware Control. Navigate to the system component instance home page for the WebLogic Server Domain that contains the Oracle HTTP Server instance you want to delete.
2. Open the WebLogic Server Domain menu and select **Administration** then **Create/Delete OHS**.
3. In the OHS Instances page, select the instance you want to delete and click **Delete**.
4. In the confirmation window, click **Yes** to complete the deletion.

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

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 if 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; you will not be able to delete the instance if it is the only one in the domain. 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 WebLogic Server domain performs deletion by using `ohs_deleteInstance()`. See Deleting an Instance 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 (assuming your instance name was 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 state, it could appear as SHUTDOWN or even FAILED_NOT_RESTARTABLE.

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

1. Shutdown all running instances (see 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 before deletion.
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**.

Changing the Default Node Manager Port Number

You can change the default value of the Node Manager port by using either WLST or the Oracle WebLogic Server Administration console.

This section includes the following topics:

- Changing the Default Node Manager Port Using WLST
- Changing the Default Node Manager Port Using Oracle WebLogic Server Administration Console
- Changing the Default Node Manager Port Using WLST

- Changing the Default Node Manager Port Using Oracle WebLogic Server Administration Console

Changing the Default Node Manager Port Using WLST

To change the default Node Manager port number using WLST, use the custom command `readDomain` to open the domain. Navigate to the directory containing Node Manager for the machine. Set the `ListenPort` property, then update the domain.

```
...
readDomain('DOMAIN_HOME')
cd('/Machines/Machine_Name/NodeManager/Node_Manager_Name')
set('ListenPort',9090)
updateDomain()
closeDomain()
...
```

In this example, *`DOMAIN_HOME`* represents the root directory of the domain. `Machines` and `NodeManager` are directories. The *`Node_Manager_Name`* is the name of Node Manager belonging to the *`Machine_Name`* machine. The default Node Manager name is `localmachine`. The default *`Machine_Name`* is also `localmachine`. The `ListenPort` value is set to 9090.

Changing the Default Node Manager Port Using Oracle WebLogic Server Administration Console

Follow these steps to change the default Node Manager port number using Oracle WebLogic Server Administration Console.

1. Manually edit the *`DOMAIN_HOME`*`/nodemanager/nodemanager.properties` file to change the value of the `ListenPort` property.
2. In the WebLogic Server Administration Console, change the configuration of the machine associated with Node Manager, to point it to the new port number.

   From the left pane of the Console, expand Environment and then select Machines. Select the machine whose configuration you want to edit. Select the Configuration tab, then the Node Manager tab. Change the Listen Port to the port updated in `nodemanager.properties` file. Click Save.

Updating the Node Manager Username and Password in a Standalone Domain

You can update username and password of the Node Manager in a standalone domain using WLST commands:

1. Launch the WebLogic Scripting Tool (WLST) by running the following command from the location *`MW_HOME`*`/oracle_common/common/bin`:

   UNIX: `./wlst.sh`

   Windows: `wlst.cmd`
2. Execute the following WLST commands:

   a. `readDomain('$`*`DOMAIN_HOME`*`')`

b. `cd('SecurityConfiguration/$DOMAIN_NAME')`

c. `set('NodeManagerUsername','new_NodeManager_Username')`

d. `set('NodeManagerPasswordEncrypted','new_NodeManager_password')`

e. `updateDomain()`

f. `closeDomain()`

If the Node Manager username and password have been saved using `storeConfig` option with `startComponent`, then delete the following after changing the Node Manager credentials and before you restart OHS:

- `user_home/.wlst/nm-key-domain_name.props`
- `user_home/.wlst/nm-cfg-domain_name.props`

Remotely Administering Oracle HTTP Server

You can remotely manage an Oracle HTTP Server instance running in a standalone environment from a collocated Oracle HTTP Server implementation running on a separate machine. Use WLST or Fusion Middleware Control to start, stop, and configure the server from the remote machine.

This section provides information about how to set up Oracle HTTP Server to run remotely.

- Setting Up a Remote Environment

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:

- Host Requirements for a Remote Environment.
- Task 1: Set Up an Expanded Domain on host1.
- Task 2: Pack the Domain on host1.
- Task 3: Unpack the Domain on host2.
- Task 4: Run Oracle HTTP Server Remotely
- Host Requirements for a Remote Environment
- Task 1: Set Up an Expanded Domain on host1
- Task 2: Pack the Domain on host1
- Task 3: Unpack the Domain on host2
- Task 4: Run Oracle HTTP Server Remotely

Host Requirements for a Remote Environment

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

- A collocated installation (for this example, 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 the collocated *MW_HOME* on *host1*. For example:

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

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.See *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.
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.
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).

Table 4-1 (Cont.) Setting Up an Expanded Domain

For...	Select or Enter...
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** (the OPSS steps may take some minutes).

Task 2: Pack the Domain on host1

On host1, use the `pack` command to pack the domain. The `pack` command creates a template archive (`.jar`) file that contains a snapshot of either an entire domain or a subset of a domain.

On Unix, run the following command:

```
MW_HOME/oracle_common/common/bin/pack.sh -domain=path_to_domain -
template=path_to_template -template_name=name -managed=true
```

For example:

```
MW_HOME/oracle_common/common/bin/pack.sh -domain=MW_HOME/user_projects/domains/
ohs1_domain -template=/tmp/ohs1_tmplt.jar -template_name=ohs1 -managed=true
```

Task 3: Unpack the Domain on host2

The unpack command creates a full domain or a subset of a domain used for a Managed Server domain directory on a remote machine. Use the following steps to unpack the domain you packed on host1 in Task 2: Pack the Domain on host1, on host2.

1. Copy the template file created in Task 2: Pack the Domain on host1 from host1 to host2.
2. Run the `unpack` command on Unix to unpack the domain:

   ```
   MW_HOME/oracle_common/common/bin/unpack.sh -domain=path_to_domain -
   template=path_to_template
   ```

   For example:

   ```
   MW_HOME/oracle_common/common/bin/unpack.sh -domain=MW_HOME/user_projects/domains/
   ohs1_domain -template=/tmp/ohs1_tmplt.jar
   ```

Task 4: Run 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', '<password>', '<nm-host>', '<nm-port>', '<domain-name>',
'<domain-directory>','ssl')
nmStart(serverName='ohs1', serverType='OHS')
```

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

Configuring SSL for Admin Port

Admin port is used internally by Oracle HTTP Server (OHS) to communicate with the OHS plugin for Node Manager. The OHS plugin for Node Manager has been enhanced to use SSL for its communication with the Node Manager.

The configuration steps described in the following topics are necessary to set up SSL communication between the OHS admin host (SSL server) and the OHS plugin for Node Manager (SSL client):

- Performing Server-Side Configuration
- Ensuring that the Host Name Verification Succeeds
- Performing Client-Side Configuration

- Performing Server-Side Configuration
  To complete the server-side configuration, you must create a wallet and enable SSL for Oracle HTTP Server admin host by modifying the `admin.conf` file present in the staging directory.
- Ensuring that the Host Name Verification Succeeds
  Host name verification happens as part of the SSL handshake between the Node Manager and the Oracle HTTP Server (OHS) admin host.
- Performing Client-Side Configuration
  On the client-side, you must configure trust for the Node Manager.

Performing Server-Side Configuration

To complete the server-side configuration, you must create a wallet and enable SSL for Oracle HTTP Server admin host by modifying the `admin.conf` file present in the staging directory.

To do this, refer to the following topics:

1. Creating a Wallet
2. Enabling SSL for Oracle HTTP Server Admin Host

For information about modifying the `admin.conf` file, see Modifying an Oracle HTTP Server Configuration File.

- Creating a Wallet
  Create a wallet that contains a certificate signed by a trusted CA.

- Enabling SSL for Oracle HTTP Server Admin Host
  Enable SSL for the admin host by configuring the following `mod_ossl` directives in a `<IfModule ossl_module>` block.

Creating a Wallet

Create a wallet that contains a certificate signed by a trusted CA.

Consider the requirements for ensuring the success of the host-name verification step of the SSL handshake while choosing the `Common Name` attribute of the certificate's Distinguished Name(DN). See Ensuring that the Host Name Verification Succeeds.

To create a wallet, refer to the following topics depending on your installation type:

- Creating a Wallet for a Standalone Installation
- Creating a Wallet for a Collocated Installation

Creating a Wallet for a Standalone Installation

To create a wallet for a standalone installation, use the `KEYTOOL` utility to create a keystore, generate a Certificate Signing Request (CSR), import the required certificates to the keystore, convert this keystore to a wallet using the `ORAPKI` utility, and then configure Oracle HTTP Server admin host to use this wallet.

To do this, complete the following steps:

1. Set the following environment variables:

   On UNIX:

   ```
   export ORACLE_HOME=absolute_path_to_ORACLE_HOME
   export PATH=$ORACLE_HOME/oracle_common/bin:$PATH
   export JAVA_HOME=absolute_path_to_JDK8
   ```

   On Windows:

   ```
   set ORACLE_HOME=absolute_path_to_ORACLE_HOME
   set PATH=%ORACLE_HOME%\oracle_common\bin:%PATH%
   set JAVA_HOME=absolute_path_to_JDK8
   ```

2. Set up a working directory and change the directory to the same:

   ```
   mkdir walletkey
   cd walletkey
   ```

3. Create a keystore and a private key:

   ```
   keytool -genkey -alias ca_cert -keyalg RSA -keysize 2048 -sigalg
   SHA256withRSA -dname "CN=hostname.domainname,O=My Company
   Corporation,L=Denver,ST=CO,C=US" -keypass keypass_password -keystore
   keystore.jks -storepass storepass_password
   ```

   In this command:

   - The alias `ca_cert` is what will be established. You can choose a different name.

- `keystore.jks` is the name you choose for the new keystore.
- *`keypass_password`* and *`storepass_password`* are the password you specify for keypass and storepass respectively.

4. Generate a Certificate Signing Request (CSR) and send it to your Certificate Authority (CA) before you proceed (otherwise, use a self-signed):

```
keytool -certreq -v -alias ca_cert -file server.csr -sigalg
SHA256withRSA -keypass keypass_password -storepass
storepass_password -keystore keystore.jks
```

In this command:

- `ca_cert` is the alias you specified in the previous step.
- `server.csr` is what you give the CA.
- `keystore.jks` is the keystore.

5. Import the root trust certificate into the keystore:

```
keytool -import -v -noprompt -trustcacerts -alias root -file
root.crt -keystore keystore.jks
```

In this command:

- The alias `root` is the name you choose for the intermediate CA trust certificate.
- `root.crt` is the CA's root trust certificate.
- `keystore.jks` is the keystore.

6. If supplied from CA, import the Intermediate trust certificate into a the keystore, and choose an alias:

```
keytool -import -v -noprompt -trustcacerts -alias intermediate -
file intermediate.crt -keystore keystore.jks
```

In this command:

- The alias `intermediate` is the name you choose for the intermediate CA trust certificate.
- `intermediate.crt` is the CA's intermediate trust certificate.
- `keystore.jks` is the keystore.

7. Import the signed server certificate into the keystore:

```
keytool -import -v -alias ca_cert -file server.crt -keystore
keystore.jks
```

In this command:

- The alias `ca_cert` is the name you had chosen for server certificate.
- `server.crt` is the signed server certificate that you normally get from the CSR.

- `keystore.jks` is the keystore.

8. Convert the keystore to the wallet:

```
orapki wallet create -wallet ./wallet -auto_login_only
orapki wallet jks_to_pkcs12 -wallet ./wallet -keystore ./keystore.jks -
jkspwd jks_password
```

9. Configure the wallet in the Oracle HTTP Server `admin.conf` file. To make it simple and consistent, use a generic central location as shown in the following example. Ensure that the location is owned by the same Oracle user.

   Example of `admin.conf` file:

```
<VirtualHost AdminHostIP:AdminPort>
<IfModule ossl_module>
 ...
 SSLWallet "/usr/oracle/ohs/wallets"
 ...
</IfModule>
</VirtualHost>
```

Creating a Wallet for a Collocated Installation

For a collocated installation, create a wallet for Oracle HTTP Server admin host via Fusion Middleware Control and configure the Oracle HTTP Server admin host to use this wallet.

1. Log in to the Fusion Middleware Control using the WebLogic username and password:

   `http://host.domain:port/em`

2. Start the relevant OHS component (for example, `ohs1`) via Fusion Middleware Control.

> **Note:**
>
> A keystore is uniquely identified by an application stripe and a keystore within that stripe. Keys and certificates are created in keystores within stripes. Stripe names within the security store are unique in the security store, and the keystore names within a stripe are unique in the stripe. For example, (stripe1,keystoreA), (stripe1,keystoreB), and (stripe2,keystoreA) refer to three distinct keystores. Applications can create more than one keystore within the application stripe.

3. Create a Stripe for Oracle HTTP Server:
   a. Navigate to the weblogic domain, go to **Security**, and click **Keystore**.
   b. Click **Create Stripe**.
   c. Create new stripe by name `OHS`. Note that the name is case sensitive.
4. Create a Keystore for OHS instance:
   a. Click on the ohs instance.
   b. Navigate to **OracleHTTPServer**, go to **Security**, and click **Keystore**.

c. Click **Create Keystore** and then click **Create it as a Policy**.

d. Enter the keystore name. For example, `Test`. A new keystore is created with the name *instancename*`_Test` (for example, `ohs1_Test`).

5. Generate Keypair:

a. Select the new keystore (`ohs1_Test`) and click **Manage**.

b. Click **Generate Keypair**.

c. Enter the required details and click **OK**.

6. Generate CSR:

a. Select the new Keypair generated.

b. Click **Generate CSR**. The page with the following information is displayed:

```
Certificate signing request with Alias: ohs_cert is exported
successfully. To export it to a
      file, click "Export CSR". You can send this file to a CA
or you can cut and paste the entire
      text in the box from BEGIN NEW CERTIFICATE REQUEST to END
NEW CERTIFICATE REQUEST. Once you
      get your certificate back from CA you can continue with
import.
```

c. Click **Export CSR** and save the file.
Ensure that the **Lock and Edit** is used so that the changes are committed. If the keystore is not saved, you cannot import the new certificate. Therefore, before requesting the certificate, exit the browser and go back to verify that the keystore is saved.

7. Obtain CA signed certificate by sending CSR to any CA and obtaining the certificates.

8. Import the Trusted Certificate:

a. Navigate to **OracleHTTPServer**, go to **Security**, and click **Keystore**.

b. Select the keystore from which the CSR was generated and click **Manage**.

c. Click **Import**.

d. In the Certificate Type, select **Trusted Certificate** and either paste the contents of the root CA certificate `rootca.crt`, or select the file and click **OK**.

e. Repeat the above steps for any other Trusted CA Certificates in the chain.

9. Import the Trusted Certificate to weblogic domain. Also import the root CA certificate and any other Trusted CA Certificates to weblogic `system` stripe under trust keystore:

a. Navigate to the weblogic domain, go to **Security**, and click **Keystore**.

b. Expand **system** stripe, select **trust keystore**, and click **Manage**.

c. Click **Import**.

d. In the Certificate Type, select **Trusted Certificate** and either paste the contents of the root CA certificate `rootca.crt`, or select the file and click **OK**.

e. Repeat the above steps for any other Trusted CA Certificates in the chain.

10. Import the User Certificate:
    a. Navigate to **OracleHTTPServer**, go to **Security**, and click **Keystore**.
    b. Select the keystore from which the CSR was generated, and click **Manage**.
    c. Click **Import**.
    d. In the Certificate Type, select **Certificate** and either paste the contents of `server.crt`, or select the file and click **OK**.
11. Export Key store to wallet:
    a. Navigate to **OracleHTTPServer**, go to **Security**, and click **Keystore**.
    b. Select the keystore from which the CSR was generated, and click **Manage**.
    c. Click **Export Keystore to Wallet**. An Auto-Login Only Wallet is created in keystore directory of OHS instance.

> **Note:**
>
> Before you export the keystore to a wallet, ensure that you click **Lock and Edit** and then click **Activate Changes**.

12. Configure the wallet in the Oracle HTTP Server by editing the `admin.conf` file to point to the newly created wallet.

    Example of `admin.conf` file:

```
<VirtualHost AdminHostIP:AdminPort>
<IfModule ossl_module>
 ...
 SSLWallet "/usr/oracle/ohs/wallets"
 ...
</IfModule>
</VirtualHost>
```

> **Note:**
>
> `admin.conf` file cannot be edited via Fusion Middleware Control. To manually edit it, see Modifying an Oracle HTTP Server Configuration File.

Enabling SSL for Oracle HTTP Server Admin Host

Enable SSL for the admin host by configuring the following `mod_ossl` directives in a `<IfModule ossl_module>` block.

By default, `admin.conf` file includes the following configuration settings.

- `SSLEngine ON`
  For more information, see SSLEngine Directive.
- `SSLProtocol TLSv1.2`
  For more information, see SSLProtocol Directive.
- `SSLCipherSuite`

For more information, see SSLCipherSuite Directive.

- `SSLWallet`
  Set this directive to the wallet created in Creating a Wallet. For more information, see SSLWallet Directive.

Sample configuration:

```
<VirtualHost 127.0.0.1:9991>
<IfModule ossl_module>
 SSLEngine on
 SSLProtocol TLSv1.2
 SSLCipherSuites
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GC
M_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_A
ES_256_CBC_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_
WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RS
A_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECD
HE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_E
CDHE_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_W
ITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES
_256_CBC_SHA256,SSL_RSA_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_AES_256_CBC_S
HA
 SSLWallet "<wallet location>"
</IfModule>
</VirtualHost>
```

Ensuring that the Host Name Verification Succeeds

Host name verification happens as part of the SSL handshake between the Node Manager and the Oracle HTTP Server (OHS) admin host.

Host name verification succeeds if the host name in the admin host URL to which the Node Manager connects, matches the host name in the digital certificate that the OHS admin host sends back as part of the SSL connection.

To ensure that this verification step succeeds, you must configure the host name for Oracle HTTP Server admin host correctly as described in the following sections:

- ServerName Directive Configuration
- Listen Directive Configuration

ServerName Directive Configuration

Use the `ServerName` directive to configure the host name for the Oracle HTTP Server admin host. The host name configured must match the `Common Name` attribute of the SSL certificate's `Distinguished Names` or match the `subjectAltName` extension. Place the `ServerName` directive within the `<VirtualHost>` block in the `admin.conf` file.

> **Note:**
>
> If `ServerName` directive is not configured when SSL is enabled for the communication between Node Manager and the OHS admin host, OHS fails to start with the following message:
>
> `ServerName directive is not configured in admin.conf of <ohs_instance>`

Once the host name is configured, changes to the Listen directive configuration may be required, as Listen directive and host name configurations are linked.

Listen Directive Configuration

Choose an IP address and port for the admin host and configure the Listen directive with this. The host name configured using the `ServerName` directive must map to the IP address configured in the `Listen` directive of the `admin.conf` file. This is to avoid the host name resolution errors during the communication between Node Manager and OHS admin host. The IP address used for the Listen directive must match the one used with the `<VirtualHost>` directive.

> **Note:**
>
> Use `nslookup` to ensure that the IP address used in the Listen directive is correctly mapped to the host name of the admin host.
>
> If the `Listen` directive is configured to listen on all available interfaces (that is, `Listen <port>`), instead of a specific IP address (that is, `Listen <ipaddress>:<port>`, OHS fails to start with the following message:
>
> ```
> HostName/IP address is not configured for Listen directive in
> admin.conf of <ohs_instance_name>
> ```

After you configure SSL on the server-side, the `admin.conf` configuration looks like the following sample:

```
#[Listen] OHS_PROXY_PORT
Listen <IP>:<PORT>
#[VirtualHost] OHS_PROXY_VH
<VirtualHost <IP>:<PORT>>

// Ensure <HOSTNAME> resolves to <IP>
ServerName <HOSTNAME>
<Location /dms/>
  SetHandler dms-handler
  Require all granted
</Location>
CustomLog "||${PRODUCT_HOME}/bin/odl_rotatelogs
${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/admin_log 43200" common
<IfModule ossl_module>
  SSLEngine on
```

```
  SSLProtocol TLSv1.2
  SSLCipherSuite
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_GC
M_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_A
ES_256_CBC_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_
WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RS
A_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECD
HE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_E
CDHE_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_W
ITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_CBC_SHA256,TLS_RSA_WITH_AES
_256_CBC_SHA256,SSL_RSA_WITH_AES_128_CBC_SHA,SSL_RSA_WITH_AES_256_CBC_S
HA

// Ensure CN attribute of the certificate's DN matches <HOSTNAME>
  SSLWallet "<WALLET LOCATION>"
 </IfModule>
</VirtualHost>
```

> **Note:**
>
> In the above sample, `<IP>`, `<PORT>`, `<HOSTNAME>`, and `<WALLET LOCATION>` are the details of your environment.

Performing Client-Side Configuration

On the client-side, you must configure trust for the Node Manager.

Ensure that Node Manager is able to trust the certificate configured for the Oracle HTTP Server (OHS) admin host. This is done by exporting the certificate of root CA that signed the user certificate present in the OHS admin host's wallet and importing the same into the Node Manager's wallet for the instance as a trusted certificate. The Oracle HTTP Server plugin for Node Manager is enhanced to maintain a per-instance wallet that contains the trusted certificates for the OHS admin host of that instance.

To configure the Node Manager's wallet for an instance, add the `nm-wallet` property to the `ohs.plugins.nodemanager.properties` file located at `$DOMAIN_HOME/config/fmwconfig/components/COMPONENT_TYPE/COMPONENT_NAME`, and set it to the absolute path to the wallet that contains the trusted certificates.

To set up trust for the Node Manager:

1. Export the root CA certificate that signed the user certificate present in the Oracle HTTP Server admin host's wallet:

   ```
   $orapki wallet export -wallet path_to_server_wallet -dn "DN for
   root CA certificate" -cert root_CA.crt
   ```

2. Create a wallet for the Node Manager :

   ```
   $orapki wallet create -wallet /test/my_nm_wallet -auto_login_only
   ```

3. Import the certificate for the root CA into *my_nm_wallet* as a trusted certificate:

```
$orapki wallet add -wallet /test/my_nm_wallet -trusted_cert -cert
root_CA.crt -auto_login_only
```

Configure the `nm-wallet` property in the `ohs.plugins.nodemanager.properties` file to point to the Node Manager's wallet:

1. Open the file `ohs.plugins.nodemanager.properties` located at `$DOMAIN_HOME/config/fmwconfig/components/COMPONENT_TYPE/COMPONENT_NAME` in a text editor.
2. Add `nm-wallet=/test/my_nm_wallet` to the end of the file.

> **Note:**
>
> You cannot edit the `ohs.plugins.nodemanager.properties` file using Fusion Middleware Control or WLST. To edit it manually, see Modifying an Oracle HTTP Server Configuration File.

5

Working with Oracle HTTP Server

When working with an installed version of Oracle HTTP Server, there are some common tasks that you have to perform, such as editing configuration files, specifying server properties, and more.

This chapter includes the following sections:

- About Editing Configuration Files
- Specifying Server Properties
- Configuring Oracle HTTP Server Instances
- Configuring the mod_security Module
- About Editing Configuration Files
  Configuration files are to be edited only after the Administration Server is stopped to avoid losing the changes.
- Specifying Server Properties
  Server properties include items like the document root, administrator email, directory index, and operating system details. You can set Oracle HTTP Server properties by using Fusion Middleware Control only or by directly editing the configuration files. You cannot use WLST commands to specify the server properties.
- Configuring Oracle HTTP Server Instances
  Some of the common Oracle HTTP Server instance configuration procedures are related to secure sockets, MIME settings, Oracle WebLogic Server proxy plug-in (mod_wl_ohs), mod_proxy_fcgi, and more.
- Configuring the mod_security Module
  You can use the open-source `mod_security` module to detect and prevent intrusion attacks against Oracle HTTP Server. For example, specifying a `mod_security` rule to screen all incoming requests, and deny requests that match the conditions specified in the rule.

About Editing Configuration Files

Configuration files are to be edited only after the Administration Server is stopped to avoid losing the changes.

For instances that are part of a WebLogic Server Domain, Fusion Middleware Control and the management infrastructure manages the Oracle HTTP Server configuration. 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 stopped. When the administration server is subsequently started (or restarted), the results of any manual edits will be replicated to the run-time directory on the node of the managed instance.

See Understanding Configuration Files.

The following sections provide more information on modifying configuration files.

Editing a Configuration File for a Standalone Domain

For standalone instances, you can edit the configuration directly within the staging directory at any time. The runtime config files are updated on start, restart or stopping of the Oracle HTTP Server instance.

Editing a Configuration File for a WebLogic Server Domain

You can modify configuration files for a Weblogic Server Domain. Use the Fusion Middleware Control to edit these files. The changes are displayed on the Advanced Server Configuration page after you restart the Oracle HTTP Server.

> **Note:**
>
> You cannot edit `admin.conf` file using Fusion Middleware Control. You must use a text editor to edit the `admin.conf` file manually.

To open and edit configuration files using Fusion Middleware Control, do the following:

1. Select **Administration** from the HTTP Server menu.
2. Select **Advanced Configuration** from the Administration menu item.
3. In the Advanced Server Configuration page, select the configuration file from the **Select File** drop-down list, such as the `httpd.conf` file, then click **Go**.
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 Restarting Oracle HTTP Server Instances .

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

Specifying Server Properties

Server properties include items like the document root, administrator email, directory index, and operating system details. You can set Oracle HTTP Server properties by using Fusion Middleware Control only or by directly editing the configuration files. You cannot use WLST commands to specify the server properties.

This section includes the following topics:

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

Specifying Server Properties by Using Fusion Middleware Control

Follow these steps to specify the server properties by using Fusion Middleware Control.

1. Select **Administration** from the Oracle HTTP Server menu.
2. Select **Server Configuration** from the Administration menu.
3. In the Server Configuration page, enter the server properties.
    a. Enter the documentation root directory in the **Document Root** field that forms the main document tree visible from the website.
    b. Enter the e-mail address in the **Administrator's E-mail** field that the server will include in error messages sent to the client.
    c. 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.
    d. Use the Modules region to enable or disable modules. The available modules are mod_authnz_fcgi and mod_proxy_fcgi. See About Configuring mod_proxy_fcgi.
    e. 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.
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 as described in Restarting Oracle HTTP Server Instances .

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

Specify Server Properties by Editing the httpd.conf File

You can specify server properties by manually editing the `httpd.conf` file. Follow these steps to edit 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. See Understanding Configuration Files.

1. Open the `httpd.conf` file (the "master" or "staging" copy: `$DOMAIN_HOME/config/fmwconfig/components/OHS/instance_name/httpd.conf`)by using either a text editor or the Advanced Server Configuration page in Fusion Middleware Control. (See 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">
   Options Indexes MultiViews    AllowOverride None    Require all granted</
   Directory>
   ```

6. Save the file.

7. Restart Oracle HTTP Server as described in Restarting Oracle HTTP Server Instances .

Configuring Oracle HTTP Server Instances

Some of the common Oracle HTTP Server instance configuration procedures are related to secure sockets, MIME settings, Oracle WebLogic Server proxy plug-in (mod_wl_ohs), mod_proxy_fcgi, and more.

> **Note:**
>
> This section does not include initial system configuration information. For initial system configuration instructions, see *Installing and Configuring Oracle HTTP Server*.

This section includes the following topics:

- Disabling the Options Method
- Updating Oracle HTTP Server Component Configurations on a Shared File System

> **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.

- Secure Sockets Layer Configuration
- Configuring Secure Sockets Layer in Standalone Mode
- Exporting the Keystore to an Oracle HTTP Server Instance Using WLST
- Configuring MIME Settings Using Fusion Middleware Control
- About Configuring mod_proxy_fcgi
- About Configuring the Oracle WebLogic Server Proxy Plug-In (mod_wl_ohs)
- Removing Access to Unneeded Content
- Using the apxs Command to Install Extension Modules
- Disabling the Options Method
- Updating Oracle HTTP Server Component Configurations on a Shared File System

Secure Sockets Layer Configuration

Secure Sockets Layer (SSL) is an encrypted communication protocol that is designed for securely sending messages across the Internet. SSL resides between Oracle HTTP Server on the application layer and the TCP/IP layer. It transparently handles 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 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/*componentName*/ssl.conf to point to the location of your real wallet.

Oracle strongly recommends that you do not use a certificate that uses the Message Digest 5 algorithm (MD5). This algorithm has been severely compromised. The MD5 certificate must be replaced with a certificate that uses Secure Hash Algorithm 2 (SHA-2), which provides more secure encryption.

For the changes to take effect, restart Oracle HTTP Server, as described in Restarting Oracle HTTP Server Instances .

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

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 use the mod_ossl module to Oracle HTTP Server which enables the server to use SSL.

- Configure SSL
- Specify SSLVerifyClient on the Server Side
- Enable SSL Between Oracle HTTP Server and Oracle WebLogic Server
- Using SAN Certificates with Oracle HTTP Server
- Configure SSL
- Specify SSLVerifyClient on the Server Side
- Enable SSL Between Oracle HTTP Server and Oracle WebLogic Server
- Using SAN Certificates with Oracle HTTP Server

Configure SSL

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

- Task 1: Create a Real Wallet
- Task 2: (Optional) Customize Your Configuration
- Basic SSL Configuration Example
- Task 1: Create a Real Wallet
- Task 2: (Optional) Customize Your Configuration
- Basic SSL Configuration Example

Task 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 `ssl.conf` (the pre-installation 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.

Task 2: (Optional) Customize Your Configuration

Optionally, you can further customize your configuration using `mod_ossl` directives.

> **See Also:**
>
> - mod_ossl Module for a list and descriptions of directives accepted by `mod_ossl`.
> - SSLFIPS Directive for information on how to configure the `SSLFIPS` directive and a list of the cipher suites it accepts.

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

Basic SSL Configuration Example

Your SSL configuration must contain, at minimum, the directives in the following example.

```
LoadModule ossl_module        "${PRODUCT_HOME}/modules/mod_ossl.so"
Listen 4443
ServerName www.testohs.com
SSLEngine on
#  SSL Protocol Support:
#  List the supported protocols.
SSLProtocol TLSv1.2 TLSv1.1 TLSv1
#  SSL Cipher Suite:
#  List the ciphers that the client is permitted to negotiate.
SSLCipherSuite
SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA,TLS_RSA
_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA
SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/instances/$
{COMPONENT_NAME}/keystores/default"
</VirtualHost>To enable client authentication, do the following:
```

Specify SSLVerifyClient on the Server Side

This section describes the different ways of using the **SSLVerifyClient** directive to authenticate and authorize access. Use the appropriate client certificate on the client side for the HTTPS connection. See your client documentation for information on getting and using a client certificate. Ensure that the Oracle server wallet trusts your client certificate.

To ensure that the server trusts the client certificate, you can check whether the client certificate is self-signed or signed by a certificate authority (CA). In both cases, the certificate must be added to the list of trusted certificates.

You can add a trusted client certificate to an Oracle wallet using one of the following ways:

- Adding a Trusted Client Certificate in a Standalone Oracle HTTP Server Installation
- Adding a Trusted Client Certificate in Collocated Oracle HTTP Server Installation

The following subsections describe the different methods of using the **SSLVerifyClient** directive to authenticate and authorize access:

- Forcing Clients to Authenticate Using Certificates
- Forcing a Client to Authenticate for a Particular URL
- Authorizing a Client for a Particular URL
- Allowing Clients with Strong Ciphers and CA Client Certificate or Basic Authentication
- Adding a Trusted Client Certificate in a Standalone Oracle HTTP Server Installation
- Adding a Trusted Client Certificate in Collocated Oracle HTTP Server Installation
- Forcing Clients to Authenticate Using Certificates
- Forcing a Client to Authenticate for a Particular URL
- Authorizing a Client for a Particular URL
- Allowing Clients with Strong Ciphers and CA Client Certificate or Basic Authentication

Adding a Trusted Client Certificate in a Standalone Oracle HTTP Server Installation

To add a trusted certificate to the wallet in a standalone installation, use the **`orapki`** command. See orapki in *Administering Oracle Fusion Middleware*.

Adding a Trusted Client Certificate in Collocated Oracle HTTP Server Installation

To add a trusted certificate to a wallet in a collocated installation, use the Fusion Middleware Control or the WebLogic Scripting Tool.

1. Import the certificate into the trusted certificate list of the keystore.
2. Export keystore into the server's wallet after importing trusted certificates to the keystore.

To import certificate using the Fusion Middleware Control, see Managing Certificates with Fusion Middleware Control in *Securing Applications with Oracle Platform Security Services*. Export keystore option is not provided in the Fusion Middleware Control.

To import certificate and export keystore using the WebLogic Scripting Tool, see Managing Certificates with WLST and Managing Keystores with WLST in *Securing Applications with Oracle Platform Security Services*.

Forcing Clients to Authenticate Using Certificates

You can force the client to validate its client certificate and allow access to the server using `SSLVerifyClient`. 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"
```

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. In this case, the `SSLVerifyClient` appears in a `Location` block.

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

Authorizing a Client for a Particular URL

To authorize a client for a particular URL, 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 this directive, see SSLOptions Directive 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>
```

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>
```

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. See Use SSL with Plug-Ins and Parameters for Oracle WebLogic Server Proxy Plug-In in *Using Oracle WebLogic Server Proxy Plug-Ins*.

Using SAN Certificates with Oracle HTTP Server

A Subject Alternative Name (SAN) Certificate or Unified Communications Certificates (UCC) can secure multiple sub-domains that are specified in Subject Alternative name field.

You can use the **Subject Alternative Name** (SAN) field to specify additional host names (for example, site, IP address, command name) that are to be protected by a single SSL certificate. Using a SAN certificate, you can secure host names on different base domains in one SSL certificate. You can also host multiple SSL enabled sites on a single server by using Multi-Domain (SAN) Certificate with Subject Alternative Names. Certificates with SAN extension do not support use of wildcards. So you must add each subdomain individually.

Create Certificate Request with SAN Extension by Using orapki Utility

Use the orapki utility to create certificate request with SAN extension. See Adding a Certificate Request to an Oracle Wallet.

Sample Configuration Using SAN Certificates

1. Create a `<VirtualHost>` block for each host that you want to serve using the same IP address and port.
2. In each `<VirtualHost>` block, set up the *ServerName* directive to designate which host is being served.

   For example, if `VH1` is the first virtual host block, set the *ServerName* as `ServerName ns1.example.com`. Similarly, if `VH2` is the second virtual host block, set the *ServerName* as `ServerName ns2.example.com`.
3. Generate a certificate with the host names referring the different virtual hosts added to the SAN extension field.
4. In each `<VirtualHost>` block, set up the *SSLWallet* directive to the wallet that contains the certificate generated in Step 3.

   For example, `SSLwallet server`.
5. Save the changes and start Oracle HTTP Server.

Sample Configuration Example

```
Listen 4443
<VirtualHost>
 ServerName ns1.example.com
 SSLWallet "server"
</VirtualHost>

<VirtualHost>
 ServerName ns2.example.com
 SSLWallet "server"
</VirtualHost>
```

Restrictions

Oracle HTTP Server does not support Server Name Indication (SNI) extension. In absence of SNI support, when setting up more than one SSL enabled virtual host by using a certificate with several `SubjectAltName` extension entries, only the `per-vhost mod_ossl` directives set for the first virtual host are considered.

Consider the following configuration:

```
# Ensure that Apache listens on port 443
Listen 443
<VirtualHost *:443>
  # Because this virtual host is defined first, it will
  # be used as the default
  DocumentRoot /www/example1
  ServerName ns1.example.com
  # Other directives here
  SSLCipherSuite AES
  SSLProtocol TLSv1
</VirtualHost>
<VirtualHost *:443>
  DocumentRoot /www/example2
  ServerName ns2.example.com
  # Other directives here
  SSLCipherSuite AES-GCM
  SSLProtocol TLSv1.2
</VirtualHost>
```

When connecting to both `ns1.example.com` and `ns2.example.com`, permitted ciphers and protocols are AES and TLSv1 respectively. Although the cipher suite directive is set to AES-GCM and the protocol version is set to TLSv1.2 for `ns2.example.com`, the ones used in handshake while connecting to `ns2.example.com` would be AES cipher and TLSv1 protocol only.

Exporting the Keystore to an Oracle HTTP Server Instance Using WLST

The collocated Oracle HTTP server uses the Oracle wallet during run time. It is recommended not to manage certificates in the Oracle wallet using tools like orapki.

Instead, use the central storage and unified management available with the Keystore Service to manage wallets and their contents through the export, import, and synchronization features of that service. The `exportKeyStore` command provided by KSS, can be used for exporting the keystore to the wallet. However, there are many nuances that the user has to be aware of while using the `exportKeyStore` command. Hence, a custom OHS WLST command called `ohs_exportKeystore` is provided.

Use the WLST custom command `ohs_exportKeyStore` to export the keystore to the Oracle wallet after modifying the keystore. For more information about this command and naming conventions for keystores, see ohs_exportKeyStore.

1. Launch WLST from the command line.

   Linux or UNIX: `$ORACLE_HOME/oracle_common/common/bin/wlst.sh`

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

2. Connect to the Administration Server instance:

   ```
   connect('<userName', '<password>', '<host>:<port>')
   ```

3. Issue the `ohs_exportKeyStore` WLST custom command:

   ```
   ohs_exportKeyStore(keyStoreName = '<keystore_name>', instanceName =
   '<name_of_the_OHS_instance>')
   ```

   For example, to export the `ohs1_myKeystore` keystore to the `ohs1` Oracle HTTP Server instance:

   ```
   ohs_exportKeyStore(keyStoreName = 'ohs1_myKeystore', instanceName = 'ohs1')
   ```

Configuring MIME Settings Using Fusion Middleware Control

Oracle HTTP Server uses Multipurpose Internet Mail Extension (MIME) settings 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

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

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.

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. Scroll to the MIME Types region.
3. Click **Add Row** in MIME Configuration region. A new, blank row is added to the list.
4. Enter the MIME type and its associated file extension.
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 Restarting Oracle HTTP Server Instances .

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

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. Scroll to the MIME Encoding region.
3. Expand the MIME Encoding region, if necessary, 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 Restarting Oracle HTTP Server Instances .

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. Scroll to the MIME Languages region.
3. Expand the MIME Languages region, if necessary, 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. To choose a default MIME language, select the desired row, then click **Set As Default**. The default language will appear in the Default MIME Language field.
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 Restarting Oracle HTTP Server Instances .

About Configuring mod_proxy_fcgi

The mod_proxy_fcgi module does not have configuration directives. Instead, it uses the directives set on the mod_proxy module. Unlike the mod_fcgid and mod_fastcgi modules, the mod_proxy_fcgi module has no provision for starting the application process. The purpose of mod_proxy_fcgi is to move this functionality outside of the web server for faster performance. So, mod_proxy_fcgi simply will act as a reverse proxy to an external FastCGI server.

For more information on configuring the mod_proxy_fcgi module, see Task 3: Configure mod_proxy_fcgi to Act as a Reverse Proxy to an External FastCGI Server and Task 4: Setup an External FastCGI Server.

About 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 manually editing the mod_wl_ohs.conf configuration file.

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 WebLogic Proxy Plug-In for Oracle HTTP Server in *Using Oracle WebLogic Server Proxy Plug-Ins*.

- Configuring SSL for mod_wl_ohs

Configuring SSL for mod_wl_ohs

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between the plug-in and Oracle WebLogic Server. The SSL protocol provides confidentiality and integrity to the data passed between the plug-in and WebLogic Server. See Using SSL with Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

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. Starting with the Oracle HTTP Server 12*c* (12.2.1) release, some of these sections are commented out.

You might want to tailor this extra content in your own environment to suit your use cases. To access the httpd.conf file, see About Editing Configuration Files to access the file.

This section includes the following topics:

- Edit the cgi-bin Section
- Edit the Fancy Indexing Section
- Edit the Product Documentation Section

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
    Require all granted
</Directory>
...
```

Edit the Fancy Indexing Section

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

```
...
# Uncomment the following line to enable the fancy indexing configuration
# below.
# Define ENABLE_FANCYINDEXING
<IfDefine ENABLE_FANCYINDEXING>

# 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">
    Options Indexes MultiViews
    AllowOverride None
    Require all granted
</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
</IfDefine>
```

Edit the Product Documentation Section

Uncomment the `Define MANUAL_ENABLE` line to enable the manual configuration of product documentation.

```
...
#
# Uncomment the following line to enable the manual configuration below.
# Define ENABLE_MANUAL
<IfDefine ENABLE_MANUAL>
AliasMatch ^/manual(?:/(?:de|en|es|fr|ja|ko|pt-br|ru|tr))?(/.*)?$ "$
{PRODUCT_HOME}/manual$1"

<Directory "${PRODUCT_HOME}/manual">
    Options Indexes
    AllowOverride None
    Require all granted

    <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>
</IfDefine>
```

Using the apxs Command to Install Extension Modules

> **Note:**
>
> This command is only for UNIX and Linux and is necessary 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/2.4/programs/apxs.html

The Apache Extension Tool (apxs) can 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.

> **Note:**
>
> Once any third-party module is created and loaded, it falls under the third-party criteria specified in the Oracle HTTP Server support policy. Before continuing with this procedure, you should be aware of this policy. See Oracle HTTP Server Support.

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 `ORACLE_HOME`, a `LoadModule` directive in httpd.conf or other configuration file loads 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 Modifying an Oracle HTTP Server Configuration File.

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.

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>
```

Updating Oracle HTTP Server Component Configurations on a Shared File System

You might encounter functional or performance issues when an Oracle HTTP Server component is created on a shared file system, such as NFS (Network File System). In particular, lock files or UNIX sockets used by Oracle HTTP Server might not work or may have severe performance degradation; Oracle WebLogic Server requests routed by mod_wl_ohs may have severe performance degradation due to file system accesses in the default configuration.

Table 5-1 provides information about the Lock file issues and the suggested changes in the `httpd.conf` file specific to the operating systems.

Table 5-1 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 `Mutex fnctl:fileloc default` to `Mutex sysvsem default` where `fileloc` is the value of the directive `Mutex` (three places in httpd.conf).

Table 5-1 (Cont.) Lock File issues

Operating System	Description	httpd.conf changes
Solaris	Lock files are not required. The cross-process pthread mutex is the preferred cross-process mutex implementation.	Change `Mutex fnctl:fileloc default` to `Mutex pthread default` where `fileloc` is the value of the directive `Mutex` (three places in httpd.conf).
Other UNIX platforms		Change the file location specified in the `Mutex` directive to point to a local file system (three places in httpd.conf).
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 file system.	

Configuring the mod_security Module

You can use the open-source `mod_security` module to detect and prevent intrusion attacks against Oracle HTTP Server. For example, specifying a `mod_security` rule to screen all incoming requests, and deny requests that match the conditions specified in the rule.

The `mod_security` module 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. The shared object `mod_security2.so` has a dependency on cURL and OpenSSL libraries. These libraries are also included in the Oracle HTTP Server installation. The `mod_security` module provided as part of the Oracle HTTP Server installation supports the `SecRemoteRules` directive. This directive allows an user to load rules from a target server. The communication from Oracle HTTP server to the target server that hosts the `mod_security` rules happens over TLS and is implemented using cURL and OpenSSL.

The TLS communication between Oracle HTTP Server and the target server succeeds only if Oracle HTTP Server trusts the remote server with which it communicates. To establish trust, update the trust store used by Oracle HTTP Server with the CA certificate of the remote server.

To add the CA certificate chain of the remote server to the trust store:

1. Obtain the CA certificate chain for the remote server.
2. Append the contents of certificate chain to the default trust store path.

The trust store path used by `libcurl` on Linux is `/etc/pki/tls/certs/ca-bundle.crt`.

On Windows:

1. Create a file named `curl-ca-bundle.crt`.
2. Place `curl-ca-bundle.crt` in either Windows System directory (for example, `C:\windows\system32`), or Windows Directory (for example, `C:\windows`), or any directory present in `%PATH%` (for example, `${`*PRODUCT_HOME*`}/bin`).
3. Add the CA certificate chain used to sign the certificate of the remote server to `curl-ca-bundle.crt`.

On Solaris x64 and Solaris SPARC platforms, the cURL library requires the CA certificates to be available at `/etc/pki/tls/certs/ca-bundle.crt`.

Starting version 12*c* (12.2.1.1.0), Oracle HTTP Server supports `mod_security` version 2.9.0 directives, variables, action, phases, and functions. See `http://www.modsecurity.org/documentation.html`.

Sample mod_security.conf File provides a usable example of the `mod_security.conf` file, including the `LoadModule` statement.

> **Note:**
>
> - `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` 2.9.0. Only documentation applicable to open source `mod_security` 2.9.0 is applicable to the Oracle HTTP Server implementation of the module.
> - In Oracle HTTP Server versions 11.1.1.7 and later, `mod_security` is not loaded or configured by default. However, if you have an installation patched from version 11.1.1.6, implementing the patch might have already loaded and configured the module.
> - Oracle supports the Oracle supplied version of `mod_security`. Newer versions from `modsecurity.org` is not supported.

The `mod_security` configuration can be added to the `httpd.conf` configuration file, or it can appear in a separate `mod_security.conf` configuration file.

This section contains the following information:

- Configuring mod_security in the httpd.conf File
- Configuring mod_security in a mod_security.conf File
- Configuring SecRemoteRules in the mod_security.conf File
- Sample mod_security.conf File
- Configuring mod_security in the httpd.conf File
- Configuring mod_security in a mod_security.conf File
- Configuring SecRemoteRules in the mod_security.conf File
  The **SecRemoteRules** is an optional directive that you can use to load rules from a remote server.
- Sample mod_security.conf File

Configuring mod_security in the httpd.conf File

You can configure the mod_security module by entering mod_security directives in the httpd.conf file in an `IfModule` container. To make the mod_security module available when Oracle HTTP Server is running, ensure that the mod_security configuration begins with the following lines:

```
...
#Load module
LoadModule security2_module "${PRODUCT_HOME}/modules/mod_security2.so"
...
```

Configuring mod_security in a mod_security.conf File

You can specify the mod_security directives in a separate mod_security.conf file and include that file in the httpd.conf file by using the `Include` directive.

1. You must create the mod_security.conf file yourself, preferably by using the template in Sample mod_security.conf File.

   Copy and paste the sample into a text editor, then edit it for your system.

2. To make the mod_security module available 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"
   ```

3. Save the file with the name "mod_security.conf" and include it in your httpd.conf file by using the `Include` directive.

   If you implement mod_security.conf file as described, it will use the `LoadModule` directive to load mod_security2.so into the run time environment.

Configuring SecRemoteRules in the mod_security.conf File

The **SecRemoteRules** is an optional directive that you can use to load rules from a remote server.

Syntax

```
SecRemoteRules some-key https://www.yourserver.com/plain-text-rules.txt
```

Table 5-2 provides information about the variables of SecRemoteRules.

Table 5-2 SecRemoteRules Variables

Variable	Description
some-key	These keys can be used by the target server to provide different content for different keys. You must provide these keys. Along with these keys, `mod_security` sends its unique ID and the status call in the format of headers to the target web server. The following headers are used: • ModSec-status • ModSec-unique-id • ModSec-key The optional option `crypto` tells `mod_security` to expect some encrypted content from server. The utilization of `SecRemoteRules` is only allowed over TLS. Thus, this option may not be necessary.

Table 5-2 (Cont.) SecRemoteRules Variables

Variable	Description
yourserver.com	*yourserver.com* is the remote server that hosts the `mod_security` rules. When the SecRemoteRules directive is configured on a server *S1*, *S1* establishes an SSL connection with *yourserver.com* to fetch the mod_security rules. Here, the `plain-text-rules.txt` file contains the mod_security rules. Server *S1* acts as an SSL client and *yourserver.com* acts as an SSL server. The SSL client is implemented using `libcurl`. By default, `libcurl` verifies the peer SSL certificate. The verification is done by using a CA certificate store that the SSL library can use to ensure that the peer's server certificate is valid. If the server uses a certificate signed by a CA that is not included in the store you use, add the CA certificate for your server to the existing default CA certificate store. The trust store path used by `libcurl` on Linux is `/etc/pki/tls/certs/ca-bundle.crt`. To add the remote server certificate to the trust store, do the following: 1. Extract the CA certificate for a particular server. If you use the **openssl** tool, you can do the following to extract the CA certificate for a particular server: a. `openssl s_client -connect xxxxx.com:443 |tee logfile` b. Type QUIT and press Enter. The certificate will have BEGIN CERTIFICATE and END CERTIFICATE markers. 2. Append the contents of certificate to the default trust store path. `/etc/pki/tls/certs/ca-bundle.crt` Ensure that you do not add a new line at the end of the file. `libcurl` also verifies server host name verification. That is, `libcurl` considers the server as the intended server when the Common Name field or a Subject Alternate Name field in the certificate matches the host name in the URL to which you told curl to connect. The communication might fail if this condition is not met.

Sample mod_security.conf File

The following code illustrates a sample mod_security.conf configuration file.

Example 5-1 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 minimizes 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
```

6

Configuring High Availability for Web Tier Components

Use the instructions in this chapter to configure an Oracle HTTP Server highly available deployment in which Oracle HTTP Servers and WebLogic Managed Servers reside on different hosts, behind a load balancer.

This chapter includes the following sections:

- Configuring Oracle HTTP Server for High Availability
  To configure an example high availability deployment of Oracle HTTP Server (OHS), you must meet specific prerequisites. You can then install OHS on an additional web server, then configure and validate OHS high availability.

Oracle HTTP Server Single-Instance Characteristics

Oracle HTTP Server (OHS) is based on Apache infrastructure and includes Oracle modules that you can use to extend OHS core functionality.

OHS has these components to handle client requests

- **HTTP listener** handles incoming requests and routes them to the appropriate processing utility.
- **Modules (mods)** implement and extend OHS functionality. OHS includes many standard Apache modules. Oracle also includes modules that are specific to OHS to support OHS and OHS component integration.

OHS can also be a proxy server, both forward and reverse. A reverse proxy enables content served by different servers to appear as if it comes from one server.

Oracle HTTP Server and Domains

Oracle HTTP Server (OHS) doesn't require a WebLogic domain but you usually use it with one. Oracle recommends associating OHS with a domain so that you can incorporate OHS into the Administration Console, where you can manage and monitor it.

The `mod_wl_ohs` module handles the link to Managed Servers. You configure `mod_wl_ohs` by routing requests of a particular type, such as JSPs, or by routing requests destined to a URL to specific Managed Servers.

OHS usually front ends a cluster. In this configuration, a special `mod_wl_ohs` directive, `WebLogicCluster`, specifies a comma-separated list of cluster members.

These steps describe the `mod_wl_ohs` directive process:

1. `mod_wl_ohs` receives a request for a Managed Server then sends the request to one cluster member in the directive. At least one Managed Server must be available to fulfill the request.
2. The Managed Server receives the request, processes it, and sends a complete list of cluster members back to `mod_wl_ohs`.
3. When `mod_wl_ohs` receives the updated list, it dynamically adds previously unknown servers to the known servers list. By doing this, all future requests are load balanced across the cluster member list. The benefit is that new Managed Servers are added to a cluster without updating `mod_wl_ohs` or adding OHS.

> **Note:**
>
> The `mod_wl_ohs` directive `DynamicServerList` controls whether or not unknown servers are added to the known servers list. You must set `DynamicServerList` to `ON` to enable dynamic addition of servers.

> **Note:**
>
> When you start, you don't need to include all current Managed Servers in the `mod_wl_ohs` directive. A high availability setup requires only two cluster members in the list for the first call to work. For more information about running an OHS high availability deployment, see Configuring the WebLogic Proxy Plug-In for Oracle HTTP Server in *Using Oracle WebLogic Server Proxy Plug-Ins*.
>
> For more information about Oracle WebLogic clusters, see Introduction and Roadmap in *Administering Clusters for Oracle WebLogic Server*.

Oracle HTTP Server Startup and Shutdown Lifecycle

After Oracle HTTP Server starts, it is ready to listen for and respond to HTTP(S) requests.

The request processing model is different on Microsoft Windows systems compared to UNIX systems:

- For Microsoft Windows, there is one parent process and one child process. The child process creates threads that handle client requests. The number of created threads is static and you can configure them for performance.
- For UNIX, there is one parent process that manages multiple child processes. Child processes handle requests. The parent process brings up more child processes as necessary, based on configuration.

> **Note:**
>
> For more information about the OHS processing model, see Oracle HTTP Server Processing Model.

Starting and Stopping Oracle HTTP Server

Use Fusion Middleware Control or the WebLogic Scripting Tool (WLST) to start, stop, and restart Oracle HTTP Server.

If you plan to use WLST, you should familiarize yourself with that tool; see Getting Started Using the Oracle WebLogic Scripting Tool (WLST) in the *Oracle Fusion Middleware Administrator's Guide*.

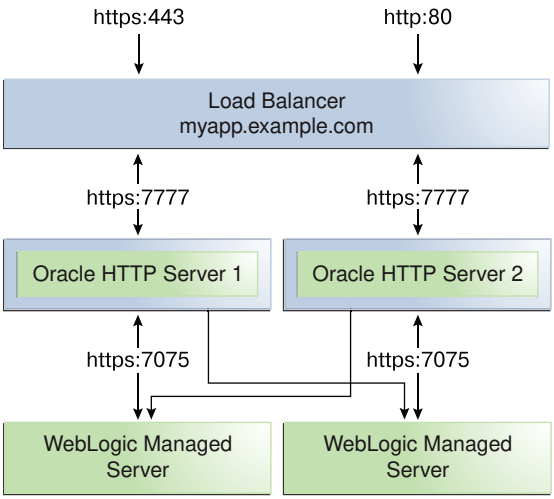
> **Note:**
>
> For more information about starting and stopping an OHS instance, see Performing Basic Oracle HTTP Server Tasks.

Oracle HTTP Server High Availability Architecture and Failover Considerations

Oracle HTTP Servers and Managed Servers reside on different hosts, behind a load balancer, in a high availability topology.

Figure 6-1 shows two Oracle HTTP Servers behind a load balancer.

Figure 6-1 Oracle HTTP Server High Availability Architecture



The load balancer receives user requests and forwards them to connected Oracle HTTP Servers. The load balancer receives requests on standard HTTP/HTTPS ports (80/443). However, it then passes requests to Oracle HTTP Servers using completely different ports. Advantages of this setup are:

- Actual ports are hidden from users.
- Users don't have to add port numbers to the URL.

On UNIX-based systems, starting OHS with root privileges isn't mandatory. Only root can start a process that uses a port less than 1024. However, for processes that use a port number below 1024, you must use root privilege to start a process.

The load balancer routes requests to the functioning Oracle HTTP Server.

Figure 6-1 also shows how OHS distributes requests to Managed Servers. For high availability, each pair of components (OHS and Managed Servers) should reside on different host computers. Managed Servers belong to the same cluster; to load balance across a set of Managed Servers, they must belong to the same cluster.

Oracle HTTP Server Failure Protection and Expected Behaviors

Oracle HTTP Server (OHS) has two failure types: **process failures** and **node failures**. An individual operating system process may fail. A node failure can involve failure of the entire host computer that OHS runs on.

Table 6-1 OHS Failure Types and Failure Protections

Failure Type	Protection
Process	Node Manager protects and manages OHS processes. If an OHS process fails, Node Manager automatically restarts it.
Node	Load balancer in front of OHS sends a request to another OHS if the first one doesn't respond or URL pings indicate it failed.
Managed Server	If a Managed Server in a cluster fails, `mod_wl_ohs` automatically redirects requests to one of the active cluster members. If the application stores state, state replication is enabled within the cluster, which enables redirected requests access to the same state information.
Database	Typically, an issue only when using `mod_oradav` or `mod_plsql`. With Oracle RAC databases, the Oracle RAC connection determines failure characteristics. **If client connection failover is configured**, in-flight transactions roll back. Database reconnection is required. **If Transparent Application Failover (TAF) is configured**, any in-flight database write rolls back but automatic database reconnection occurs and select statements recover automatically. TAF fails over select statements only; package variables are lost. TAF, a JDBC Oracle Call Interface driver feature, enables an application to automatically reconnect to a database if the database instance the connection is made to fails. In this case, active transactions roll back.

Configuring Oracle HTTP Server Instances on Multiple Machines

If you use the Configuration Wizard to configure Oracle HTTP Server (OHS) and OHS is part of a domain, update the `mod_wl_ohs.conf` file for each instance.

The file is in the *`DOMAIN_HOME`*`/config/fmwconfig/components/OHS/`*`componentName`* directory. Restart the Administration Server to propagate changes to all OHS instances in the domain, even if they reside on a different host. See Configuring mod_wl_ohs.conf.

> **Note:**
>
> If you install and configure OHS instances in separate domains, you must manually copy changes to other Oracle HTTP Servers. You must verify that the changes apply to all OHS instances and that they are synchronized.

Configuring Oracle HTTP Server for High Availability

To configure an example high availability deployment of Oracle HTTP Server (OHS), you must meet specific prerequisites. You can then install OHS on an additional web server, then configure and validate OHS high availability.

- Prerequisites to Configure a Highly Available OHS
  Complete the following prerequisites before configuring a highly available Oracle HTTP Server deployment:
- Installing and Validating Oracle HTTP Server on WEBHOST2
- Configuring and Validating an OHS High Availability Deployment
  To configure and validate the OHS high availability deployment, update `mod_wl_ohs.conf` and then use test URLs to validate OHS configuration.

Prerequisites to Configure a Highly Available OHS

Complete the following prerequisites before configuring a highly available Oracle HTTP Server deployment:

- Load Balancer Prerequisites
- Configuring Load Balancer Virtual Server Names and Ports
- Managing Load Balancer Port Numbers
- Installing and Validating Oracle HTTP Server on WEBHOST1
- Creating Virtual Host(s) on WEBHOST1
- Configuring mod_wl_ohs.conf
- Configuring mod_wl_conf if you use SSL Termination
- Creating proxy.conf File
- Load Balancer Prerequisites
  To distribute requests against Oracle HTTP Server, you can either use an external load balancer to distribute HTTP(S) requests between available Oracle HTTP Servers, or configure Oracle HTTP Server VirtualHost proxy for load balancing.
- Configuring Load Balancer Virtual Server Names and Ports
  In an OHS installation, virtual servers are configured for HTTP connections, which are distributed across the HTTP servers.
- Managing Load Balancer Port Numbers
  Many Oracle Fusion Middleware components and services use ports. As an administrator, you must know the port numbers that services use and ensure that two services don't use the same port number on your host computer.
- Installing and Validating Oracle HTTP Server on WEBHOST1
- Creating Virtual Host(s) on WEBHOST1
  For each virtual host or site name that you use, add an entry to the OHS configuration.
- Configuring mod_wl_ohs.conf
  After you install and configure OHS, link it to any defined Managed Servers by editing the `mod_wl_ohs.conf` file.

- Configuring mod_wl_conf if you use SSL Termination
  If you use SSL termination AND route requests to WebLogic, you must take additional configuration steps.
- Creating proxy.conf File
  If you are not using an external load balancer, you can configure Oracle HTTP Server virtual host proxy for load balancing.

Load Balancer Prerequisites

To distribute requests against Oracle HTTP Server, you can either use an external load balancer to distribute HTTP(S) requests between available Oracle HTTP Servers, or configure Oracle HTTP Server VirtualHost proxy for load balancing.

If you have an external load balancer, it must have features that Third-Party Load Balancer Requirements describes.

If you want to configure Oracle HTTP Server for load balancing, use virtual host based proxy configuration. To do this, create a separate configuration file, for example, `proxy.conf` with the configuration details, and append this configuration into `httpd.conf` using `include` tag. For example, `include "proxy.conf"`. For more information, see Creating proxy.conf File.

Configuring Load Balancer Virtual Server Names and Ports

In an OHS installation, virtual servers are configured for HTTP connections, which are distributed across the HTTP servers.

If your site serves requests for HTTP and HTTPS connections, Oracle recommends that HTTPS requests terminate at the load balancer and pass through as HTTP requests. To do this, the load balancer should be able to perform the protocol conversion and must be configured for persistent HTTP sessions.

This example configuration assumes that the load balancer is configured as:

- **Virtual Host:** `Myapp.example.com`
- **Virtual Port:** `7777`
- **Server Pool:** `Map`
- **Server:** `WEBHOST1`, `Port 7777`, `WEBHOST2`, `Port 7777`

Managing Load Balancer Port Numbers

Many Oracle Fusion Middleware components and services use ports. As an administrator, you must know the port numbers that services use and ensure that two services don't use the same port number on your host computer.

Most port numbers are assigned during installation. It is important that any traffic going from Oracle HTTP Servers to Oracle WebLogic Servers has access through any firewalls.

Installing and Validating Oracle HTTP Server on WEBHOST1

To install Oracle HTTP Server on WEBHOST1, see the steps in Installing the Oracle HTTP Server Software in *Installing and Configuring Oracle HTTP Server*.

Validate the installation using the following URL to access the OHS home page:

```
http://webhost1:7777/
```

Creating Virtual Host(s) on WEBHOST1

For each virtual host or site name that you use, add an entry to the OHS configuration.

Create a file named `virtual_hosts.conf` in the `DOMAIN_HOME/config/fmwconfig/components/OHS/ohs_component_name/moduleconf` directory as follows:

```
NameVirtualHost *:7777
<VirtualHost *:7777>
 ServerName http://myapp.example.com:80
 RewriteEngine On
 RewriteOptions inherit
 UseCanonicalName On
</VirtualHost>
```

If you are using SSL/SSL Termination (*):

```
NameVirtualHost *:7777
<VirtualHost *:7777>
 ServerName https://myapp.example.com:443
 RewriteEngine On
 RewriteOptions inherit
 UseCanonicalName On
</VirtualHost>
```

> **Note:**
>
> You can also use Fusion Middleware Control to create virtual hosts. See Wiring Components Together in *Administering Oracle Fusion Middleware*.

Configuring mod_wl_ohs.conf

After you install and configure OHS, link it to any defined Managed Servers by editing the `mod_wl_ohs.conf` file.

The file is in *`DOMAIN_HOME`*`/config/fmwconfig/components/OHS/`*`componentName`* directory.

For more information about editing the `mod_wl_ohs.conf` file, see Configuring the WebLogic Proxy Plug-In for Oracle HTTP Server in *Oracle Fusion Middleware Using Oracle WebLogic Server Proxy Plug-Ins* 12.1.2.

> **Note:**
>
> You can also use Fusion Middleware Control to link OHS to Managed Servers. See Wiring Components Together in *Administering Oracle Fusion Middleware*.

The following example shows `mod_wl_ohs.conf` entries:

```
LoadModule weblogic_module PRODUCT_HOME/modules/mod_wl_ohs.so

<IfModule mod_weblogic.c>
 WebLogicCluster apphost1.example.com:7050, apphost2.example.com:7050
 MatchExpression *.jsp
 </IfModule>

<Location /weblogic>
 SetHandler weblogic-handler
 WebLogicCluster apphost1.example.com:7050,apphost2.example.com:7050
 DefaultFileName index.jsp
</Location>

<Location /console>
 SetHandler weblogic-handler
 WebLogicCluster apphost1.example.com
 WebLogicPort 7003
</Location>
```

These examples show two different ways to route requests to Managed Servers:

- The `<ifModule>` block sends any requests ending in `*.jsp` to the WebLogic Managed Server cluster located on APPHOST1 and APPHOST2.
- The `<Location>` block sends any requests with URLs that have a `/weblogic` prefix to the Managed Server cluster located on APPHOST1 and APPHOST2.

Configuring mod_wl_conf if you use SSL Termination

If you use SSL termination AND route requests to WebLogic, you must take additional configuration steps.

To configure mod_wl_conf if you use SSL termination:

1. In the WebLogic console, verify that WebLogic Plugin Enabled is set to true, either at the domain, cluster, or Managed Server level.
2. Add these lines to the Location block, which directs requests to Managed Servers:

   ```
   WLProxySSL ON
   WLProxySSLPassThrough ON
   ```

For example:

```
<Location /weblogic>
 SetHandler weblogic-handler
 WebLogicCluster apphost1.example.com:7050,apphost2.example.com:7050
 WLProxySSL On
 WLProxySSLPassThrough ON
 DefaultFileName index.jsp
</Location>
```

After you enable the WebLogic plugin, restart the Administration Server.

Creating proxy.conf File

If you are not using an external load balancer, you can configure Oracle HTTP Server virtual host proxy for load balancing.

To do this, create a separate configuration file, for example, `proxy.conf` at the following locations:

- *MW_HOME*/user_projects/domains/*DOMAIN_NAME*/config/fmwconfig/components/OHS/*INSTANCE_NAME*/
- *MW_HOME*/user_projects/domains/*DOMAIN_NAME*/config/fmwconfig/components/OHS/instances/*INSTANCE_NAME*/

Add the following snippet to the newly created configuration file (at both the locations):

```
LoadModule lbmethod_byrequests_module
"${PRODUCT_HOME}/modules/mod_lbmethod_byrequests.so"

<VirtualHost myohs.com:8080>

   # Proxy : HTTP - HTTP
   <Proxy "balancer://ha">
      ProxySet lbmethod=byrequests
      BalancerMember http://OHS_Host1:Port
      BalancerMember http://OHS_Host1:Port
   </Proxy>

   # Proxy pass : HTTP - HTTP
   ProxyPass        "/ha"   "balancer://ha"
   ProxyPassReverse "/ha"   "balancer://ha"

</VirtualHost>
```

Append this configuration file (`proxy.conf`) into `httpd.conf` file.

Installing and Validating Oracle HTTP Server on WEBHOST2

To install Oracle HTTP Server on WEBHOST2, see Installing the Oracle HTTP Server Software in *Oracle Fusion Middleware Installing and Configuring Oracle HTTP Server*.

Validate the installation on WEBHOST2 by using the following URL to access the Oracle HTTP Server home page:

```
http://webhost2:7777/
```

Configuring and Validating an OHS High Availability Deployment

To configure and validate the OHS high availability deployment, update `mod_wl_ohs.conf` and then use test URLs to validate OHS configuration.

- Configuring Virtual Host(s) on WEBHOST2
  Update the `mod_wl_ohs.conf` file located in *DOMAIN_HOME*`/config/fmwconfig/components/OHS/`*componentName* directory.
- Validating the Oracle HTTP Server Configuration
  You validate the OHS configuration by using specific URLs.

Configuring Virtual Host(s) on WEBHOST2

Update the `mod_wl_ohs.conf` file located in *DOMAIN_HOME*`/config/fmwconfig/components/OHS/`*componentName* directory.

You must then restart the Administration Server to propagate changes to all OHS instances in the domain.

Validating the Oracle HTTP Server Configuration

You validate the OHS configuration by using specific URLs.

`http://myapp.example.com/`

`https://myapp.example.com` (if using SSL/SSL termination)

`http://myapp.example.com:7777/weblogic`

7

Managing and Monitoring Server Processes

You have tools and procedures that help to manage and monitor the performance of Oracle HTTP Server.

This chapter includes the following sections. These sections discuss the procedures and tools that manage the server in your environment.

- Oracle HTTP Server Processing Model
- Monitoring Server Performance
- Oracle HTTP Server Performance Directives
- Understanding Process Security for UNIX
- Oracle HTTP Server Processing Model
  There are two types of processing models that help to monitor Oracle HTTP Server: Request Process Model and Single Unit Process Model.
- Monitoring Server Performance
  Oracle Fusion Middleware automatically and continuously measures runtime performance for Oracle HTTP Server and Oracle WebLogic Server proxy plug-in module.
- Oracle HTTP Server Performance Directives
  Oracle HTTP Server performance is managed by directives specified in the configuration files. Use Fusion Middleware Control to tune performance-related directives for Oracle HTTP Server.
- Understanding Process Security for UNIX
  Special configuration is required to allow Oracle HTTP Server to bind to privileged ports when installed on UNIX.

Oracle HTTP Server Processing Model

There are two types of processing models that help to monitor Oracle HTTP Server: Request Process Model and Single Unit Process Model.

The following sections describe the processing models for Oracle HTTP Server.

- Request Process Model
- Single Unit Process Model
- Request Process Model
- Single Unit Process Model

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 can dynamically start 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.

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.

Monitoring Server Performance

Oracle Fusion Middleware automatically and continuously measures runtime performance for Oracle HTTP Server and Oracle WebLogic Server proxy plug-in module.

The server 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 hanging, you can view the metrics to find out more about the problem. Fusion Middleware Control provides real-time data. Cloud Control can be used to view historical data.

These sections describe performance metrics and how to view them:

- Oracle HTTP Server Performance Metrics
- Viewing Performance Metrics
- Oracle HTTP Server Performance Metrics
- Viewing Performance Metrics
  You can view the performance metrics of the Oracle HTTP Server and Oracle WebLogic Server Proxy Plug-In module by using the Fusion Middleware Control or issuing the appropriate WLST command. View performance metrics to monitor and analyze the server performance.

Oracle HTTP Server Performance Metrics

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

Oracle HTTP Server Metrics

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

Oracle HTTP Server Virtual Host Metrics

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

Oracle HTTP Server Module Metrics

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

Viewing Performance Metrics

You can view the performance metrics of the Oracle HTTP Server and Oracle WebLogic Server Proxy Plug-In module by using the Fusion Middleware Control or issuing the appropriate WLST command. View performance metrics to monitor and analyze the server performance.

You can view Oracle HTTP Server and Oracle WebLogic Server Proxy Plug-In module performance metrics by using the procedures described in the following sections:

- Viewing Server Metrics by Using Fusion Middleware Control
- Viewing Server Metrics Using WLST
- Viewing Server Metrics by Using Fusion Middleware Control
- Viewing Server Metrics Using WLST

Viewing Server Metrics by Using Fusion Middleware Control

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.
2. From the Oracle HTTP Server menu on the Oracle HTTP Server home page, 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.
4. Select additional metrics to add them to the Performance Summary.

Viewing Server Metrics Using WLST

To obtain and view metrics for an instance from the command line, you must connect to, and issue the appropriate WLST command. These commands allow you to perform any of these functions:

- Display Metric Table Names
- Display metric tables
- Dump metrics

> **Note:**
>
> For more information on using WLST, see *Understanding the WebLogic Scripting Tool*.

Before attempting this procedure:

Before attempting to access server metrics from the command line, ensure the following:

- The domain exists and the instance for which you want to see the metrics exist.
- The instance is running.
- Node Manager is running on the instance machine.

The Administration server can be running, but this is not required.

To view metrics using WLST:

> **Note:**
>
> In both managed and standalone domain types, the following procedure will work whether you run the commands from the same machine or from a machine that is remote to the server.

1. Launch WLST:

   On Linux or UNIX:

   ```
   $ORACLE_HOME/oracle_common/common/bin/wlst.sh
   ```

   On Windows:

   ```
   $ORACLE_HOME\oracle_common\common\bin\wlst.cmd
   ```

2. From the selected domain directory (for example, *ORACLE_HOME*/user_projects/domains/*domainName*), connect to the instance:

   ```
   nmConnect('username', 'password', nm_host, nm_port, domainName)
   ```

3. Enter one of the following WLST commands, depending on what task you want to accomplish:

   - `displayMetricTableNames(servers=['serverName'], servertype='serverType')`
   - `displayMetricTables(servers=['serverName'], servertype='serverType')`
   - `dumpMetrics(servers=['serverName'], servertype='serverType')`

   For example:

   ```
   displayMetricTableNames(servers=['ohs1'], servertype='OHS')
   displayMetricTables(servers=['ohs1'], servertype='OHS')
   dumpMetrics(servers=['ohs1'], servertype='OHS')
   ```

Oracle HTTP Server Performance Directives

Oracle HTTP Server performance is managed by directives specified in the configuration files. Use Fusion Middleware Control to tune performance-related directives for Oracle HTTP Server.

The following sections describe the Oracle HTTP Server performance directives.

- Understanding Performance Directives
- Configuring Performance Directives by Using Fusion Middleware Control

Understanding Performance Directives

Oracle HTTP Server uses directives declared in httpd.conf and other configuration files. 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 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 in UNIX (non-Linux) environments. This MPM implements a hybrid multi-process multi-threaded server. By using threads to serve requests, it can serve many 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. If you are using Worker MPM, then you must configure the mod_cgid module for your CGI applications instead of the mod_cgi module. For more information, see the following URL:

  http://httpd.apache.org/docs/2.4/mod/mod_cgid.html

- 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. If you are going to implement a CGI module with this MPM, use only mod_fastcgi.
- Event: This is the default MPM for Oracle HTTP Server in Linux environments. This MPM is designed to allow more requests to be served simultaneously by passing off some processing work to supporting threads, freeing up the main threads to work on new requests. It is based on the Worker MPM, which implements a hybrid multi-process multi-threaded server. Run-time configuration directives are identical to those provided by Worker.

The following sections describe how to change the MPM type value for an Oracle HTTP Server instance in a standalone and an Oracle WebLogic Server domain

Changing the MPM Type Value in a Standalone Domain

To change the MPM type value for an Oracle HTTP Server instance in a standalone domain, follow these steps:

1. Navigate to the `ohs.plugins.nodemanager.properties` file at the following location: `${ORACLE_INSTANCE}/config/fmwconfig/components/OHS/${COMPONENT_NAME}`.
2. Edit the `ohs.plugins.nodemanager.properties` file to make the following changes.

   Look for the key `mpm` in an uncommented line.

   - If you find the key in an uncommented line, then replace the existing value of `mpm` with the value you want to set for MPM.
   - If you do not find it in an uncommented line, then add a new line to the file using the following format:

     ```
     mpm = mpm_value
     ```

     where *`mpm_value`* is the value you want to set as MPM.
3. Start or re-start the Oracle HTTP Server instance.

Changing the MPM Type Value in a WebLogic Server Managed Domain

To change the MPM type value for an Oracle HTTP Server instance in an Oracle WebLogic Server domain, follow these steps.

> **Note:**
>
> The following steps assume that the Administration Server and Node Manager for the domain are already up and running.

1. Launch WLST from the command line.

   Linux or UNIX: `$ORACLE_HOME/oracle_common/common/bin/wlst.sh`
2. Connect to the Administration Server instance:

   ```
   connect('<userName', '<password>', '<host>:<port>')
   ```
3. Navigate to the Mbean containing the MPM type value key.

   You can use the `editCustom()` command only when WLST is connected to the Administration Server. Use `cd` to navigate the hierarchy of management objects. This example assumes that Oracle HTTP Server instance with name '`ohs1`'.

   ```
   editCustom()
   cd('oracle.ohs')
   cd('oracle.ohs:type=OHSInstance.NMProp,OHSInstance=ohs1,component=OHS')
   ```

4. Set the MPM type value key.

   Start an edit session and set the MPM type value key `Mpm` to the type value. In this example the type value is set to `event`.

```
startEdit()
set('Mpm','event')
save()
activate()
```

Configuring Performance Directives by Using Fusion Middleware Control

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

Use the Performance Directives page of Fusion Middleware Control to tune performance-related directives for Oracle HTTP Server.

Performance directives management consists of these areas: request, connection, and process configuration. The following sections describe how to set these configurations.

- Setting the Request Configuration by Using Fusion Middleware Control
- Setting the Connection Configuration by Using Fusion Middleware Control
- Setting the Process Configuration by Using Fusion Middleware Control

- Setting the Request Configuration by Using Fusion Middleware Control
- Setting the Connection Configuration by Using Fusion Middleware Control
- Setting the Process Configuration by Using Fusion Middleware Control

Setting the Request Configuration by Using Fusion Middleware Control

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 (`MaxRequestWorkers` directive).

   This setting limits the number of requests that can be dealt with simultaneously. The default value is 400. This is applicable for all Linux/UNIX platforms.
4. Set the maximum requests per child process in the Maximum Request per Child Process field (`MaxConnectionsPerChild` 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 Restarting Oracle HTTP Server Instances .

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

Setting the Connection Configuration by Using Fusion Middleware Control

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 5 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 Restarting Oracle HTTP Server Instances .

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

Setting the Process Configuration by Using Fusion Middleware Control

The child process and configuration settings impact the ability of the server to process requests. You might 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 3. 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 3. 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 Restarting Oracle HTTP Server Instances .

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

Understanding Process Security for UNIX

Special configuration is required to allow Oracle HTTP Server to bind to privileged ports when installed on UNIX.

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

8

Managing Connectivity

You can manage and monitor the performance of Oracle HTTP Server connectivity by creating ports, viewing port number usage, and configuring virtual hosts.

This chapter includes the following sections which describes the procedures for managing Oracle HTTP Server connectivity:

- Default Listen Ports
- Defining the Admin Port
- Viewing Port Number Usage
- Managing Ports
- Configuring Virtual Hosts

- Default Listen Ports
  Listen ports (SSL and non-SSL) have a default range of port numbers.
- Defining the Admin Port
  Admin port is used internally by Oracle HTTP Server to communicate with Node Manager. This port is configured in the `admin.conf` file.
- Viewing Port Number Usage
  You can view ports using Fusion Middleware Control or WLST.
- 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.
- 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 the virtual ports run on the same physical server is not apparent to the end user.

Default Listen Ports

Listen ports (SSL and non-SSL) have a default range of port numbers.

Automatic port assignment occurs only if you use `ohs_createInstance()` or Fusion Middleware Control. The default, non-SSL port is 7777. If port 7777 is occupied, the next available port number, within a range of 7777-65535, is assigned. The default SSL port is 4443. Similarly, if port 4443 is occupied, the next available port number, within a range of 4443-65535, is assigned.

If you create instances using Configuration Wizard, then you must perform your own port management. The Configuration Wizard has no automatic port assignment capabilities.

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

Defining the Admin Port

Admin port is used internally by Oracle HTTP Server to communicate with Node Manager. This port is configured in the `admin.conf` file.

The communication between Node Manager and admin port happens over SSL, by default. It is possible to use plain-text communication by disabling SSL on the admin port, however it's not recommended. If SSL is disabled, a warning message indicating plain-text communication is logged to the console and the `ohs_nm.log` during Oracle HTTP Server start-up, and then the OHS starts successfully. The following is the sample warning message:

```
"SSL is not enabled for the admin port of 'ohs1'. Thus,
the connection between NodeManager and the admin port of 'ohs1' is not
secure. SSL must be enabled for this connection. For more
information on how to enable SSL for this connection, refer to OHS
documentation".
```

See Configuring SSL for Admin Port.

Automatic Admin port assignment occurs only if you use `ohs_createInstance()` or Fusion Middleware Control. If you create instances using Configuration Wizard, then you must perform your own Admin port management. The Configuration Wizard has no automatic port assignment capabilities.

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.

Viewing Port Number Usage

You can view ports using Fusion Middleware Control or WLST.

This section includes the following topics:

- Viewing Port Number Usage by Using Fusion Middleware Control
- Viewing Port Number Usage Using WLST
- Viewing Port Number Usage by Using Fusion Middleware Control
- Viewing Port Number Usage Using WLST

Viewing Port Number Usage by Using Fusion Middleware Control

You can view how ports are assigned on the Fusion Middleware Control Port Usage detail page. 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.

Viewing Port Number Usage Using WLST

If you are using Oracle HTTP Server in collocated mode, then you can use WLST commands to view the port number information on a given instance.

1. Launch WLST:

   ```
   $ORACLE_HOME/oracle_common/common/bin/wlst.sh
   ```

2. Connect to the AdminServer.
3. Use the `editCustom()` command to navigate to the root of the `oracle.ohs` MBean. You can use the `editCustom()` command only when WLST is connected to the Administration Server. Use `cd` to navigate the hierarchy of management objects, then `get()` to get the value of the `Ports` parameter:

   ```
   editCustom()
   cd('oracle.ohs')
   cd('oracle.ohs:type=OHSInstance,name=ohs1')
   get('Ports')
   ```

WLST will return a value similar to the following:

```
array(java.lang.String,['7777', '4443', '127.0.0.1:9999'])
```

> **Note:**
>
> On Unix, you can also `cd` into the directory of the master copy of the Oracle HTTP Server configuration files and do a `grep` for the Listen directives.

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 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*.

This section includes the following topics:

- Creating Ports Using Fusion Middleware Control
- Editing Ports Using Fusion Middleware Control

> **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.

- Creating Ports Using Fusion Middleware Control
- Editing Ports Using Fusion Middleware Control
- Disabling a Listening Port in a Standalone Environment

Creating Ports Using Fusion Middleware Control

You create a port for an Oracle HTTP Server endpoint on the Fusion Middleware Control Create port page. 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.

   You can configure SSL for a port on the Virtual Hosts page, as described in Configuring Virtual Hosts Using Fusion Middleware Control.
6. In **Port**, enter the port number.
7. Click **OK**.
8. Restart Oracle HTTP Server. See Restarting Oracle HTTP Server Instances .

> **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. See Registering Oracle HTTP Server mod_osso with OSSO Server.

Editing Ports Using Fusion Middleware Control

You can edit the values for existing ports on the Fusion Middleware Control Edit Port page. To edit 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 Node Manager, 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.

   You can be configure SSL for a port on the Virtual Hosts page as described in Configuring Virtual Hosts Using Fusion Middleware Control.
7. Click **OK**.
8. Restart Oracle HTTP Server. See Restarting Oracle HTTP Server Instances .

> **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.

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 staging configuration file 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. See Understanding Configuration Files.

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 the virtual ports run on the same physical server is not apparent to the end user.

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

The current release of Oracle HTTP Server enables you to use IPv6 and IPv4 addresses as the virtual host name.

You can also configure multiple addresses for the same virtual host; that is, a virtual host can be configured to serve on multiple addresses. This allows requests to different addresses to be served with the same content from the same virtual host.

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

- Creating Virtual Hosts Using Fusion Middleware Control
- Configuring Virtual Hosts Using Fusion Middleware Control

- Creating Virtual Hosts Using Fusion Middleware Control
- Configuring Virtual Hosts Using Fusion Middleware Control

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

Creating Virtual Hosts Using Fusion Middleware Control

You can create a virtual host for Oracle HTTP Server on the Fusion Middleware Control Create Virtual Hosts page. 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. In the Create Virtual Host screen, 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, IP address, or IPv6 address, for example mymachine.com:8080. This will create the following VirtualHost directive:

     ```
     <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>
```

> **Note:**
>
> If you attempt to create a virtual host with a wildcard character, for example, `*:port` and no listen directive exists for that port, then the virtual host creation will fail.
>
> In this case, you must first add the listen directive, and then try to add the virtual host.

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
7. Click **OK**.
8. Restart Oracle HTTP Server. See Restarting Oracle HTTP Server Instances .

Removing Unnecessary Listen Directives

Creating a virtual host by using Fusion Middleware Control also adds the Listen directive for the virtual host. However, virtual host creation will add unnecessary Listen directives in the following situations:

- A virtual host is being created for one host name and the Listen directive already exists for the different host name resolving to the same IP address.
- A virtual host is being created for one host name and the Listen directive already exists for the IP address that the host name resolves to.
- A virtual host is being created for multiple host names that resolve to the same IP address.

In these situations, Oracle HTTP Server will fail to start because there are multiple Listen directives for the same IP address. You must remove any extra Listen directives configured for the same IP address.

Configuring Virtual Hosts Using Fusion Middleware Control

You can use the options on the Configure menu of the Virtual Hosts page to specify Server, MIME, Log, 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 the 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 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. Configuring MIME Settings Using Fusion Middleware Control.
   - **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 Configuring Oracle HTTP Server Logs.
   - **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 About 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 Restarting Oracle HTTP Server Instances .

9

Managing Oracle HTTP Server Logs

Managing Oracle HTTP Server logs includes configuring the server logs, viewing the cause of an error and its corrective action, and more.

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:

- Overview of Server Logs
  Oracle HTTP Server has two types of server logs: error logs and access logs. Error log files record server problems, and access log files record details of components and applications being accessed and by whom.
- Configuring Oracle HTTP Server Logs
  You can use Fusion Middleware Control to configure error and access logs.
- Configuring the Log Level Using WLST
  You can use WLST commands to set the LogLevel directive, which controls the verbosity of the error log.
- Log Directives for Oracle HTTP Server
  Oracle HTTP Server can be configured to use either Oracle Diagnostic Logging (ODL) for generating diagnostic messages or the legacy Apache HTTP Server message format.
- Viewing Oracle HTTP Server Logs
  You can view server logs using Fusion Middleware Control, WLST, or a text editor.
- Recording ECID Information
  You can configure Oracle HTTP Server logs to record Execution Context ID (ECID) information.

Overview of Server Logs

Oracle HTTP Server has two types of server logs: error logs and access logs. Error log files record server problems, and access log files record details of components and applications being accessed and by whom.

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

This section contains the following topics:

- About Error Logs
- About Access Logs
- Configuring Log Rotation

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.

> **Note:**
>
> ODL error logging cannot have separate log files for each virtual host. It can only be configured globally for all virtual hosts.

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).

```
LogFormat "%h %l %u %t %E \"%r\" %>s %b" common
```

The CLF format contains the following fields:

```
host ident remote_logname remote_usre date  ECID request authuser 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.
- `remote_logname`: Remote log name (from `identd`, if supplied). Use `%l` to specify the remote log name in the log.
- `remote_user`: Remote user if the request was authenticated. Use `%u` to specify the remote user 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.
- `ECID`: Capture ECID information. Use `%E` to capture ECID in the log. See also Configuring Access Logs for ECID Information.
- `request`: This is the request line, in double quotes, from the client. Use `%r` to specify request in the log.
- `authuser`: This is the user ID for the authorized user. Use `%a` to specify the authorized user field 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.

Configuring 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/2.4/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/2.4/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 9-1 describes the size- and time-based rotation options:

Table 9-1 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.
`maxFileSize`	The maximum size (in MB) of log files.
`allFileSize`	The total size (in MB) of files retained.

With time-based rotation, log rotation of Oracle HTTP Server using the `odl_rotatelogs` is calculated by default according to UTC time. For example, setting log rotation to 86400 (24 hours) rotates the logs every 12:00 midnight, UTC. If Oracle HTTP Server is running on a server in IST (Indian Standard Time) which is UTC+05:30, then the logs are rotated at 05:30 a.m.

As an alternative to using the `-u` option with the UTC offset, you can use the `-l` option provided by Apache. This option causes Oracle HTTP Server to use local time as the base for the interval. Using the`-l` option in an environment which changes the UTC

offset (such as British Standard Time (BST) or Daylight Savings Time (DST)) can lead to unpredictable results.

- Syntax and Examples for Time- and Size-Based Log Rotation

Syntax and Examples for Time- and Size-Based Log Rotation

The following examples demonstrate the `odl_rotatelogs` syntax to set time- and size-based log 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
  ```

Configuring Oracle HTTP Server Logs

You can use Fusion Middleware Control to configure error and access logs.

The following sections describe logging tasks that can be set from the Log Configuration page:

- Configuring Error Logs Using Fusion Middleware Control

Configuring Error Logs Using Fusion Middleware Control

You configure error logs on the Fusion Middleware Control Log Configuration page. 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.
3. Set up the following error log configuration tasks on this page:
   - Configuring the Error Log Format and Location
   - Configuring the Error Log Level
   - Configuring Error Log Rotation Policy

Configuring the Error Log Format and Location

You can change the error log format and location on the Fusion Middleware Control Log Configuration page. By default, Oracle HTTP Server 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.
   - ODL-Text: the format of the diagnostic messages conform to an Oracle standard and are written in text 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 Restarting Oracle HTTP Server Instances .

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 Restarting Oracle HTTP Server Instances .

> **Note:**
>
> The log levels are different for the Apache HTTP Server log format and ODL-Text format.
>
> - For details on ODL log levels, refer to Setting the Level of Information Written to Log Filesin *Administering Oracle Fusion Middleware*.
> - For details on Apache HTTP Server log levels, refer to the LogLevel Directive in the Apache HTTP Server documentation.

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 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 Restarting Oracle HTTP Server Instances .

Configuring Access Logs Using Fusion Middleware Control

You can configure an access log format and rotation policy for Oracle HTTP Server from the Fusion Middleware Control Log Configuration page.

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

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

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. Navigate to the Oracle HTTP Server home page.
2. Select **Log Configuration** from the Administration menu.
3. From the Log Configuration page, navigate to the Access Log section.
4. Click **Manage Log Formats**.

   The Manage Custom Access Log Formats page is displayed.
5. Select an existing format to change or remove, or click **Add Row** to create a new format.
6. 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.

   For information about log format directives, see Apache HTTP Server documentation.
7. Click **OK** to save the new format.

Configuring the Access Log File

You can configure rotation policy for the access log on the Fusion Middleware Control Create or Edit Access Log page. To configure an access log for file Oracle HTTP Server, do the following:

1. Navigate to the Oracle HTTP Server home page.
2. Select **Log Configuration** from the Administration menu.
3. From the Log Configuration page, navigate to the Access Log section.
4. 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.
5. Enter the path for the access log in the **Log File Path** field. This directory must exist before you enter it.
6. Select an existing access log format from the **Log Format** menu.

7. 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.
8. Click **OK** to continue.

   You can create multiple access log files.

Configuring the Log File Creation Mode (umask) (UNIX/Linux Only)

Set the value of default file mode creation mask (`umask`) before starting the Oracle HTTP Server instance. The value that you set for umask determines the file permissions for the files created by Oracle HTTP Server instance such as the error log, access log, and so on. If `umask` is not set explicitly, then a value of `0027` is used by default.

This section contains the following information:

Configure umask for an Oracle HTTP Server Instance in a Standalone Domain

To configure the default file mode creation mask in a standalone domain, set the `umask` property in the `ohs.plugins.nodemanager.properties` file under the staging location:

DOMAIN_HOME`/config/fmwconfig/components/OHS/`*instanceName*`/ohs.plugins.nodemanager.properties`

Configure umask for an Oracle HTTP Server Instance in a WebLogic Server Managed Domain

To configure the default file mode creation mask in a WebLogic Server (either Full-JRF or Restricted-JRF) domain, follow these steps:

1. Start the AdminServer and NodeManager for the domain, for example:

   ```
   <Domain_HOME>/bin/startWebLogic.sh &
   <DOMAIN_HOME>/bin/startNodeManager.sh &
   ```

2. Start WLST and connect to the AdminServer.

   ```
   <ORACLE_HOME>/oracle_common/bin/wlst.sh
   connect('<userName', <'password'>, <'adminServerURL'>
   ```

3. Navigate to the following MBean. Note that the ObjectName for this MBean is dependent on the name of Oracle HTTP Server instance. In this example, the name of Oracle HTTP Server instance is `ohs1`

```
editCustom()
cd('oracle.ohs')
cd('oracle.ohs:OHSInstance=ohs1,component=OHS,type=OHSInstance.NMProp')
```

4. Set the value of `umask` to the desired value.

```
startEdit()
set('Umask','0022')
```

5. Save and activate the changes.

```
save()
activate()
```

Configuring the Log Level Using WLST

You can use WLST commands to set the LogLevel directive, which controls the verbosity of the error log.

For more information on the LogLevel directive, see the Apache documentation: http://httpd.apache.org/docs/current/mod/core.html#loglevel

Follow these steps to set the LogLevel directive using WLST commands.

1. Launch WLST.

```
$ORACLE_HOME/oracle_common/common/bin/wlst.sh
```

2. Connect to Administration Server.

```
connect('<user-name>', '<password>','<host>:<port>')
```

3. Use the `editCustom()` command to navigate to the root of the `oracle.ohs` MBean. You can use the `editCustom()` command only when WLST is connected to the Administration Server. Use `cd` to navigate the hierarchy of management objects, in this case, `ohs1` under `oracle.ohs`. Use the `startEdit()` command to start an edit session.

```
editCustom()
cd('oracle.ohs')
cd('oracle.ohs:type=OHSInstance,name=ohs1')
startEdit()
```

4. Use the `set` command to set the value of the log level attribute. The following example sets the global log level to `trace7`, the module `status` log level to `error`, and the module `env` log level to `warn` (warning).

```
set('LogLevel','trace7 status:error env:warn')
```

5. Save, then activate your changes. The edit lock associated with this edit session is released once the activation is completed.

```
save()
activate()
```

Log Directives for Oracle HTTP Server

Oracle HTTP Server can be configured to use either Oracle Diagnostic Logging (ODL) for generating diagnostic messages or the legacy Apache HTTP Server message format.

The following sections describe Oracle HTTP Server error and access log-related directives in the `httpd.conf` file.

- Oracle Diagnostic Logging Directives
- Apache HTTP Server Log Directives

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

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 or ODL text format.

`OraLogMode Apache | ODL-Text`

Default value: `ODL-Text`

For example: `OraLogMode ODL-Text`

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

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 `ODL-Text`. 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`

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 `ODL-Text`. When `OraLogMode` is set to `Apache`, `OraLogSeverity` is ignored and LogLevel is used instead. In the following syntax, *`short_module_identifierName`* is the module name with the trailing `_module` omitted.

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

Default value: `WARNING:32`

For example: `OraLogSeverity mime NOTIFICATION:32`

msg_type

Message types can be specified in upper or lowercase, 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.

OraLogRotationParams

Enables you to choose the rotation policy for an error log file. This directive is used only when OraLogMode is set to `ODL-Text`. 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 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.

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

- ErrorLog
- LogLevel
- LogFormat
- CustomLog

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 `ODL-Text`, 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

LogLevel

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

This directive is used only when `OraLogMode` is set to `Apache`. When `OraLogMode` is set to `ODL-Text`, 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

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 %E \"%r\" %>s %b" commonLogFormat "%h %l %u %t %E \"%r\"
%>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combinedLogFormat "%h %l %u %t %E
\"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedio
```

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

CustomLog

Use the `CustomLog` directive 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

Viewing Oracle HTTP Server Logs

You can view server logs using Fusion Middleware Control, WLST, or a text editor.

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. You can use this information to generate statistical reports about the server's usage patterns.

See Overview of Server Logs for more information on error logs and access logs.

This section describes the methods to view Oracle HTTP Server logs:

- Viewing Logs Using Fusion Middleware Control
- Viewing Logs Using WLST
- Viewing Logs in a Text Editor

- Viewing Logs Using Fusion Middleware Control
- Viewing Logs Using WLST
- Viewing Logs in a Text Editor

Viewing Logs Using Fusion Middleware Control

To access the log messages for an Oracle HTTP Server instance:

1. Navigate to the Oracle HTTP Server home page.
2. Select the server instance for which you want to view log messages.
3. From the Oracle HTTP Server drop-down list, select **Logs**, then **View Log Messages**.

   The Log Messages page opens.

For information about searching and viewing log files, see Viewing Log Files and Their Messages Using Fusion Middleware Control in *Administering Oracle Fusion Middleware*.

Viewing Logs Using WLST

To obtain and view server logs from the command line, you need to connect to Node Manager and issue the appropriate WebLogic Scripting Tool (WLST) command. These commands allow you to perform any of these functions:

- List server logs.
- Display the content of a specific log.

> **Note:**
>
> For more information on using WLST, see *Understanding the WebLogic Scripting Tool*.

Before attempting this procedure:

Before attempting to access server metrics from the command line, ensure the following:

- The domain exists.
- The instance you want to start exists.
- Node Manager is running on the instance machine.

To use this procedure, the instance and Administration server can be running but do not need to be.

To view metrics using WLST:

> **Note:**
>
> For managed domains, this procedure will work on an Administration server running on either the Administration machine or on a remote machine, whether the instance is in a running state or a shutdown state. For standalone domains, the procedure will work only on a local machine; however the instance can be either in a running *or* shutdown state.

1. Launch WLST:

   From Linux or UNIX:

   ```
   $ORACLE_HOME/oracle_common/common/bin/wlst.sh
   ```

   From Windows:

   ```
   C:\ORACLE_HOME\oracle_common\common\bin\wlst.cmd
   ```

2. From the selected domain directory (for example, *ORACLE_HOME*/user_projects/domains/*domainName*), connect to Node Manager:

   ```
   nmConnect('username', 'pwd', localhost, 5556, domainName)
   ```

3. Enter one of the following WLST commands, depending on what task you want to accomplish:

   - `listLogs(nmConnected=1, ...)`
   - `displayLogs(nmConnected=1, ...)`

   For example:

   ```
   listLogs(nmConnected=1, target='ohs1')
   displayLogs(nmConnected=1, target='ohs1', tail=5)
   ```

Viewing Logs in a Text Editor

You can also use a text editor 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. Download a log file to your local client and view the log files using another tool.

Recording ECID Information

You can configure Oracle HTTP Server logs to record Execution Context ID (ECID) information.

The following sections describe how to record Execution Context ID (ECID) information in error logs and access logs.

- About ECID Information
- Configuring Error Logs for ECID Information
- Configuring Access Logs for ECID Information

About ECID Information

An ECID is a globally unique ID that can be attached to requests between Oracle components. The ECID enables you to track log messages pertaining to the same request when multiple requests are processed in parallel.

The Oracle HTTP Server module `mod_context` scans each incoming request for an ECID-Context key in the URI or cookie, or for the ECID-Context header. If found, then the value is used as the execution context if it is valid. If it is not, then mod_context creates a new execution context for the request and adds it as the value of the ECID-Context header.

Configuring Error Logs for ECID Information

ECID information is recorded as part of Oracle Diagnostic Logging (ODL). ODL is a method for reporting diagnostic messages which presents a common format for diagnostic messages and log files, and a method for correlating all diagnostic messages from various components.

To configure Oracle HTTP Server error logs to record ECID information, ensure that the `OraLogMode` directive in the `httpd.conf` file is set to the default value, `odl`. The `odl` value specifies standard Apache log format and ECID information for log records specifically associated with a request.

For more information on `OraLogMode` and other possible values for this directive, see OraLogMode.

> **Note:**
>
> Oracle recommends that you enter the directives before any modules are loaded (`LoadModule` directive) in the `httpd.conf` file so that module-specific logging severities are in effect before modules have the opportunity to perform any logging.

Configuring Access Logs for ECID Information

By default, the `LogFormat` directive in the `httpd.conf` file is configured to capture ECID information:

```
LogFormat "%h %l %u %t %E \"%r\" %>s %b" common
```

If you want to add response time measured in microseconds, then add `%D` as follows:

```
LogFormat "%h %l %u %t %E %D \"%r\" %>s %b" common
```

If you want to suppress the capture of ECID information, then remove `%E` from the `LogFormat` directive:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

10

Managing Application Security

Oracle HTTP Server supports three main categories of security, namely, authentication, authorization, and confidentiality.

To know more about Oracle HTTP Server security features and configuration information for setting up a secure website, see the following sections:

- About Oracle HTTP Server Security
- Classes of Users and Their Privileges
- Authentication, Authorization and Access Control
- Implementing SSL
- Using mod_security
- Using Trust Flags

- About Oracle HTTP Server Security
  Oracle HTTP Server supports all three security categories, namely, authentication, authorization, and confidentiality. Oracle HTTP Server's security infrastructure is primarily provided by Apache security modules.
- Classes of Users and Their Privileges
  Oracle HTTP Server authorizes and authenticates users before allowing them to access or modify resources on the server, based on their user privileges.
- Authentication, Authorization and Access Control
  Oracle HTTP Server provides user authentication and authorization at two stages: access control and user authentication and authorization.
- Implementing SSL
  Oracle HTTP Server secures communications by using a Secure Sockets Layer (SSL) protocol. SSL secures communication by providing message encryption, integrity, and authentication. The SSL standard allows the involved components (such as browsers and HTTP servers) to negotiate which encryption, authentication, and integrity mechanisms to use.
- Using mod_security
  mod_security is an open-source module that you can use to detect and prevent intrusion attacks against Oracle HTTP Server.
- Using Trust Flags
  Trust flags allow adequate roles to be assigned to certificates to facilitate operations like certificate chain validation and path building. However, by default, wallets do not support trust flags.
- Enabling Perfect Forward Secrecy on Oracle HTTP Server
  Perfect Forward Secrecy (PFS) is a feature of specific key agreement protocols that gives assurance that your session keys will not be compromised even if the private key of the server is compromised.

About Oracle HTTP Server Security

Oracle HTTP Server supports all three security categories, namely, authentication, authorization, and confidentiality. Oracle HTTP Server's security infrastructure is primarily provided by Apache security modules.

Oracle HTTP Server 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 you can configure 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.

Classes of Users and Their Privileges

Oracle HTTP Server authorizes and authenticates users before allowing them to access or modify resources on the server, based on their user privileges.

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. These users have access to URLs defined in `http.conf` file. See 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 *Securing Applications with Oracle Platform Security Services*.

Authentication, Authorization and Access Control

Oracle HTTP Server provides user authentication and authorization at two stages: access control and user authentication and authorization.

- 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. You can configure the server 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.

- Access Control
- User Authentication and Authorization
- Support for FMW Audit Framework

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.

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. You can authenticate users with either Apache HTTP Server modules or with WebGate.

- Authenticating Users with Apache HTTP Server Modules
- Authenticating Users with WebGate
- Authenticating Users with Apache HTTP Server Modules
- Authenticating Users with WebGate

Authenticating Users with Apache HTTP Server Modules

The Apache HTTP Server authentication directives can be used to verify that users are who they claim to be.

For more information about how to authenticate users, see Apache HTTP Server documentation.

Authenticating Users 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.

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*

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.

You can configure audit filters using Fusion Middleware Control or by editing auditconfig.xml directly.

- Managing Audit Policies Using Fusion Middleware Control

> **See Also:**
>
> Overview of Audit Features in *Securing Applications with Oracle Platform Security Services*

Managing Audit Policies Using Fusion Middleware Control

Use the Audit Policies page in Fusion Middleware Control to assign audit policies to a selected Oracle HTTP Server instance.

1. Navigate to the Oracle HTTP Server Home Page.
2. Select the server instance to which you want to apply audit policies.
3. From the **Oracle HTTP Server** drop-down menu, select **Security**, and then select **Audit Policy**.

   The Audit Policy page appears.

For more information about setting audit policies, see Managing Audit Policies for Java Components with Fusion Middleware Control in *Securing Applications with Oracle Platform Security Services*.

Implementing SSL

Oracle HTTP Server secures communications by using a Secure Sockets Layer (SSL) protocol. SSL secures communication by providing message encryption, integrity, and authentication. The SSL standard allows the involved components (such as browsers and HTTP servers) to negotiate which encryption, authentication, and integrity mechanisms to use.

For details on how to implement SSL for Oracle HTTP Server, see Configuring SSL for the Web Tier in *Administering Oracle Fusion Middleware*. For information on using mod_ossl, Oracle's SSL module, see mod_ossl Module—Enables Cryptography (SSL). For information about mod_ossl directives, see mod_ossl Module.

The mod_wl_ohs module also contains a configuration for SSL. See Using SSL with Plug-ins and Parameters for Web Server Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

These sections describes SSL features that are supported for this release.

- Global Server ID Support
- PKCS #11 Support
- SSL and Logging
- Terminating SSL Requests

Global Server ID Support

The global ID support feature adds support SSL protocol features called variously as *step-up*, *server gated crypto*, or *global server ID*.

Step-up is a feature that allows old, weak encryption browsers, to step-up so that public keys greater than 512 bits and bulk encryption keys greater than 64 bits can be used in the SSL protocol. This means that server X.509 certificates that contain public keys in excess of 512 bits and which contain step-up digital rights can now be used by Oracle Application Server. Such certificates are often called 128 bit certificates, even though the certificate itself typically contains a 1024 bit certificate. The Verisign Secure Site Pro is an example of such a certificate which can now be used by Oracle Application Server.

Global Server ID functionality is provided by default.

PKCS #11 Support

Public-Key Cryptography Standards #11, or PKCS #11 for short, is a public key cryptography specification that outlines how systems use hardware security modules, which are basically "boxes" where cryptographic functions (encryption/decryption) are performed and where encryption keys are stored.

Oracle HTTP Server supports the option of having dedicated SSL hardware through `nCipher`. `nCipher` is a certified third-party accelerator that improves the performance of the PKI cryptography that SSL uses.

> **See Also:**
>
> - *Administering Oracle Fusion Middleware*
> - `http://www.ncipher.com`

SSL and Logging

SSL and communication related debugging can be set using the `SSLTraceLogLevel` directive. Here you can set different verbosity of log level according to your logging requirements. This directive generates SSL and communication logs. See SSLTraceLogLevel Directive.

> **Note:**
>
> SSL logs will work when Oracle HTTP Server logs is set for INFO or higher level.

Terminating SSL Requests

The following sections describe how to terminate requests using SSL before or within Oracle HTTP Server, where the `mod_wl_ohs` module forwards 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.

This section includes the following topics:

- About Terminating SSL at the Load Balancer
- About Terminating SSL at Oracle HTTP Server
- About Terminating SSL at the Load Balancer
- About Terminating SSL at Oracle HTTP Server

About Terminating SSL at the Load Balancer

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 10-1 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 10-1 Terminating SSL Before Oracle HTTP Server

- Terminating SSL at the Load Balancer

Terminating SSL at the Load Balancer

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 mod_certheaders Module—Enables Reverse Proxies.

> **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.example.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:
   <VirtualHost <name>:<port>
       # Use name and port used in url:
       ServerName <www.example.com:port>
       SimulateHttps On
       # The rest of your desired configuration for this VirtualHost goes
here
   </VirtualHost>
```

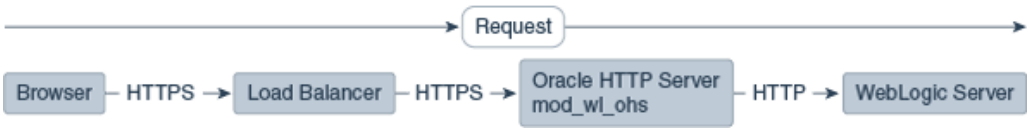
6. Restart Oracle HTTP Server and test access to the server, First test a static page such as `https://host:port/index.html` and then your test your application.

About Terminating SSL at Oracle HTTP Server

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

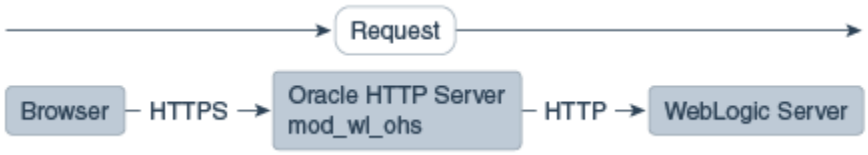
The following figures illustrate request flows, showing where HTTPS stops. In Figure 10-2, an HTTPS request is sent from the browser. The load balancer transmits the HTTPS request to Oracle HTTP Server. SSL is terminated in Oracle HTTP Server and the HTTP request is sent to WebLogic Server.

Figure 10-2 Terminating SSL at Oracle HTTP Server—With Load Balancer



In Figure 10-3 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 10-3 Terminating SSL at Oracle HTTP Server—Without Load Balancer



- Terminating SSL at Oracle HTTP Server

Terminating SSL at Oracle HTTP Server

To instruct the Oracle HTTP Server to treat requests as if they were received through HTTPS, configure the `WLSProxySSL` directive in the `mod_wl_ohs.conf` file and ensure that the `SecureProxy` directive is not configured.

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 Oracle WebLogic Server Proxy Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

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. Enable the WebLogic Plug-In flag for your managed servers or cluster.

   By default, this option is not enabled. Complete the following steps to enable the WebLogic Plug-In flag:

   a. Log in to the Oracle WebLogic Server Administration Console.

   b. In the **Domain Structure** pane, expand the **Environment** node.

   c. Click on **Clusters**.

   d. Select the cluster to which you want to proxy requests from Oracle HTTP Server.

      The Configuration: General tab appears.

   e. Scroll down to the **Advanced** section, expand it.

f. Click **Lock and Edit**.

g. Set the **WebLogic Plug-In Enabled** to *yes*.

h. Click **Save and Activate the Changes**.

i. Restart the servers for the changes to be effective.

5. Restart Oracle HTTP Server and test access to a Java application.

   For example: `https://host:port/path/application_name.`

Using mod_security

mod_security is an open-source module that you can use to detect and prevent intrusion attacks against Oracle HTTP Server.

An example of how you can use mod_security to prevent intrusion is by specifying 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.

See Configuring the mod_security Module.

Using Trust Flags

Trust flags allow adequate roles to be assigned to certificates to facilitate operations like certificate chain validation and path building. However, by default, wallets do not support trust flags.

You can use the orapki utility to maintain trust flags in the certificates installed in an Oracle Wallet. You can create and convert wallets to support trust flags, create and maintain appropriate flags in each certificate, and so on. For more information on trust flags and instructions on how to incorporate them into your security strategy, see Creating and Managing Trust Flags in *Administering Oracle Fusion Middleware*.

Enabling Perfect Forward Secrecy on Oracle HTTP Server

Perfect Forward Secrecy (PFS) is a feature of specific key agreement protocols that gives assurance that your session keys will not be compromised even if the private key of the server is compromised.

In Apache, the `SSLHonorCipherOrder directive` is used. This directive is supported in Oracle HTTP Server 12.2.1 and later.

Oracle HTTP Server out of the box configuration does not explicitly enable Perfect Forward Secrecy feature. To enable PFS, do the following configuration changes in the Oracle HTTP Server:

1. Configure TLS1.2 protocol for OHS server using SSLProtocol directive. See SSLProtocol Directive.
2. Enforce the ordering of server cipher suites by setting SSLHonorCipherOrder to `ON`. See SSLHonorCipherOrder Directive.

3. Use ECC certificates in Oracle HTTP Server wallet. See Adding an ECC Certificate to Oracle Wallets with orapki in *Administering Oracle Fusion Middleware*.
4. Configure ECDHE_ECDSA Cipher Suites in OHS. For the list of supported ECDHE_ECDSA cipher suites, see SSLCipherSuite Directive.

A

Oracle HTTP Server WLST Custom Commands

There are specific WLST Server commands for managing Oracle HTTP Server in WebLogic Server domains. Most are online commands, which require a connection between WLST and Administration Server for the domain.

This appendix contains information on Oracle HTTP Server specific WLST commands:

- Getting Help on Oracle HTTP Server WLST Custom Commands
- Oracle HTTP Server Commands
- Getting Help on Oracle HTTP Server WLST Custom Commands
  Online help is available for Oracle HTTP Server WLST custom commands.
- Using WLST Online Commands
- Oracle HTTP Server Commands
  Use the `ohs_createInstance` and `ohs_deleteInstance` commands to create and delete Oracle HTTP Server instances instead of using the Configuration Wizard. These custom commands perform additional error checking and assign ports automatically in the case of instance creation.

Getting Help on Oracle HTTP Server WLST Custom Commands

Online help is available for Oracle HTTP Server WLST custom commands.

To get online help, enter `help('manageohs')` from the WLST command line and it will display all the of the WLST custom commands for Oracle HTTP Server.

To get help for specific WLST custom commands, enter `help('custom_command_name')` from the WLST command line, for example:

```
help('ohs_createInstance')
```

Using WLST Online Commands

Perform the following steps before you use WLST online commands:

1. Open the CLI and launch WLST using the following command:
   - (Linux/UNIX) `$ORACLE_HOME/oracle_common/common/bin/wlst.sh`
   - (Windows) `$ORACLE_HOME\oracle_common\common\bin\wlst.cmd`
2. Connect to the Admin Server of the Domain.

   For more information about connecting to Admin Server, see connect section in WLST Command Reference for Oracle WebLogic Server.

3. Run the custom OHS WLST online command with the recommended arguments as shown in the following example:

```
ohs_createInstance(instanceName='xxx', machine='yyy',
serverName='zzz', ...)
```

Oracle HTTP Server Commands

Use the `ohs_createInstance` and `ohs_deleteInstance` commands to create and delete Oracle HTTP Server instances instead of using the Configuration Wizard. These custom commands perform additional error checking and assign ports automatically in the case of instance creation.

The WLST custom commands listed in Table A-1 manage Oracle HTTP Server instances in WebLogic Server domains.

Table A-1 Oracle HTTP Server Commands

Use this command...	To...	Use with WLST...
ohs_addAdminProperties	Add the LogLevel property to Oracle HTTP Server Administration server property file.	Online
ohs_addNMProperties	Add a property to the Oracle HTTP Server Node Manager plug-in property file.	Online
ohs_createInstance	Create a new instance of Oracle HTTP Server.	Online
ohs_deleteInstance	Delete the specified Oracle HTTP Server instance.	Online
ohs_exportKeyStore	Exports the keyStore to the specified Oracle HTTP Server instance.	Online
ohs_updateInstances	Creates a keystore in the KSS database in the case where Oracle HTTP Server instances were created using Configuration Wizard.	Online

- ohs_addAdminProperties
  The `ohs_addAdminProperties` command adds the `LogLevel` property to Oracle HTTP Server Administration server property file (`ohs_admin.properties`); `LogLevel` is the only parameter `ohs_addAdminProperties` currently supports. This command is available when WLST is connected to an Administration Server instance.
- ohs_addNMProperties
- ohs_createInstance
- ohs_deleteInstance
- ohs_exportKeyStore
- ohs_updateInstances

ohs_addAdminProperties

The `ohs_addAdminProperties` command adds the `LogLevel` property to Oracle HTTP Server Administration server property file (`ohs_admin.properties`); `LogLevel` is the

only parameter `ohs_addAdminProperties` currently supports. This command is available when WLST is connected to an Administration Server instance.

Use with WLST: Online

Syntax

```
ohs_addAdminProperties(logLevel = 'value')
```

Argument	Description
`LogLevel`	The granularity of information written to the log. The default is `INFO`. The following other values are accepted: • ALL • CONFIG • FINE • FINER • FINEST • OFF • SEVERE • WARNING

Example

This example creates a log file when log level is set to `FINEST`.

```
ohs_addAdminProperties(logLevel = 'FINEST')
```

ohs_addNMProperties

Use with WLST: Online

Description

The `ohs_addNMProperties` command adds a property to the Oracle HTTP Server Node Manager plug-in property file (`ohs_nm.properties`). This command is available when WLST is connected to an Administration Server instance.

Syntax

```
ohs_addNMProperties(logLevel = 'value', machine='node-manager-machine-name')
```

Argument	Description
`LogLevel`	The granularity of information written to the log. The default is `INFO`; other values accepted are: • ALL • CONFIG • FINE • FINER • FINEST • OFF • SEVERE • WARNING
`machine`	The name of the machine on which Node Manage is running.

Example

This example creates a log file with name `ohs_nm.log` under the path *<domain_dir>*`/system_components/OHS` with log level is set to `FINEST` on the target machine, `my_NM_machine`. The user need not restart Node Manager.

```
ohs_addNMProperties(logLevel = 'FINEST', machine = 'my_NM_machine')
```

ohs_createInstance

Use with WLST: Online

Description

The `ohs_createInstance` command creates a new instance of Oracle HTTP Server, allowing critical configuration such as listening ports to be specified explicitly or assigned automatically.

Syntax

```
ohs_createInstance(instanceName='xxx', machine='yyy', serverName='zzz', ...)
```

Argument	Definition
`instanceName`	The name of the managed instance being created.

`machine`	The existing machine entry for the instance. This name (often *<hostName>.myCorp.com*) is set during creation of the WebLogic Server Domain. If you forget the name, you can check $ORACLE_INSTANCE/config/config.xml and look for the <machine> block. Alternately, in WLST you can find the machine name by running:
	`serverConfig()`
	`cd('Machines')`
	`ls()`

`listenPort`	(Optional) The port number of the non-SSL server. If this value is not specified, a port is automatically assigned. Listen ports typically begin at 7777 and go up from there.
`sslPort`	(Optional) The port number of the SSL virtual host. If this value is not specified, a port is automatically assigned. SSL ports typically start at 4443 and go up from there.
`adminPort`	(Optional) The port number used for communication with Node Manager. If this value is not specified, a port is automatically assigned. Administration ports typically begin at 9999 and go up from there.
`serverName`	(Optional) The value of the ServerName directive of the non-SSL server. If this value is not specified, the host name of the machine and the listen port will be used to construct the value.

Example

The following example creates an Oracle HTTP Server instance called `ohs1` that runs on the machine `abc03.myCorp.com`:

```
ohs_createInstance(instanceName='ohs1', machine='abc03.myCorp.com')
```

ohs_deleteInstance

Use with WLST: Online

Description

The `ohs_deleteInstance` command deletes a specified Oracle HTTP Server instance. The instance must be stopped before you can delete it. This command will return an error if the instance is in the `UNKNOWN` or `RUNNING` state.

Syntax

```
ohs_deleteInstance(instanceName='xxx')
```

`instanceName` is the name of the Oracle HTTP Server instance.

Example

The following example deletes the Oracle HTTP Server instance `ohs1`.

```
ohs_deleteInstance(instanceName='ohs1')
```

ohs_exportKeyStore

Use with WLST: Online

Description

The `ohs_exportKeyStore` command exports the keystore to the specified Oracle HTTP Server instance location. This command is available when WLST is connected to an Administration Server instance. For more information on how to use this command, see Exporting the Keystore to an Oracle HTTP Server Instance Using WLST.

Syntax

```
ohs_exportKeyStore(keyStoreName='<keyStoreName>', instanceName = '<instanceName>')
```

Argument	Description
keyStoreName	The name of the keystore.
instanceName	The name of the Oracle HTTP Server instance.

Naming Conventions for Keystores

The keystore name (`keyStoreName`) must start with the string: `<instanceName>_`.

For example, presume that the keystore must be exported to an Oracle HTTP Server instance named `ohs1`. Then the names of all of the keystores that must be exported to `ohs1` must start with `ohs1_`.

If this syntax is not followed while creating the keystore, then the export of the keystore might not be successful.

Example

This example exports the keystore `ohs1_myKeystore` to the Oracle HTTP Server instance `ohs1`.

```
ohs_exportKeyStore(keyStoreName='ohs1_myKeystore', instanceName = 'ohs1')
```

ohs_updateInstances

Use with WLST: Online

Description

The `ohs_updateInstances` command is available only when WLST is connected to an Administration Server instance. It will parse across all of the Oracle HTTP Server instances in the domain and perform the following tasks:

- Create a new keystore with the name `<instanceName>_default` if one does not exist.
- Put a demonstration certificate, `demoCASignedCertificate`, in the newly created keystore.
- Export the keystore to the instance location.

This command is to be used after an Oracle HTTP Server instance is created using Configuration Wizard in collocated mode only. See Associating Oracle HTTP Server Instances With a Keystore Using WLST.

Syntax

```
ohs_updateInstances()
```

This command does not take any arguments.

Example

```
ohs_updateInstances()
```

B

Migrating to the mod_proxy_fcgi and mod_authnz_fcgi Modules

The `mod_fastcgi` module was deprecated in the previous release and has been replaced in the current release by the `mod_proxy_fcgi` and the `mod_authnz_fcgi` modules. You must complete certain tasks to migrate from the `mod_fastcgi` module to the `mod_proxy_fcgi` and `mod_authnz_fcgi` modules.

The `mod_proxy_fcgi` module uses `mod_proxy` to provide FastCGI support. The `mod_authnz_fcgi` module allows FastCGI authorizer applications to authenticate users and authorize access to resources.

Complete the following tasks to migrate from the `mod_fastcgi` module to the `mod_proxy_fcgi` and `mod_authnz_fcgi` modules:

- Task 1: Replace LoadModule Directives in htttpd.conf File
- Task 2: Delete mod_fastcgi Configuration Directives From the htttpd.conf File
- Task 3: Configure mod_proxy_fcgi to Act as a Reverse Proxy to an External FastCGI Server
- Task 4: Setup an External FastCGI Server
- Task 5: Setup mod_authnz_fcgi to Work with FastCGI Authorizer Applications
- Task 1: Replace LoadModule Directives in htttpd.conf File
  To update the `LoadModule` directives in the Oracle HTTP Server configuration file, `httpd.conf` open this file in an editor and replace the modules`mod_fastcgi` and `mod_fcgi` with the modules`mod_proxy` ,`mod_proxy_fcgi` , and `mod_authnz_fcgi` .
- Task 2: Delete mod_fastcgi Configuration Directives From the htttpd.conf File
  To migrate to the new modules provided by Oracle HTTP Server, you must delete the configuration directives that belong to the deprecated module `mod_fastcgi` in the `httpd.conf` file.
- Task 3: Configure mod_proxy_fcgi to Act as a Reverse Proxy to an External FastCGI Server
  The `mod_proxy_fcgi` module does not have configuration directives. Instead, it uses the directives set on the `mod_proxy` module. Unlike the `mod_fcgid` and `mod_fastcgi` modules, the `mod_proxy_fcgi` module has no provision for starting the application process. The purpose of `mod_proxy_fcgi` is to move this functionality outside of the web server for faster performance. So, `mod_proxy_fcgi` simply will act as a reverse proxy to an external FastCGI server.
- Task 4: Setup an External FastCGI Server
  An external FastCGI server enables you to run FastCGI scripts external to the web server or even on a remote machine. Therefore, you must set up an external FastCGI server.
- Task 5: Setup mod_authnz_fcgi to Work with FastCGI Authorizer Applications
  You can set up `mod_authnz_fcgi` module to work with FastCGI authorizer applications to authenticate users and authorize access to resources. It supports generic FastCGI authorizers that participate in a single phase for authentication and authorization, and

Apache httpd specific authenticators and authorizers. FastCGI authorizers can authenticate using the user ID and password for basic authentication or authenticate using arbitrary mechanisms.

Task 1: Replace LoadModule Directives in htttpd.conf File

To update the `LoadModule` directives in the Oracle HTTP Server configuration file, `httpd.conf` open this file in an editor and replace the modules`mod_fastcgi` and `mod_fcgi` with the modules`mod_proxy` ,`mod_proxy_fcgi` , and `mod_authnz_fcgi` .

Edit the `httpd.conf` file to comment out the `LoadModule` lines for `mod_fastcgi` and `mod_fcgi`. Add `LoadModule` lines for `mod_proxy`, `mod_proxy_fcgi`, and `mod_authnz_fcgi`. For example:

```
# LoadModule fastcgi_module modules/mod_fastcgi.so
# LoadModule fcgi_module modules/mod_fcgi.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_fcgi_module modules/mod_proxy_fcgi
LoadModule authnz_fcgi_module modules/mod_authnz_fcgi
```

Task 2: Delete mod_fastcgi Configuration Directives From the htttpd.conf File

To migrate to the new modules provided by Oracle HTTP Server, you must delete the configuration directives that belong to the deprecated module `mod_fastcgi` in the `httpd.conf` file.

For more information on these directives, see Module mod_fastcgi.

- `FastCgiServer`
- `FastCgiConfig`
- `FastCgiExternalServer`
- `FastCgiIpcDir`
- `FastCgiWrapper`
- `FastCgiAuthenticator`
- `FastCgiAuthenticatorAuthoritative`
- `FastCgiAuthorizer`
- `FastCgiAuthorizerAuthoritative`
- `FastCgiAccessChecker`
- `FastCgiAccessCheckerAuthoritative`

Task 3: Configure mod_proxy_fcgi to Act as a Reverse Proxy to an External FastCGI Server

The `mod_proxy_fcgi` module does not have configuration directives. Instead, it uses the directives set on the `mod_proxy` module. Unlike the `mod_fcgid` and `mod_fastcgi`

modules, the `mod_proxy_fcgi` module has no provision for starting the application process. The purpose of `mod_proxy_fcgi` is to move this functionality outside of the web server for faster performance. So, `mod_proxy_fcgi` simply will act as a reverse proxy to an external FastCGI server.

For examples of using `mod_proxy_fcgi`, see:

http://httpd.apache.org/docs/trunk/mod/mod_proxy_fcgi.html

For information about the directives available for `mod_proxy`, including reverse proxy examples, see:

http://httpd.apache.org/docs/trunk/mod/mod_proxy.html

Another way to setup the `mod_proxy_fcgi` module to act as a reverse proxy to a FastCGI server is to force a request to be handled as a reverse-proxy request. To do this, you must create a suitable Handler pass-through (also known as *Access via Handler*). For more information about how to set up a Handler pass-through, see:

http://httpd.apache.org/docs/trunk/mod/mod_proxy.html#handler

Task 4: Setup an External FastCGI Server

An external FastCGI server enables you to run FastCGI scripts external to the web server or even on a remote machine. Therefore, you must set up an external FastCGI server.

The following list provides information on some available FastCGI server solutions:

- fcgistarter, a utility for starting FastCGI programs. This solution is provided by Apache httpd 2.4. It only works on UNIX systems. See http://httpd.apache.org/docs/trunk/programs/fcgistarter.html.
- PHP-FPM, an alternative PHP FastCGI implementation. This solution is included with PHP release 5.3.3 and later. See http://php.net/manual/en/install.fpm.configuration.php.
- spawn-fcgi, a utility for spawning remote and local FastCGI processes. See http://redmine.lighttpd.net/projects/spawn-fcgi/wiki/WikiStart.

Task 5: Setup mod_authnz_fcgi to Work with FastCGI Authorizer Applications

You can set up `mod_authnz_fcgi` module to work with FastCGI authorizer applications to authenticate users and authorize access to resources. It supports generic FastCGI authorizers that participate in a single phase for authentication and authorization, and Apache httpd specific authenticators and authorizers. FastCGI authorizers can authenticate using the user ID and password for basic authentication or authenticate using arbitrary mechanisms.

For more information about using `mod_authnz_fcgi`, see http://httpd.apache.org/docs/trunk/mod/mod_authnz_fcgi.html.

C

Setting CGIDScriptTimeout When Using mod_cgid

Oracle HTTP Server includes `mod_cgi` and `mod_cgid` modules provided by Apache to run the CGI scripts.

When using a multi-threaded MPM on Unix, the `mod_cgid` module should be loaded instead of the `mod_cgi` module for better performance and to avoid unnecessary burden on the operating system due to forked multiple threads. The `mod_cgid` module has optimizations to improve the system performance in a multi-threaded environment as compared to the `mod_cgi` module. See Apache Module mod_cgid.

By default, the `mod_cgid` module is loaded when using a multi-threaded MPM on Unix. To verify the configuration:

1. Open the `httpd.conf` file using the Advanced Server Configuration page in the Fusion Middleware Control or a text editor.
2. In the `LoadModule` section, if `mod_cgid` is not configured already, add the following lines to load the `mod_cgid` module:

```
<IfDefine OHS_MPM_EVENT>
    LoadModule cgid_module "${PRODUCT_HOME}/modules/mod_cgid.so"
</IfDefine>

<IfDefine OHS_MPM_WORKER>
    LoadModule cgid_module "${PRODUCT_HOME}/modules/mod_cgid.so"
</IfDefine>
```

The `mod_cgid` module supports the `CGIDScriptTimeout` directive that can be used to limit the length of time to wait for more output from the CGI program.

- CGIDScriptTimeout Directive
  This directive limits the length of time to wait for more output from the CGI program.

CGIDScriptTimeout Directive

This directive limits the length of time to wait for more output from the CGI program.

If the time exceeds, the request and the CGI get terminated. It can be used to limit resource exhaustion due to the CGI scripts that stop communicating with the server and can protect against both unintentional errors and malicious actions (for example, DoS attacks).

By default, `mod_cgid` uses the Timeout Directive to limit the length of time to wait for CGI output. This timeout can be overridden with the `CGIDScriptTimeout` directive. The default value of `CGIDScriptTimeout` is the Timeout directive, when it is not set or set to `0`. To configure `CGIDScriptTimeout`:

1. Open the `httpd.conf` file using the Advanced Server Configuration page in the Fusion Middleware Control or a text editor.
2. Add the following lines for configuring the `CGIDScriptTimeout` directive:

```
<IfModule cgid_module>
#
# CGIDScriptTimeout: Limits the waiting time for output from the
CGI program
# Replace 20 with the actual timeout value to be set in seconds
#
 CGIDScriptTimeout 20
</IfModule>
```

> **Note:**
>
> Testing should be performed with your application to ensure the best results. The timeout value should be set based on the time required by your CGI program to send output back to OHS. The above configuration instructs OHS to wait for 20 seconds for output from the CGI program.

D

Frequently Asked Questions

This appendix provides answers to frequently asked questions about Oracle HTTP Server. 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.

- How Do I Create Application-Specific Error Pages?
- What Type of Virtual Hosts Are Supported for HTTP and HTTPS?
- Can I Use Different Language and Character Set Versions of Document?

- Can I Apply Apache HTTP Server Security Patches to Oracle HTTP Server?
- Can I Upgrade the Apache HTTP Server Version of Oracle HTTP Server?
- Can I Compress Output From Oracle HTTP Server?
- How Do I Create a Namespace That Works Through Firewalls and Clusters?
- How Can I Enhance Website Security?
- Why is REDIRECT_ERROR_NOTES not set for "File Not Found" errors?
- How can I hide information about the Web Server Vendor and Version
- Can I Start Oracle HTTP Server by Using apachectl or Other Command Line Tool?
- How Do I Configure Oracle HTTP Server to Listen at Port 80?
- How Do I Terminate Requests Using SSL Within Oracle HTTP Server?
- How Do I Configure End-to-End SSL Within Oracle HTTP Server?
- Can Oracle HTTP Server Front-End Oracle WebLogic Server?
- What is the Difference Between Oracle WebLogic Server Domains and Standalone Domains?
- Can Oracle HTTP Server Cache the Response Data?
- How Do I Configure a Virtual Server-Specific Access Log?
- How to Enable SSL for Oracle HTTP Server by Using Fusion Middleware Control?
  You can enable SSL for Oracle HTTP Server using Fusion Middleware control.

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`

What Type of Virtual Hosts Are Supported for HTTP and HTTPS?

(Apache 2.4 required)

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 processes 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.

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`

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). See 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.

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.

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

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 can 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.

How Can I Enhance Website Security?

The following are some general guidelines for securing your web site.

- Use a commercial firewall between your ISP and your Web server.
- 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.
- Always validate all input from Web forms and output from your applications. Be sure to validate encodings, long input strings and input that contains non-printable characters, HTML tags, or javascript tags.
- Encrypt the contents of cookies when it is relevant.
- Check often for security patches for all your system and application software, and install them as soon as possible. Only accept patches from Oracle or your Oracle support representative.
- When it is relevant, use an intrusion detection package to monitor for defaced Web pages, viruses, and presence of rootkits. If possible, mount system executables and Web content on read-only file systems.
- Consider using Pen testing or other relevant security testing on your application. Consider configuring web security using the appropriate custom mod_security rules to protect your application. For more information on mod_security, see Configuring the mod_security Module and Using mod_security.
- Remove unneeded content from the httpd.conf file.See Removing Access to Unneeded Content.
- Take precautions to protect your web pages from clickjacking attempts. There is a lot of helpful information available on the internet. For more information on clickjacking, see the Security Best Practices section in "Security Vulnerability FAQ for Oracle Database and Fusion Middleware Products (Doc ID 1074055.1)".

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.

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.

Can I Start Oracle HTTP Server by Using apachectl or Other Command Line Tool?

Oracle HTTP Server process management is handled by Node Manager. You can use the `startComponent` command to start Oracle HTTP Server without using WLST or Fusion Middleware Control directly. See Starting Oracle HTTP Server Instances from the Command Line.

How Do I Configure Oracle HTTP Server to Listen at Port 80?

By default, Oracle HTTP Server is not able to bind to ports on UNIX in the reserved range (typically less than 1024). You can enable Oracle HTTP Server to listen on a port in the reserved range (for example, the default port 80) by following the instructions in Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only).

How Do I Terminate Requests Using SSL Within Oracle HTTP Server?

You can terminate requests using SSL before or within Oracle HTTP Server, where the `mod_wl_ohs` module forwards 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. See Terminating SSL at the Load Balancer and Terminating SSL at Oracle HTTP Server.

How Do I Configure End-to-End SSL Within Oracle HTTP Server?

Support for Secure Sockets Layer (SSL) is provided by the Oracle WebLogic Server Proxy Plug-In. You can use the SSL protocol to protect the connection between the plug-in and Oracle WebLogic Server. The SSL protocol provides confidentiality and integrity to the data passed between the plug-in and WebLogic Server. See Use SSL with Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins* for information on setting up SSL libraries and for setting up one-way or two-way SSL communications between the web server and Oracle WebLogic Server.

If you will be configuring SSL in Oracle HTTP Server but not on Oracle WebLogic Server, then you can terminate SSL for requests sent by Oracle HTTP Server. For information on configuring this scenario, see Terminating SSL at Oracle HTTP Server.

Can Oracle HTTP Server Front-End Oracle WebLogic Server?

Oracle HTTP Server is the web server component for Oracle Fusion Middleware. The server uses 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 using its built-in Oracle WebLogic Server Proxy Plug-In (`mod_wl_ohs` module) to route dynamic content requests to WebLogic-managed servers.

For information about the topologies you into which you can install Oracle HTTP Server, see Oracle HTTP Server Topologies.

What is the Difference Between Oracle WebLogic Server Domains and Standalone Domains?

Oracle HTTP Server can be installed in either a standalone, a Full-JRF, or a Restricted-JRF domain. A standalone domain is a container for system components, such as Oracle HTTP Server. It is ideal for a DMZ environment because it has the least overhead. A standalone domain has a directory structure similar to an Oracle WebLogic Server Domain, but it does not contain an Administration Server, or Managed Servers, or any management support. 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.

WebLogic Server Domains support all WebLogic Management Framework tools. An Oracle WebLogic Server domain can be either Full-JRF or Restricted JRF. A WebLogic Server Domain in Full-JRF mode 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.

The purpose of the Restricted-JRF domain is to simplify Oracle HTTP Server administration by using the WebLogic server domain. A Restricted-JRF Oracle WebLogic Server domain is similar to a Full-JRF domain except that a connection to an external database is not required. All of the Oracle HTTP Server functionality through Fusion MiddleWare Control and WLST is still available, with the exception of cross component wiring.

For more details on each of these domains, see Domain Types.

Can Oracle HTTP Server Cache the Response Data?

Oracle HTTP Server now includes the Apache mod_cache and mod_cache_disk modules to cache response data.

For more information, on mod_cache and mod_cache_disk, see mod_cache in the Apache documentation:

http://httpd.apache.org/docs/2.4/mod/mod_cache.html

How Do I Configure a Virtual Server-Specific Access Log?

Within every VirtualHost directive, you can use the Apache LogFormat and CustomLog directives to configure Virtual Host-specific access log format and log files. See LogFormat and CustomLog.

How to Enable SSL for Oracle HTTP Server by Using Fusion Middleware Control?

You can enable SSL for Oracle HTTP Server using Fusion Middleware control.

The steps mentioned in this section is applicable to Oracle HTTP Server - Version 12.2.1.0.0 and later.

Complete the following steps to enable SSL for Oracle HTTP Server using Fusion Middleware control:

- Start Node Manager and Admin Server
- Create Keystore
- Generate Keypair
- Generate CSR for a Certificate
- Import the Trusted Certificate
- Import the Trusted Certificate to WebLogic Domain
- Import the User Certificate
- Export Keystore to Wallet
- Start Node Manager and Admin Server
- Create Keystore
- Generate Keypair
- Generate CSR for a Certificate
- Import the Trusted Certificate
- Import the Trusted Certificate to WebLogic Domain
- Import the User Certificate
- Export Keystore to Wallet
- Enable SSL

Start Node Manager and Admin Server

1. Start the Node Manager in the collocated `ORACLE_HOME`.

```
$ORACLE_HOME/user_projects/domains/bin/startNodeManager.sh
```

2. Start the Admin Server in the collocated `ORACLE_HOME`.

   ```
   $ORACLE_HOME/user_projects/domains/bin/startWeblogic.sh
   ```

3. Log in to Fusion Middleware Control with the Weblogic user name and password.

   For example, *http://host.domain:7001/em*.

Create Keystore

1. Log in to **Fusion Middleware Control**.
2. Go to **Domain**, click **Security**, and then click **Keystore**.

   The **Keystore** page appears.
3. Click **Create Keystore**.

   The **Create Keystore** dialog box appears.
4. In this dialog box, enter the following data:
   - Keystore Name: Enter a unique name. For example, Test.
   - Protection Type: Choose Policy.

   A new keystore is created with the name _Test, that is, *ohs1_Test*.
   Once the keystore is created, select the new keystore *ohs1_Test*, and then click Manage to perform all other steps

Generate Keypair

To generate a certificate with an associated keypair:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**.

   The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.

   The **Manage Certificates** page appears.
6. Click **Generate Keypair**.

   The Generate Keypair dialog appears.
7. Enter the details, and the click OK.

   The new certificate appears in the list of certificates. You can view the certificate details by clicking on the certificate alias.

The generated keypair is wrapped in a CA signed certificate. To use this certificate for SSL or where trust needs to be established, applications must either use the domain trust store as their trust store or import the certificate to a custom application-specific trust store.

Generate CSR for a Certificate

To generate a CSR for a certificate or trusted certificate:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, and then **Keystore**.

   The **Keystore** page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.

   The **Manage Certificates** page appears.
6. Select the row corresponding to the new keypair and click **Generate CSR**.

   The **Generate CSR** dialog appears.
7. Copy and paste the entire CSR into a text file, and click **Close**.

   Alternatively, you can click **Export CSR** to automatically save the CSR to a file.

You can send the resulting certificate request to a certificate authority (CA) which will return a signed certificate.

Import the Trusted Certificate

To import a certificate into a password-protected keystore.

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate Oracle HTTP Server.
3. Navigate to **Security**, and then **Keystore**.

   The **Keystore** page appears.
4. Expand the stripe in which the keystore resides. Select the keystore from which the CSR was generated.
5. Click **Manage**.

   The **Manage Certificates** page appears.
6. Click **Import**.

   The **Import Certificate** dialog appears.
7. In the **Certificate Type**, select *Trusted Certificate*.
8. In **Alias**, enter a name for the **Alias**.
9. In **Certificate Source**, either paste the content of the trusted certificate in **Paste Certificate String here** text box or select a trusted certificate file.
10. Click **OK**.

    Repeat these steps for any other trusted CA certificates in the chain.

    The imported trusted certificate appears in the list of certificates.

Import the Trusted Certificate to WebLogic Domain

You also need to import root CA certificate and any other Trusted CA Certificates to WebLogic "system" stripe under trust keystore.

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate WebLogic domain.
3. Navigate to **Security**, and then **Keystore**.

   The **Keystore** page appears.
4. Expand the stripe in which the keystore resides. Select the keystore from which the CSR was generated.
5. Click **Manage**.

   The **Manage Certificates** page appears.
6. Click **Import**.

   The **Import Certificate** dialog appears.
7. In the **Certificate Type**, select *Trusted Certificate*.
8. In **Alias**, enter a name for the **Alias**.
9. In **Certificate Source**, either paste the content of the trusted certificate in **Paste Certificate String here** text box or select a trusted certificate file.
10. Click **OK**.

    Repeat these steps for any other trusted CA certificates in the chain.

    The imported trusted certificate appears in the list of certificates.

If you miss this step, then trying to export keystore to wallet fails with the following error message:

```
Error "Failed to export keystore to wallet. Error message: null"
While Trying to Export Keystore to Wallet
```

See Note: 2140257.1

Import the User Certificate

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate Oracle HTTP Server.
3. Navigate to **Security**, and then **Keystore**.

   The **Keystore** page appears.
4. Expand the stripe in which the keystore resides. Select the keystore from which the CSR was generated.
5. Click **Manage**.

   The **Manage Certificates** page appears.
6. Click **Import**.

   The **Import Certificate** dialog appears.

7. In the **Certificate Type**, select *Certificate*.
8. In **Alias**, enter a name for the **Alias**.
9. In **Certificate Source**, either paste the content of the user certificate in **Paste Certificate String here** text box or select a user certificate file.
10. Click **OK**.

    The imported user certificate appears in the list of certificates.

Export Keystore to Wallet

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate Oracle HTTP Server.
3. Navigate to **Security**, and then **Keystore**.

    The **Keystore** page appears.
4. Expand the stripe in which the keystore resides. Select the keystore from which the CSR was generated.
5. Click **Manage**.

    The **Manage Certificates** page appears.
6. Click **Import**.

    The **Import Certificate** dialog appears.
7. Click **Export Keystore to Wallet**.

    You get an auto login wallet, cwallet.sso, that does not need a password. This auto login enabled wallet is also associated with a PKCS#12 wallet (ewallet.p12).

Enable SSL

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 **SSL Configuration**.
7. Check the **Enable SSL** box.
8. Select a wallet from the drop-down list.

    Here, select the path to *Test* wallet.
9. Click **OK** to apply the changes.
10. Restart the Oracle HTTP Server instance by navigating to **Oracle HTTP Server**, then **Control**, then **Restart.**
11. Open a browser session and connect to the port number that was SSL-enabled.

E

Troubleshooting Oracle HTTP Server

You can get help to troubleshoot some of the common problems that you might encounter when using Oracle HTTP Server.

- Oracle HTTP Server Fails to Start Due to Port Conflict
  If Oracle HTTP Server cannot start due to a port conflict, a message containing the string `[VirtualHost: main] (98)Address already in use` is generated. This error condition occurs if the listen port configured for Oracle HTTP Server is the same as the one in use by another process.
- System Overloaded by Number of httpd Processes
  When the system is overloaded by too many `httpd` processes, there are insufficient resources for normal processing. This slows down the response time. You can lower the value of `MaxRequestWorkers` to a value the machine can accommodate.
- Permission Denied When Starting Oracle HTTP Server On a Port Below 1024
  If you try to start Oracle HTTP Server on a port below 1024, a message containing the string `[VirtualHost: main] (13)Permission denied: make_sock: could not bind to address [::]:443` is generated. This error condition occurs because root privileges are needed to bind these ports.
- Using Log Files to Locate Errors
  There are three types of log files that help you locate errors, namely, rewrite, script, and error.
- Recovering an Oracle HTTP Server Instance on a Remote Host
  To recover an Oracle HTTP Server instance on a remote host, you must use `tar` and `untar`; `pack.sh` and `unpack.sh` do not work in this scenario.
- Oracle HTTP Server Performance Issues
  You might encounter performance issues when running Oracle HTTP Server. The documentation includes several topics to explain such performance related problems.

- Out of DMS Shared Memory
  When there is an incorrect calculation of the required shared memory for Oracle HTTP Server DMS, error logs are displayed. These problems can be resolved by setting the DMS shared memory directive to a value larger than the default value of 4096 or continuing to set the directive 50% higher until the problem is resolved.
- Oracle HTTP Server Fails to Start When mod_security is Enabled on RHEL or Oracle Linux 7
  If `mod_security` is configured in Oracle HTTP Server in Red Hat Enterprise Linux (RHEL) or Oracle Linux (OL) 7, Oracle HTTP Server fails to start. This error condition occurs because there is no symbolic link /lib64/liblzma.so.0
- Oracle HTTP Server Fails to Start due to Certificates Signed Using the MD5 Algorithm
  If Oracle HTTP Server cannot start due to the server wallet containing a certificate signed with the Message Digest 5 (MD5) algorithm, you can replace the MD5 certificate with a Secure Hash Algorithm 2 (SHA-2) certificate.
- Node Manager Logs Don't Show Clear Message When a Component Fails to Start
  When an Oracle HTTP Server (OHS) component fails to start, the following errors are seen in *ORACLE_INSTANCE*`/servers/`*COMPONENT_NAME*`/logs/`*COMPONENT_NAME*`.log`:
- SSL Handshake Fails Due to Certificate Chain

Oracle HTTP Server Fails to Start Due to Port Conflict

If Oracle HTTP Server cannot start due to a port conflict, a message containing the string `[VirtualHost: main] (98)Address already in use` is generated. This error condition occurs if the listen port configured for Oracle HTTP Server is the same as the one in use by another process.

The generated message may look like the following:

```
[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.

> **Note:**
>
> If the Oracle HTTP Server instance was created with the config Wizard, there is no automated port management. It is possible to create multiple instances using the same Listen port.

System Overloaded by Number of httpd Processes

When the system is overloaded by too many `httpd` processes, there are insufficient resources for normal processing. This slows down the response time. You can lower the value of `MaxRequestWorkers` to a value the machine can accommodate.

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 `MaxRequestWorkers` to a value the machine can accommodate.

Permission Denied When Starting Oracle HTTP Server On a Port Below 1024

If you try to start Oracle HTTP Server on a port below 1024, a message containing the string `[VirtualHost: main] (13)Permission denied: make_sock: could not bind to address [::]:443` is generated. This error condition occurs because root privileges are needed to bind these ports.

The generated message may look like the following:

```
[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 Starting Oracle HTTP Server Instances on a Privileged Port (UNIX Only) to start Oracle HTTP Server on a Privileged Port.

Using Log Files to Locate Errors

There are three types of log files that help you locate errors, namely, rewrite, script, and error.

The log files are explained in the following sections:

- Rewrite Log
- Script Log
- Error Log
- Rewrite Log
- Script Log
- Error Log

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 value of the `LogLevel` directive controls the level of detail.

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`

Error Log

This log file records overall server problems. Refer to Managing Oracle HTTP Server Logs for details on configuring and viewing error logs.

Recovering an Oracle HTTP Server Instance on a Remote Host

To recover an Oracle HTTP Server instance on a remote host, you must use `tar` and `untar`; `pack.sh` and `unpack.sh` do not work in this scenario.

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.

Oracle HTTP Server Performance Issues

You might encounter performance issues when running Oracle HTTP Server. The documentation includes several topics to explain such performance related problems.

- Special Runtime Files Reside on a Network File System
- UNIX Sockets on a Network File System
- DocumentRoot on a Slow File System
- Instances Created on Shared File Systems

- Special Runtime Files Reside on a Network File System
- UNIX Sockets on a Network File System
- DocumentRoot on a Slow File System

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:

In httpd.conf, change `Mutex fnctl:fileloc default` to `Mutex sysvsem default` where `fileloc` is the value of the directive `LockFile` (two places).

On Solaris:

In httpd.conf, change `Mutex fnctl:fileloc default` to `Mutex pthread default` where `fileloc` is the value of the directive `LockFile` (two places).

UNIX Sockets on a Network File System

The mod_cgid module is not enabled by default. If enabled, this module uses UNIX sockets internally. If UNIX sockets reside on a slower file system (for example, network file system), a severe performance degradation could be observed. You can set the following directive to avoid the issue:

- If mod_cgid is enabled, use the `ScriptSock` directive to place mod_cgid's UNIX socket on a local filesystem.

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 (for example, network file system), then every request that mod_wl_ohs routes to the backend server can experience performance issues. This can be overcome by setting `WLSRequest` to `ON` instead of `SetHandler weblogic-handler`.

Instances Created on Shared File Systems

If you encounter functional or performance issues when creating an Oracle HTTP Server instance on a shared file system, including NFS (Network File System), it might be due to file system accesses in the default configuration. In this case, you must update the httpd.conf file specific to your operating systems. See Updating Oracle HTTP Server Component Configurations on a Shared File System.

Out of DMS Shared Memory

When there is an incorrect calculation of the required shared memory for Oracle HTTP Server DMS, error logs are displayed. These problems can be resolved by setting the DMS shared memory directive to a value larger than the default value of 4096 or continuing to set the directive 50% higher until the problem is resolved.

An error log containing the string `dms_fail_shm_expansion: out of DMS shared memory in pid XXX, disabling DMS; increase DMSProcSharedMem directive from YYY` is displayed when an incorrect calculation of required shared memory for Oracle HTTP Server DMS. This

can be resolved by setting `DMSProcSharedMem` to a larger value than the default value of 4096. In some extreme configurations, you might see the following message in the Oracle HTTP Server error log:

```
dms_fail_shm_expansion: out of DMS shared memory in pid XXX, disabling DMS;
increase DMSProcSharedMem directive from YYY
```

This is because of an incorrect calculation of required shared memory for Oracle HTTP Server DMS. This can be resolved by setting `DMSProcSharedMem` to a larger value than the default of 4096. Continue setting `DMSProcSharedMem` 50% higher until the problem is resolved. The minimum value for `DMSProcSharedMem` is 256 and the maximum value is 65536.

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 a large enough value that Oracle HTTP Server starts without error. The value of this variable is in kilobytes and a value of 524288 is a good starting point. If the error persists, continue to increase the value by 50% until Oracle HTTP Server starts without error.

Oracle HTTP Server Fails to Start When mod_security is Enabled on RHEL or Oracle Linux 7

If `mod_security` is configured in Oracle HTTP Server in Red Hat Enterprise Linux (RHEL) or Oracle Linux (OL) 7, Oracle HTTP Server fails to start. This error condition occurs because there is no symbolic link /lib64/liblzma.so.0

The generated error looks like the following:

```
iblzma.so.0: cannot open shared object file: No such file or directory
```

Solution

1. Log in as a root user.
2. To create a symbolic link, /lib64/liblzma.so.0, run the following command:

   ```
   cd /lib64
   ln -s liblzma.so.5.0.99 liblzma.so.0
   ```

3. Verify the symlink as follows:

   ```
   ls -al *liblzma*
   ```

4. Exit root.
5. Start Oracle HTTP Server.

   For example, startComponent.sh ohs1, where *ohs1* is the Oracle HTTP Server instance you want to start.

Oracle HTTP Server Fails to Start due to Certificates Signed Using the MD5 Algorithm

If Oracle HTTP Server cannot start due to the server wallet containing a certificate signed with the Message Digest 5 (MD5) algorithm, you can replace the MD5 certificate with a Secure Hash Algorithm 2 (SHA-2) certificate.

Oracle HTTP Server fails to start if the Oracle HTTP Server wallet contains a certificate or certificate request that is signed with the Message Digest 5 (MD5) algorithm.

- **Solution:** Replace the MD5 certificate with a Secure Hash Algorithm 2 (SHA-2) certificate.
- **Workaround:** To enable MD5 supported certificate, set the `ORACLE_SSL_ALLOW_MD5_CERT_SIGNATURES` environment variable in the *ohs.plugins.nodemanager.properties* file to `1`.

To set the environment variable in Oracle HTTP Server, see Environment Variable Configuration Properties.

Node Manager Logs Don't Show Clear Message When a Component Fails to Start

When an Oracle HTTP Server (OHS) component fails to start, the following errors are seen in *ORACLE_INSTANCE*/servers/*COMPONENT_NAME*/logs/*COMPONENT_NAME*.log:

```
[OHS] [INCIDENT_ERROR:20] [AH00480] [mpm_event] [host_id: xxx] [host_addr:
xxx] [pid: xxx]
[tid:xxxx] [user: xxx] [VirtualHost: main] (11)Resource
temporarily unavailable: AH00480: apr_thread_create: unable to create worker
thread
```

This can be caused due to lack of Virtual Memory or a limit has been placed on the OHS for the number of processes it can run.

Solution

1. Check and increase the Virtual memory on the host or check the local process limits and open file descriptor limit for the user.
   For example, on Linux, check the user limits using the command `ulimit -a` and also check the `/etc/security/limits.conf` file for any system-wide user limits.

   To increase process limit, use the following command on Linux:

   ```
   $ ulimit -u xxxx
   ```

   To increase open files : file descriptors limit, use the following command on Linux:

   ```
   $ ulimit -n xxxxx
   ```

2. Kill the processes that are not required and start the OHS.

SSL Handshake Fails Due to Certificate Chain

Certain browsers, such as Internet Explorer require that the entire certificate chain be imported to the browsers for the SSL handshake to work. If your certificate was issued by an intermediate CA, you will need to ensure that the complete chain of certificates is available on the browser or the handshake will fail. If an intermediate certificate in the chain expires, it must be renewed along with all the certificates (such as OHS server) in the chain.

Solution

When you configure SSL for Oracle HTTP Server, you may need to import the entire certificate chain (rootCA, Intermediate CA's and so on).

F

Configuration Files

Oracle HTTP Server contains configuration files that specify several properties, such as the top-level web server configuration, listen ports, the administration port, the SSL configuration, the plug-ins, keystores, log files, and more.

File	Format	Description
httpd.conf	Apache HTTP Server .conf file format	Top-level web server configuration file Primary feature configured: Various, including non-SSL listening socket
ssl.conf	Apache HTTP Server .conf file format	Web server configuration file for SSL Primary feature configured: mod_ossl
admin.conf	Apache HTTP Server .conf file format	Web server configuration file for administration port. Only the listen port and local address are intended for customer configuration. Primary feature configured: mod_dms; administration port used for communication with Node Manager
mod_wl_ohs.conf	Apache HTTP Server .conf file format	Web server configuration file for WebLogic plugin Primary feature configured: WebLogic plugin (mod_wl_ohs)
mime.types	mod_mime file format	Web server configuration file for mod_mime Primary feature configured: Mime types used by mod_mime
ohs.plugins.nodemanager.properties	Java property file format	Configuration file for Oracle HTTP Server Node Manager plug-ins Primary feature configured: Oracle HTTP Server plug-ins
magic	mod_mime_magic file format	Optional, disabled web server configuration file for mod_mime_magic Primary feature configured: File content patterns used by mod_mime_magic
keystores/<*wallet-directory*>	Oracle wallet format	Oracle wallet Primary feature configured: Oracle wallets for SSL/TLS communication
auditconfig.xml	FMW audit framework audit configuration XML format	Configuration of Oracle HTTP Server auditing and logging Primary feature configured:FMW audit framework auditing of Oracle HTTP Server operations

File	Format	Description
component-logs.xml	FMW log file configuration XML format	Configuration of Oracle HTTP Server log files for log collection Primary feature configured: Log collection
component_events.xml	FMW audit framework component event XML format	Static configuration of Oracle HTTP Server audit event definitions Primary feature configured: FMW audit framework

For additional information, see the following documentation:

- Understanding Configuration Files
- Apache HTTP Server .conf file format: http://httpd.apache.org/docs/2.4/configuring.html
- mod_mime file format: http://httpd.apache.org/docs/2.4/mod/mod_mime.html
- mod_mime_magic file format: http://httpd.apache.org/docs/2.2/mod/mod_mime_magic.html

G

Property Files

Oracle HTTP Server instances can be configured using property files such as`ohs_admin.properties`, `ohs_nm.properties`, and `ohs.plugins.nodemanager.properties`.

This appendix documents the property files used by Oracle HTTP Server. The files include:

- ohs_addAdminProperties
- ohs_nm.properties File
- ohs.plugins.nodemanager.properties File

- ohs_addAdminProperties
  The `ohs_addAdminProperties` command adds the `LogLevel` property to Oracle HTTP Server Administration server property file (`ohs_admin.properties`); `LogLevel` is the only parameter `ohs_addAdminProperties` currently supports. This command is available when WLST is connected to an Administration Server instance.
- ohs_nm.properties File
  The `ohs_nm.properties` file is a per domain file used to configure the Oracle HTTP Server plug-in.
- ohs.plugins.nodemanager.properties File
  An `ohs.plugins.nodemanager.properties` file exists for each configured Oracle HTTP Server instance. This file contains parameters for configuring Oracle HTTP Server process management.

ohs_addAdminProperties

The `ohs_addAdminProperties` command adds the `LogLevel` property to Oracle HTTP Server Administration server property file (`ohs_admin.properties`); `LogLevel` is the only parameter `ohs_addAdminProperties` currently supports. This command is available when WLST is connected to an Administration Server instance.

Use with WLST: Online

Syntax

```
ohs_addAdminProperties(logLevel = 'value')
```

Argument	Description
`LogLevel`	The granularity of information written to the log. The default is `INFO`. The following other values are accepted: • ALL • CONFIG • FINE • FINER • FINEST • OFF • SEVERE • WARNING

Example

This example creates a log file when log level is set to `FINEST`.

```
ohs_addAdminProperties(logLevel = 'FINEST')
```

ohs_nm.properties File

The `ohs_nm.properties` file is a per domain file used to configure the Oracle HTTP Server plug-in.

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

ohs.plugins.nodemanager.properties File

An `ohs.plugins.nodemanager.properties` file exists for each configured Oracle HTTP Server instance. This file contains parameters for configuring Oracle HTTP Server process management.

File path: *`DOMAIN_HOME`*`/config/fmwconfig/components/OHS/`*`instance_name`*`/ohs.plugins.nodemanager.properties`

This section contains the following information:

- Cross-platform Properties
- Environment Variable Configuration Properties

- Properties Specific to Oracle HTTP Server Instances Running on Linux and UNIX

> **Note:**
>
> Any paths placed in Windows implementations of ohs.plugins.nodemanager.properties that include backslashes must have those backslashes escaped.
>
> You must do this manually after upgrading from Oracle HTTP Server 11g where paths with backslashes were migrated from opmn.xml to ohs.plugins.nodemanager.properties.
>
> For example:
>
> ```
> environment.TMP = C:\Users\user\AppData\Local\Temp\1
> ```
>
> Must be modified manually to:
>
> ```
> environment.TMP = C:\\Users\\user\\AppData\\Local\\Temp\\1
> ```

- Cross-platform Properties
  You can configure cross-platform properties for Oracle HTTP Server instances such as `config-file`, `command-line`, and more.
- Environment Variable Configuration Properties
  You can specify additional environment variables for the Oracle HTTP Server using environment properties such as `SHELL`, `LANG`, `INSTANCE_NAME`, and more.
- Properties Specific to Oracle HTTP Server Instances Running on Linux and UNIX
  You can configure properties for Oracle HTTP Server instances running on Linux or other UNIX like systems. These properties include `restart-mode`, `stop-mode`, and more.

Cross-platform Properties

You can configure cross-platform properties for Oracle HTTP Server instances such as `config-file`, `command-line`, and more.

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

Property	Description
`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`
`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 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`
`nm-wallet`	Full path to Node Manager wallet. This wallet contains trusted certificates which are used by Node Manager to establish SSL communication over OHS admin port. If the absolute path of the wallet is not configured, the default lookup directory is set to *`INSTANCE_HOME`*`/config/fmwconfig/components/`*`COMPONENT_TYPE`*`/instances/`*`COMPONENT_NAME`*`/keystores/`. Default: *`INSTANCE_HOME`*`/config/fmwconfig/components/`*`COMPONENT_TYPE`*`/instances/`*`COMPONENT_NAME`*`/keystores/default`

Example:

```
config-file = httpd.conf
command-line = -DSYMBOL
start-timeout = 120
stop-timeout = 60
restart-timeout = 180
ping-interval = 30
ping-timeout = 60
nm-wallet = <path to wallet directory>
```

Environment Variable Configuration Properties

You can specify additional environment variables for the Oracle HTTP Server using environment properties such as `SHELL`, `LANG`, `INSTANCE_NAME`, and more.

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.

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 is necessary 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 plug-in:

- SHELL: From 's environment, or defaults to /bin/sh, or cmd.exe for Windows
- ORA_NLS33: Set to $ORACLE_HOME/nls/data
- NLS_LANG: From 's environment, otherwise default
- LANG: From 's environment, otherwise default
- LC_ALL: From 's environment, if set
- TZ: From '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 Oracle HTTP Server 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 '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
```

Properties Specific to Oracle HTTP Server Instances Running on Linux and UNIX

You can configure properties for Oracle HTTP Server instances running on Linux or other UNIX like systems. These properties include `restart-mode`, `stop-mode`, and more.

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

Property	Description
mpm	Determines whether to use the prefork, worker, or event MPM for Oracle HTTP Server. mpm accepts these values: • prefork • worker • event Default: worker for UNIX, event for Linux
allow-corefiles	Determines whether ulimit should be set to allow core files to be written for Oracle HTTP Server crashes. allow-corefiles accepts these values: • yes • no Default: no

Example

```
restart-mode = graceful
stop-mode = stop
mpm = worker
allow-corefiles = no
```

H

Oracle HTTP Server Module Directives

Modules extend the basic functionality of Oracle HTTP Server and support integration between Oracle HTTP Server and other Oracle Fusion Middleware components. Oracle HTTP Server uses both Oracle developed modules or “plug-ins” and Apache and third party-developed modules. Oracle developed modules have a set of directives that Oracle HTTP Server supports.

This appendix describes the directives available in the Oracle-developed modules:

- mod_wl_ohs Module
- mod_certheaders Module
- mod_ossl Module

- mod_wl_ohs Module
  The `mod_wl_ohs` module is a key feature of Oracle HTTP Server that enables requests to be proxied from Oracle HTTP Server to Oracle WebLogic Server. This module is generally referred to as the Oracle WebLogic Server proxy plug-in.
- mod_certheaders Module
  The `mod_certheaders` module enables reverse proxies using two directives namely, `AddCertHeader` and `SimulateHttps`.
- mod_ossl Module
  The `mod_ossl` module enables strong cryptography for Oracle HTTP Server. It accepts a set of directives such as `SSLCARevocationFile`, `SSLCipherSuite`, `SSLEngine`, and more.

mod_wl_ohs Module

The `mod_wl_ohs` module is a key feature of Oracle HTTP Server that enables requests to be proxied from Oracle HTTP Server to Oracle WebLogic Server. This module is generally referred to as the Oracle WebLogic Server proxy plug-in.

The mod_wl_ohs module enhances an Oracle HTTP server installation by allowing Oracle WebLogic Server to handle requests that require dynamic functionality. In other words, you typically use a plug-in where the HTTP server serves static pages such as HTML pages, while Oracle WebLogic Server serves dynamic pages such as HTTP Servlets and Java Server Pages (JSPs). For information on this module's directives, see Parameters for Web Server Plug-Ins in *Using Oracle WebLogic Server Proxy Plug-Ins*.

mod_certheaders Module

The `mod_certheaders` module enables reverse proxies using two directives namely, `AddCertHeader` and `SimulateHttps`.

This section describes the mod_certheaders directives:

- AddCertHeader Directive
- SimulateHttps Directive

AddCertHeader Directive

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

SimulateHttps Directive

You can use mod_certheaders 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`
Default	`off`

mod_ossl Module

The `mod_ossl` module enables strong cryptography for Oracle HTTP Server. It accepts a set of directives such as `SSLCARevocationFile`, `SSLCipherSuite`, `SSLEngine`, and more.

To configure SSL for your Oracle HTTP Server, enter the `mod_ossl` module directives you want to use in the `ssl.conf` file.

The following sections describe these `mod_ossl` directives:

- SSLInsecureRenegotiation Directive
- SSLOptions Directive
- SSLProtocol Directive
- SSLProxyCipherSuite Directive
- SSLProxyEngine Directive
- SSLProxyProtocol Directive
- SSLProxyWallet Directive
- SSLRequire Directive
- SSLRequireSSL Directive
- SSLSessionCache Directive
- SSLSessionCacheTimeout Directive
- SSLTraceLogLevel Directive
- SSLVerifyClient Directive
- SSLWallet Directive
- SSLCARevocationFile Directive
- SSLCARevocationPath Directive
- SSLCipherSuite Directive
  Specifies the SSL cipher suite that the client can use during the SSL handshake. This directive uses either a comma-separated or colon-separated cipher specification string to identify the cipher suite.
- SSLEngine Directive
- SSLFIPS Directive
- SSLHonorCipherOrder Directive
- SSLInsecureRenegotiation Directive
- SSLOptions Directive
- SSLProtocol Directive
- SSLProxyCipherSuite Directive
- SSLProxyEngine Directive
- SSLProxyProtocol Directive
- SSLProxyWallet Directive
- SSLRequire Directive
- SSLRequireSSL Directive
- SSLSessionCache Directive
- SSLProxySessionCache Directive
- SSLSessionCacheTimeout Directive
- SSLTraceLogLevel Directive
- SSLVerifyClient Directive
- SSLWallet Directive

SSLCARevocationFile Directive

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

SSLCARevocationPath Directive

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.

This directive must point to a directory that contains the hash value of the CRL. To see the commands that allow you to create the hashes, see orapki in *Administering Oracle Fusion Middleware*.

Category	Value
Syntax	`SSLCARevocationPath` *`path/to/CRL_directory/`*
Example	`SSLCARevocationPath ${`*`ORACLE_INSTANCE`*`}/config/fmwconfig/components/${`*`COMPONENT_TYPE`*`}/instances/${COMPONENT_NAME}/keystores/crl`
Default	None

SSLCipherSuite Directive

Specifies the SSL cipher suite that the client can use during the SSL handshake. This directive uses either a comma-separated or colon-separated cipher specification string to identify the cipher suite.

SSLCipherSuite accepts the following prefixes:

- none: Adds the cipher to the list
- + : Adds the cipher to the list and places it in the correct location in the list
- - : Removes the cipher from the list (can be added later)
- ! : Removes the cipher from the list permanently

Tags are joined with prefixes to form a cipher specification string. Cipher suite tags are listed in Table H-1.

> **Note:**
>
> Cipher suites that use Rivest Cipher 4 (RC4) and Triple Data Encryption Standard (3DES) algorithms are deprecated from Oracle HTTP Server version 12.2.1.3 onwards due to known security vulnerabilities. These ciphers are removed from the SSLCipherSuite configuration of the default SSL port of Oracle HTTP Server. These ciphers are also removed from all supported cipher aliases except RC4 and 3DES aliases. If Oracle HTTP Server is managed through Enterprise Manager or WebLogic Scripting Tool, you cannot configure these cipher suites through these tools as these tools do not recognize the insecure RC4 and 3DES ciphers.
>
> To provide backward compatibility, Oracle HTTP Server enables the RC4 and 3DES ciphers, if you explicitly add them to the cipher suite configuration. To use these insecure ciphers, edit the SSLCipherSuite directive in your .conf files using a file editor, and then add them to the end of the cipher list.

Table 11–2 shows the tags you can use in the string to describe the cipher suite you want.

Category	Value
Example	`SSLCipherSuite ALL:!MD5` In this example, all ciphers are specified except MD5 strength ciphers.
Syntax	`SSLCipherSuite` *`cipher-spec`*

Category	Value
Default	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256, TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_GCM_SHA384, TLS_RSA_WITH_AES_128_GCM_SHA256, TLS_RSA_WITH_AES_256_CBC_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA256, SSL_RSA_WITH_AES_256_CBC_SHA, SSL_RSA_WITH_AES_128_CBC_SHA

Table H-1 SSLCipher Suite Tags

Function	Tag	Meaning
Key exchange	kRSA	RSA key exchange
Key exchange	kECDHE	Elliptic curve Diffie–Hellman Exchange key exchange
Authentication	aRSA	RSA authentication
Encryption	3DES	Triple DES encoding
Encryption	RC4	RC4 encoding
Data Integrity	SHA	SHA hash function
Data Integrity	SHA256	SHA256 hash function
Data Integrity	SHA384	SHA384 hash function
Aliases	TLSv1	All TLS version 1 ciphers

Table H-1 (Cont.) SSLCipher Suite Tags

Function	Tag	Meaning
Aliases	TLSv1.1	All TLS version 1.1 ciphers
Aliases	TLSv1.2	All TLS version 1.2 ciphers
Aliases	MEDIUM	All ciphers with 128-bit encryption
Aliases	HIGH	All ciphers with encryption key size greater than 128 bits
Aliases	AES	All ciphers using AES encryption
Aliases	RSA	All ciphers using RSA for both authentication and key exchange
Aliases	ECDSA	All ciphers using Elliptic Curve Digital Signature Algorithm for authentication
Aliases	ECDHE	All ciphers using Elliptic curve Diffie–Hellman Exchange for key exchange
Aliases	AES-GCM	All ciphers that use Advanced Encryption Standard in Galois/Counter Mode (GCM) for encryption.

Table H-2 lists the Cipher Suites supported in Oracle Advanced Security 12*c* (12.2.1).

> **Note:**
>
> When using `mod_ossl` on a Solaris Sparc platform, the underlying cryptographic libraries detect the Sparc T4 processor, and makes use of the on-core cryptography algorithms that accelerate cryptographic operations. No configuration is required to enable this feature. The following cryptographic algorithms are supported by the Oracle Sparc Enterprise T-series processors: RSA, 3DES, AES-CBC, AES-GCM, SHA1, SHA256, and SHA38.

Table H-2 Cipher Suites Supported in Oracle Advanced Security 12.2.1

Cipher Suite	Key Exchange	Authentication	Encryption	Data Integrity	TLS v1	TLS v1.1	TLS v1.2
SSL_RSA_WITH_RC4_128_SHA	RSA	RSA	RC4 (128)	SHA	Yes	Yes	Yes
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	RSA	3DES (168)	SHA	Yes	Yes	Yes
SSL_RSA_WITH_AES_128_CBC_SHA	RSA	RSA	AES (128)	SHA	Yes	Yes	Yes
SSL_RSA_WITH_AES_256_CBC_SHA	RSA	RSA	AES (256)	SHA	Yes	Yes	Yes
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	RSA	AES (128)	SHA256	No	No	Yes

Table H-2 (Cont.) Cipher Suites Supported in Oracle Advanced Security 12.2.1

Cipher Suite	Key Exchange	Authentication	Encryption	Data Integrity	TLS v1	TLS v1.1	TLS v1.2
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	RSA	AES (256)	SHA256	No	No	Yes
TLS_RSA_WITH_AES_128_GCM_SHA256	RSA	RSA	AES (128)	SHA256	No	No	Yes
TLS_RSA_WITH_AES_256_GCM_SHA384	RSA	RSA	AES (256)	SHA384	No	No	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDHE	ECDSA	AES (128)	SHA	Yes	Yes	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	ECDHE	ECDSA	AES (256)	SHA	Yes	Yes	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	ECDHE	ECDSA	AES (128)	SHA256	No	No	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	ECDHE	ECDSA	AES (256)	SHA384	No	No	Yes
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	ECDHE	ECDSA	AES (128)	SHA256	No	No	Yes
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	ECDHE	ECDSA	AES (256)	SHA384	No	No	Yes
TLS_ECDHE_RSA_WITH_RC4_128_SHA	Ephemeral ECDH with RSA signatures	RSA	RC4 (128)	SHA	Yes	Yes	Yes
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	Ephemeral ECDH with RSA signatures	RSA	3DES	SHA	Yes	Yes	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Ephemeral ECDH with RSA signatures	RSA	AES (128)	SHA	Yes	Yes	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Ephemeral ECDH with RSA signatures	RSA	AES (256)	SHA	Yes	Yes	Yes
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	Ephemeral ECDH with ECDSA signatures	ECDSA	RC4 (128)	SHA	Yes	Yes	Yes
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	Ephemeral ECDH with ECDSA signatures	ECDSA	3DES	SHA	Yes	Yes	Yes

Table H-2 (Cont.) Cipher Suites Supported in Oracle Advanced Security 12.2.1

Cipher Suite	Key Exchange	Authentication	Encryption	Data Integrity	TLS v1	TLS v1.1	TLS v1.2
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Ephemeral ECDH with RSA signatures	RSA	AES (256)	SHA384	No	No	Yes
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Ephemeral ECDH with RSA signatures	RSA	AES (128)	SHA256	No	No	Yes
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Ephemeral ECDH with RSA signatures	RSA	AES (256)	SHA384	No	No	Yes
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Ephemeral ECDH with RSA signatures	RSA	AES (128)	SHA256	No	No	Yes

SSLEngine Directive

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.

Category	Value
Syntax	`SSLEngine on|off`
Example	`SSLEngine on`
Default	`Off`

SSLFIPS Directive

This directive toggles the usage of the SSL library FIPS_mode flag. It must be set in the global server context and should not be configured with conflicting settings (`SSLFIPS on` followed by `SSLFIPS off` or similar). The mode applies to all SSL library operations.

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 results in an error.

> **Note:**
>
> Note the following restriction on SSLFIPS:
>
> - Enabling SSLFIPS mode in Oracle HTTP Server requires a wallet created with AES encrypted (compat_v12) headers. To create a new wallet or to convert an existing wallet with AES encryption, see these sections in orapki in *Administering Oracle Fusion Middleware*:
>
> Creating and Viewing Oracle Wallets with orapki
>
> Creating an Oracle Wallet with AES Encryption
>
> Converting an Existing Wallet to Use AES Encryption

The following tables describe the cipher suites that work in SSLFIPS mode with various protocols. For instructions on how to implement these cipher suites, see SSLCipherSuite Directive.

Table H-3 lists the cipher suites which work in TLS 1.0, TLS1.1, and TLS 1.2 protocols in SSLFIPS mode.

Table H-3 Ciphers Which Work in All TLS Protocols in SSLFIPS Mode

Cipher Name	Cipher Works in These Protocols:
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0, TLS1.1, and TLS 1.2
SSL_RSA_WITH_AES_128_CBC_SHA	TLS 1.0, TLS1.1, and TLS 1.2
SSL_RSA_WITH_AES_256_CBC_SHA	TLS 1.0, TLS1.1, and TLS 1.2

Table H-4 lists the cipher suites and protocols that can be used in SSLFIPS mode.

Table H-4 Ciphers Which Work in FIPS Mode

Cipher Name	Cipher Works in These Protocols:
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0 and later
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	TLS 1.0 and later
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	TLS 1.0 and later
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	TLS1.2 and later
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	TLS1.2 and later
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	TLS1.2 and later
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS1.2 and later
TLS_RSA_WITH_AES_128_CBC_SHA256	TLS1.2 and later
TLS_RSA_WITH_AES_256_CBC_SHA256	TLS1.2 and later

Table H-4 (Cont.) Ciphers Which Work in FIPS Mode

Cipher Name	Cipher Works in These Protocols:
TLS_RSA_WITH_AES_128_GCM_SHA256	TLS1.2 and later
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS1.2 and later
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	TLS1.2 and later
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	TLS1.2 and later
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS1.2 and later
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS1.2 and later
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0 and later
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	TLS 1.0 and later
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	TLS 1.0 and later

Note:

- If SSLFIPS is set to ON, and a cipher that does not support FIPS is used at the server, then client requests that use that cipher fail.
- To use the TLS_ECDHE_ECDSA cipher suite, Oracle HTTP Server requires a wallet created with an ECC user certificate. The TLS_ECDHE_ECDSA cipher suite does not work with RSA certificates.
- To use the SSL_RSA/TLS_RSA/TLS_ECDHE_RSA cipher suite, Oracle HTTP Server requires a wallet created with an RSA user certificate. The SSL_RSA/TLS_RSA/TLS_ECDHE_RSA cipher suite does not work with ECC certificates.

For more information about how to configure ECC/RSA certificates in a wallet, see *Creating and Viewing Oracle Wallets with orapki* in *Administering Oracle Fusion Middleware*.

For instructions about how to implement these cipher suites and corresponding protocols, see SSL Cipher Suite Directive and SSL Protocol.

Table H-5 lists the cipher suites that do not work in SSPFIPS mode.

Table H-5 Ciphers That Do Not Work in SSLFIPS Mode

Cipher Name	Description
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	Does not work in SSLFIPS mode in any protocol
SSL_RSA_WITH_RC4_128_SHA	Does not work in SSLFIPS mode in any protocol
TLS_ECDHE_RSA_WITH_RC4_128_SHA	Does not work in SSLFIPS mode in any protocol

SSLHonorCipherOrder Directive

When choosing a cipher during a handshake, normally the client's preference is used. If this directive is enabled, then the server's preference will be used instead.

Category	Value
Syntax	`SSLHonorCipherOrder ON | OFF`
Example	`SSLHonorCipherOrder ON`
Default	`OFF`

The server's preference order can be configured using the SSLCipherSuite directive. When SSLHonorCipherOrder is set to ON, the value of SSLCipherSuite is treated as an ordered list of cipher values.

Cipher values that appear first in this list are preferred by the server over ciphers that appear later in the list.

Example:

```
SSLCipherSuite
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,T
LS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

SSLHonorCipherOrder ON
```

In this case, the server will prefer `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` over all of the other ciphers configured in SSLCipherSuite directive as it appears first in the list and chooses this cipher for the SSL connection, if the client supports it.

SSLInsecureRenegotiation Directive

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

The accepted values for this directive are:

- Default mode: When the directive SSLInsecureRenegotion is not specified in the configuration, Oracle HTTP Server does not allow client-initiated renegotiation. This is the most secure mode of operation.
- `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`: This option can be used if support for client-initiated renegotiation is desired. When SSLInsecureRenegotiation directive is

present in the configuration and set to OFF, Oracle HTTP Server allows client-initiated renegotiation. However, only peers that support RI/SCSV will be allowed to negotiate and renegotiate a session.

Category	Value
Syntax	`SSLInsecureRenegotiation ON | OFF`
Example	`SSLInsecureRenegotiation ON`
Default	The default value is neither `ON` nor `OFF`. See description under the heading **Default mode**.

To configure SSLInsecureRenegotiation, edit the `ssl.conf` file and set `SSLInsecureRenegotiation` ON/OFF to enable or disable insecure renegotiation. This directive may be configured either in the server config context or in the virtual host context.

SSLOptions Directive

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. No password is obtained from the user; the string 'password' is substituted.
- `StrictRequire`: Denies access when, according to SSLRequireSSL Directive 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

SSLProtocol Directive

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:

- 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 +TLSv1.2.

Category	Value
Syntax	SSLProtocol [+-] TLSv1 | TLSv1.1 | TLSv1.2 | All
Example	SSLProtocol *+TLSv1 +TLSv1.1 +TLSv1.2*
Default	TLSv1.2

SSLProxyCipherSuite Directive

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 H-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 places it in the correct location in the list
- - : Removes the cipher from the list (which can be added later)
- ! : Removes the cipher from the list permanently

Tags are joined with prefixes to form a cipher specification string. Tags are joined together 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 H-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`

The SSLProxyCipherSuite directive uses the same cipher suites as the SSLCipherSuite directive. For a list of the Cipher Suites supported in Oracle Advanced Security 12.2.1, see Table H-2.

SSLProxyEngine Directive

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`

SSLProxyProtocol Directive

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:

- 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 `+TLSv1 +TLSv1.1 +TLSv1.2`.

Category	Value
Syntax	`SSLProxyProtocol [+-] TLSv1 | TLSv1.1 | TLSv1.2 | All`
Example	`SSLProxyProtocol *+TLSv1 +TLSv1.1 +TLSv1.2*`
Default	`ALL`

SSLProxyWallet Directive

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*`
Example	`SSLProxyWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/proxy"`
Default	None

SSLRequire Directive

> **Note:**
>
> SSLRequire is deprecated and must be replaced with Require expression.

Denies access unless an arbitrarily complex boolean expression is true.

Category	Value
Syntax	`SSLRequire *expression*`
Example	`SSLRequire word ">=" word |word "ge" word`
Default	None

Understanding the Expression Variable

The *expression* variable 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 H-6 and Table H-7 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 H-6 lists the standard variables for SSLRequire Directive `varname`.

Table H-6 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

Table H-6 (Cont.) Standard Variables for SSLRequire Varname

Standard Variables	Standard Variables	Standard Variables
REQUEST_FILENAME		

Table H-7 lists the SSL variables for SSLRequire Directive varname.

Table H-7 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
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	

SSLRequireSSL Directive

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

SSLSessionCache Directive

Specifies the global/interprocess session cache storage type. The cache provides an optional way to speed up parallel request processing. The accepted values are:

- `none`: disables the global/interprocess session cache. Produces no impact on functionality, but makes a major difference in performance.
- `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. Note: in this shm setting, no log files are created under /path/to/datafile on local disk.

Category	Value
Syntax	`SSLSessionCache none | shmcb:/path/to/datafile[bytes]`
Examples	`SSLSessionCache "shmcb:${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/ssl_scache(512000)"`
Default	`SSLSessionCache shmcb:/path/to/datafile[bytes]`

SSLProxySessionCache Directive

This directive toggles the usage of a global or interprocess session cache to cache SSL session information when OHS is configured to behave as a proxy through the use of the `mod_proxy` module. The type of global or interprocess SSL session cache that is used to store the SSL session information is controlled by the SSLSessionCache directive.

The number of seconds before an SSL session expires in the session cache is controlled by the SSLSessionCacheTimeout directive. The ssl-cache mutex is used to serialize access to the session cache to prevent corruption.

The accepted values for SSLProxySessionCache directive are:

- `On`: Enables the SSL session cache when OHS is configured to behave as a proxy through the use of the `mod_proxy` module. When this directive is set to `On`, it is necessary to choose the type of SSL session cache by configuring the SSLSessionCache directive appropriately.

  SSLSessionCache cannot be set to a value of `none`, as it would be a conflicting setting to turn on SSL session cache for the proxy and not specify the type of cache to use.
- `Off`: Disables SSL session caching when OHS is configured to behave as a proxy through the use of the mod_proxy module. This is not a recommended setting. The performance costs of full handshakes must be considered before choosing this option.

Category	Value
Syntax	`SSLProxySessionCache On | Off`
Context	Server Config
Default	`On`
Module Identifier	`ossl_module`

The following examples illustrate how to use the SSLSessionCache and SSLProxySessionCache directives to control the SSL session caching behaviour of OHS.

Example 1

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"
LoadModule proxy_balancer_module
"${PRODUCT_HOME}/modules/mod_proxy_balancer.so"

SSLSessionCache none
SSLProxySessionCache on
<VirtualHost _default_:443>
SSLEngine on
   <Proxy "balancer://mybalancer">
SSLProxyEngine On
    #..
    </Proxy>
#...
</VirtualHost>
```

This is not an allowed configuration. SSLProxySessionCache cannot be turned on when SSLSessionCache is set to `None`.

Example 2

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"
LoadModule proxy_balancer_module
"${PRODUCT_HOME}/modules/mod_proxy_balancer.so"

SSLSessionCache none
SSLProxySessionCache off
<VirtualHost _default_:443>
SSLEngine on
   <Proxy "balancer://mybalancer">
    SSLProxyEngine On
    #..
    </Proxy>
#...
</VirtualHost>
```

This example

- Turns off SSL session caching for the SSL enabled virtual host defined within the OHS configuration.

- Turns off SSL client session caching for requests handled by the proxy `mybalancer`.

Example 3

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"
LoadModule proxy_balancer_module
"${PRODUCT_HOME}/modules/mod_proxy_balancer.so"

SSLSessionCache
"shmcb:${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/ssl_scache(512000)"
SSLProxySessionCache off
<VirtualHost _default_:443>
SSLEngine on
   <Proxy "balancer://mybalancer">
    SSLProxyEngine On
    #..
    </Proxy>
#...
</VirtualHost>
```

This example

- Turns on SSL session caching for the SSL enabled virtual host defined within the OHS configuration.
- Turns off SSL client session caching for requests handled by the proxy `mybalancer`.

Example 4

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"

SSLSessionCache
"shmcb:${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/ssl_scache(512000)"
SSLProxySessionCache off

<VirtualHost _default_:443>
SSLEngine on
SSLProxyEngine on

ProxyPass / https://<backend_host_name>:<backend_port>/
ProxyPassReverse / https://<backend_host_name>:<backend_port>/

</VirtualHost>
```

This example

- Turns on SSL session caching for the SSL enabled virtual host defined within the OHS configuration.
- Turns off SSL client session caching for the requests handled by the reverse proxy defined within the virtual host (`_default_:443`).

Example 5

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"
LoadModule proxy_balancer_module
```

```
"${PRODUCT_HOME}/modules/mod_proxy_balancer.so"

SSLSessionCache
"shmcb:${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/
ssl_scache(512000)"
SSLProxySessionCache on
<VirtualHost _default_:443>
SSLEngine on
   <Proxy "balancer://mybalancer">
    SSLProxyEngine On
    #..
    </Proxy>
#...
</VirtualHost>
```

This example

- Turns on SSL session caching for the SSL enabled virtual host defined within the OHS configuration.
- Turns on SSL client session caching for requests handled by the proxy `mybalancer`.

Example 6

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"

SSLSessionCache
"shmcb:${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/
ssl_scache(512000)"
SSLProxySessionCache on

<VirtualHost _default_:443>
SSLEngine on
SSLProxyEngine on

ProxyPass / https://<backend_host_name>:<backend_port>/
ProxyPassReverse / https://<backend_host_name>:<backend_port>/

</VirtualHost>
```

This example

- Turns on SSL session caching for the SSL enabled virtual host defined within the OHS configuration.
- Turns on SSL client session caching for the requests handled by the reverse proxy defined within the virtual host (`_default_:443`).

Example 7

```
LoadModule proxy_module "${PRODUCT_HOME}/modules/mod_proxy.so"
LoadModule proxy_balancer_module
"${PRODUCT_HOME}/modules/mod_proxy_balancer.so"

SSLSessionCache
```

```
"shmcb:${ORACLE_INSTANCE}/servers/${COMPONENT_NAME}/logs/ssl_scache(512000)"
SSLProxySessionCache on
<VirtualHost _default_:443>
SSLEngine on
   <Proxy "balancer://mybalancer">
    SSLProxyEngine On
    #..
    </Proxy>

    <Proxy "balancer://mybalancer2">
    SSLProxyEngine On
    #..
    </Proxy>

#...
</VirtualHost>

<VirtualHost _default_:4448>
SSLEngine on
   <Proxy "balancer://mybalancer3">
    SSLProxyEngine On
    #..
    </Proxy>

    <Proxy "balancer://mybalancer4">
    SSLProxyEngine On
    #..
    </Proxy>

#...
</VirtualHost>
```

This example

- Turns on SSL session caching for all the SSL enabled virtual hosts defined within the OHS configuration.
- Turns on SSL client session caching for requests handled by the proxy `mybalancer`, `mybalancer2`, `mybalancer3`, and `mybalancer4`.

SSLSessionCacheTimeout Directive

Specifies the number of seconds before a SSL session in the session cache expires.

Category	Value
Syntax	`SSLSessionCacheTimeout` *seconds*
Example	`SSLSessionCacheTimeout 120`
Default	`300`

SSLTraceLogLevel Directive

`SSLTraceLogLevel` adjusts the verbosity of the messages recorded in the Oracle Security 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 `SSLTraceLogLevel ssl` is set, messages with log levels of error, warn, user and debug will also be posted.

> **Note:**
>
> This directive can only be set globally in the `ssl.conf` file.

SSLTraceLogLevel accepts the following log levels:

- `none`: Oracle Security 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	`SSLTraceLogLevel none | fatal | error | warn | user | debug | ssl`
Example	`SSLTraceLogLevel fatal`
Default	None

SSLVerifyClient Directive

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`

Category	Value
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`.

SSLWallet Directive

Specifies the location of the wallet with its WRL, specified as a filepath.

Category	Value
Syntax	`SSLWallet` *`file:path to wallet directory`* `file:path` may also be expressed simply as `path`.
Example	`SSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/default"`
Default	This is the default

> **Note:**
>
> If the wallet has a certificate/certificate request signed with the MD5 algorithm, Oracle HTTP Server will fail to start.