Oracle® Fusion Middleware

Using Oracle WebLogic Server Proxy Plug-Ins

14c (14.1.2.0.0)
F87194-01
December 2024

ORACLE

Oracle Fusion Middleware Using Oracle WebLogic Server Proxy Plug-Ins, 14c (14.1.2.0.0)

F87194-01

Primary Author: Oracle Corporation

Contents

3 Installing and Configuring the Oracle WebLogic Server Proxy Plug-In for Apache HTTP Server

4 Configuring Security

5 Parameters for Oracle WebLogic Server Proxy Plug-Ins

Preface

This preface describes the document accessibility features and conventions used in this guide —*Using Oracle WebLogic Server Proxy Plug-Ins*.

Documentation Accessibility

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

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

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.

Conventions

The following text conventions are used in this document:

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

1

Overview of Oracle WebLogic Server Proxy Plug-Ins

Oracle provides proxy plug-ins for use with Oracle WebLogic Server.

This chapter includes the following topics:

- What are Oracle WebLogic Server Proxy Plug-Ins?
  Oracle WebLogic Server proxy plug-ins (**WLS proxy plug-ins**) allow requests to be proxied from an HTTP web server to Oracle WebLogic Server. In this way, proxy plug-ins enable the HTTP server to communicate with applications deployed on Oracle WebLogic Server.
- Availability of Oracle WebLogic Server Proxy Plug-Ins
- New Features of the 14.1.2.0.0 Proxy Plug-Ins
  The Oracle WebLogic Server 14.1.2.0.0 proxy plug-ins add support for the following new features.
- Features Inherited from Previous 12c Releases
  In addition to the new features, WLS proxy plug-ins have also inherited features from the previous 12c releases.

What are Oracle WebLogic Server Proxy Plug-Ins?

Oracle WebLogic Server proxy plug-ins (**WLS proxy plug-ins**) allow requests to be proxied from an HTTP web server to Oracle WebLogic Server. In this way, proxy plug-ins enable the HTTP server to communicate with applications deployed on Oracle WebLogic Server.

A proxy plug-in enhances an HTTP server installation by allowing Oracle WebLogic Server to handle requests that require dynamic functionality. In other words, you typically use a proxy 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 Jakarta Server Pages (JSPs).

Oracle WebLogic Server may be operating in a different process, possibly on a different host. To the end user—the browser—the HTTP requests delegated to Oracle WebLogic Server still appear to be coming from the HTTP server.

Availability of Oracle WebLogic Server Proxy Plug-Ins

Oracle WebLogic Server 14.1.2.0.0 proxy plug-ins are available for Oracle HTTP Server and Apache HTTP Server.
The WLS proxy plug-ins are the Oracle WebLogic Server Proxy Plug-In for Oracle HTTP Server (**WLS OHS Plug-In**) and the Oracle WebLogic Server Proxy Plug-In for Apache HTTP Server (**WLS Apache Plug-In**), respectively.

Table 1-1 Availability of Version 14c (14.1.2.0.0) Plug-Ins

Web Server	Plug-In Availability	More Information
Oracle HTTP Server 14c	The WLS OHS Plug-In is included in the Oracle HTTP Server installation.	For information about configuring the WLS OHS Plug-In, see Configuring the Plug-In for Oracle HTTP Server.
Apache HTTP Server 2.4.x	The WLS Apache Plug-In is available for download on My Oracle Support (https://support.oracle.com/signin) and the Software Delivery Cloud (http://edelivery.oracle.com) web sites as ZIP files. Available for download is the WLS Apache Plug-In ZIP file, compiled with OpenSSL 1.1.1x version and OpenSSL 3.0.x. For more information, see the Oracle WebLogic Server (14.1.2.0.0) Certification Matrix.	For information about installing and configuring the WLS Apache Plug-In, see Installing and Configuring the Oracle WebLogic Server Proxy Plug-In for Apache HTTP Server.

New Features of the 14.1.2.0.0 Proxy Plug-Ins

The Oracle WebLogic Server 14.1.2.0.0 proxy plug-ins add support for the following new features.

Table 1-2 New Features of the Oracle WebLogic Server Proxy Plug-Ins

Plug-In	Functionality
WLS OHS Plug-In	• Support for Intelligent Load Balancing • Support for TLSv1.3 Protocol
WLS Apache Plug-In	• Support for HTTP/2 Protocol (also provided in 14.1.1.0.0) • Support for Intelligent Load Balancing • Support for TLSv1.3 Protocol (also provided in 14.1.1.0.0)

- Support for HTTP/2 Protocol
- Support for Intelligent Load Balancing
- Security Improvements

Support for HTTP/2 Protocol

> **Note:**
>
> HTTP/2 support is for the WLS Apache Plug-In *only*.

The HTTP/2 protocol uses a binary framing mechanism to exchange data between the client and the server. All HTTP/2 communication is split into smaller messages and frames, each of which is encoded in a binary format. As a result, both client and server must use the new

binary encoding mechanism to understand each other. An HTTP/1.x client will not understand an HTTP/2-only server, and vice versa.

The Apache Web Server supports the HTTP/1.1 and HTTP/2 protocols for both:

- Clients connecting to Apache Web Server (front-end connections)
- WLS Apache Plug-In connections to WebLogic Server (back-end connections)

> **Note:**
>
> WebSocket is not supported over HTTP/2.

Support for Intelligent Load Balancing

Intelligent load balancing enables WLS proxy plug-ins to more evenly distribute traffic across a pool of servers according to their actual capacity, for improved reliability.

> **Note:**
>
> For Intelligent Load Balancing, only Oracle WebLogic Server 14.1.2.0.0 back-end servers are supported.

The intelligent load balancing features consists of two parts:

- The calculation of the health score of a WebLogic Managed Server (Managed Server)
- The selection of a Managed Sever to route the request

Calculating the Health Score

For each Managed Server in a cluster, WebLogic Server provides a default health score calculation. The default health score calculation is based on CPU load, heap usage, Work Manager stuck threads count, and data source pending connection request counts. This health score is calculated individually, by each Managed Server, and then returned to the proxy plug-in when requested. For more information on how the health score is calculated in WebLogic Server, see Health Score-Based Intelligent Routing in *Administering Server Environments for Oracle WebLogic Server*.

WLS proxy plug-ins request the health scores of Managed Servers through the request header `X-WebLogic-Request-Server-Health-Score`.

WebLogic Server sends the health score of each Managed Server through the predefined response header `X-WebLogic-Server-Health-Score`.

```
<Location /sampleApp>
    WLSRequest On
    WebLogicCluster host1:port1,host2:port2,host3:port3
    RoutingAlgorithm Weighted-Least-Connection
    WeightUpdateInterval 7
</Location>
```

Selecting a Managed Server

In addition to the health score, WLS proxy plug-ins also use the active connection count parameter to select which Managed Server from the cluster to route the request.

The active connection count is stored at the proxy plug-in level. The active connection count represents how many requests from the proxy plug-in side are currently being served by a Managed Server.

WLS proxy plug-ins use the weighted least connection routing algorithm to select the next server from cluster. The algorithm selects the Managed Server with the lowest ratio of active connection count and Managed Server health score. This ensures that Managed Servers with higher capacities receive a proportionally larger share of the workload, while preventing overloading of less capable servers.

Configuring Intelligent Load Balancing

To enable intelligent load balancing, you must configure settings in both the WLS proxy plug-ins and WebLogic Server. If you do not configure intelligent load balancing, the round-robin routing algorithm is used by default.

In the proxy plug-in, set the `RoutingAlgorithm` directive to `Weighted-Least-Connection`. For more information, see RoutingAlgorithm.

By default, WLS proxy plug-ins request the health score of a Managed Server in one second intervals, when the request is selected to be served by a Managed Server. To reduce the frequency, configure the `WeightUpdateInterval` directive.

The following module needs to be loaded.

```
LoadModule socache_shmcb_module "${PRODUCT_HOME}/modules/mod_socache_shmcb.so"
```

Sample Configuration

```
<Location /sampleApp>
    WLSRequest On
    WebLogicCluster host1:port1,host2:port2,host3:port3
    RoutingAlgorithm Weighted-Least-Connection
    WeightUpdateInterval 7
</Location>
```

For instructions for configuring WebLogic Server, see Configuring the Health Score in *Administering Server Environments for Oracle WebLogic Server*.

Security Improvements

The following security enhancements have been made to the WLS proxy plug-ins.

Support for TLSv1.3 Protocol

WLS proxy plug-ins support the TLSv1.3 protocol.

Use the WebLogicSSLVersion directive to specify the SSL protocol version to be used for communication between the proxy plug-in and Oracle WebLogic Server.

The support for TLSv1.3 protocol includes the following features:

- Support for new TLSv1.3 cipher suites. See WebLogicSSLCiphers.
- Support for certificates signed with the RSASSA-PSS signature algorithm. See Using Certificates Signed with RSASSA-PSS Signature Algorithm.

TLS Ciphers

A few ciphers were deprecated in previous releases and are removed from the list of supported ciphers. If you want to use the deprecated ciphers for a handshake between the web server and the Oracle WebLogic Server, you must explicitly add them to the configuration using the `WebLogicSSLCiphers` directive. A warning message is generated if any cipher from the deprecated list is used.

- Default list of ciphers for TLSv1.3:
  - TLS_AES_256_GCM_SHA384
  - TLS_AES_128_GCM_SHA256
  - TLS_CHACHA20_POLY1305_SHA256
- Default list of ciphers for TLSv1.2:
  - ECDHE-RSA-AES256-GCM-SHA384
  - ECDHE-RSA-AES128-GCM-SHA256
  - ECDHE-ECDSA-AES256-GCM-SHA384
  - ECDHE-ECDSA-AES128-GCM-SHA256
- Deprecated list of ciphers in 14.1.2.0.0:
  - AES128-GCM-SHA256
  - AES256-GCM-SHA384
  - AES128-SHA256
  - AES256-SHA256
  - AES256-SHA
  - AES128-SHA

Features Inherited from Previous 12c Releases

In addition to the new features, WLS proxy plug-ins have also inherited features from the previous 12c releases.

The inherited features include the following:

- The WLS Apache Plug-In supports Apache HTTP Server 2.4.x Web Server through the `mod_wl_24.so` proxy plug-in module. So, you will need to load the `mod_wl_24.so` module with Apache HTTP Server 2.4.x. This is typically done by editing the Apache HTTP Server configuration file(s).
- The WLS Apache Plug-In does not support Apache HTTP Server 2.2.x through the `mod_wl.so` Oracle WebLogic Server module. Hence, this module has been removed from the proxy plug-in distribution.
- Oracle WebLogic Server supports deploying WebSocket applications. The WLS OHS Plug-In can now handle WebSocket connection upgrade requests and effectively proxy to WebSocket applications hosted within Oracle WebLogic Server 12.1.2 and later. See About WebSocket Proxy Configurations.

- The proxy plug-in now includes the following WLS OHS Plug-In configuration parameters:
  - `WLMaxWebSocketClients`: Limits the number of active WebSocket connections at any instant of time. The default value is `Half of MaxClients` (or `MaxRequestWorkers`).
  - `WebLogicSSLVersion`: Chooses the SSL protocol version to use while communicating HTTPS requests between the WLS OHS Plug-In and WebLogic Managed Servers and Clusters.
- The WLS proxy plug-ins provide support for monitoring the performance of the WLS proxy plug-ins where a request is proxied to the back-end Oracle WebLogic Server. See Understanding the DMS Metrics for the WLS Apache Plug-In.
- The WLS proxy plug-ins now log the debug information to the respective web server error log files. Hence, the proxy plug-in parameters specific to the debug logs (`Debug` and `WLLogFile`) have been deprecated.
- The WLS proxy plug-ins improve performance using a pool of connections from the plug-in to Oracle WebLogic Server. The proxy plug-in implements HTTP 1.1 keep-alive connections between the proxy plug-in and Oracle WebLogic Server by reusing the same connection for subsequent requests from the same proxy plug-ins. If the connection is inactive for more than 20 seconds, (or a user-defined amount of time), the connection is closed. See KeepAliveEnabled.

  > **Note:**
  >
  > The web server manages client connections.

- The WLS proxy plug-ins proxy requests to Oracle WebLogic Server based on a configuration that you specify.
  - You can proxy requests based on the URL of the request or a portion of the URL. This is called proxying by path.
  - You can also proxy a request based on the MIME type of the requested file, which is called proxying by file extension.

  You can also enable both methods. If you enable both methods and a request matches both criteria, the request is proxied by path.

  You can also specify additional parameters for each of these types of requests that define additional behavior of the proxy plug-in.

The following features are no longer supported:

- The WLS OHS Plug-In has removed support for TLS1.0 SSL protocol. Therefore, the proxy plug-in fails to connect to Oracle WebLogic Server when you configure TLS1.0 SSL protocol for SSL communication.
- The WLS OHS Plug-In considers MD5 signed certificates as insecure. Therefore, support for these certificates has been removed. If you are using SSL to connect to Oracle WebLogic Server, and if the wallet contains any certificates signed with MD5, replace them by SHA-2 signed certificates. Otherwise, the server fails to start. For more information about MD5 signed certificates, see Replacing Certificates Signed Using the MD5 Algorithm.

2

Configuring the Plug-In for Oracle HTTP Server

The Oracle WebLogic Server Proxy Plug-In (WLS OHS Plug-In) is the plug-in for proxying requests from Oracle HTTP Server to Oracle WebLogic Server. The WLS OHS Plug-In is included in the Oracle HTTP Server 14c (14.1.2.0.0) installation. You do not have to download and install it separately.

You can configure the WLS OHS Plug-In either by using Fusion Middleware Control or by editing the `mod_wl_ohs.conf` configuration file manually.

> **Note:**
>
> The WLS OHS Plug-In is now able to front-end WebSocket applications.

This chapter includes the following topics:

Preparing for Configuring the WLS OHS Plug-In

You must complete some installation and verification tasks before configuring the WLS OHS Plug-In.

- Ensure that Oracle WebLogic Server has been installed, a domain has been created, and you can access the Oracle WebLogic Remote Console. Oracle HTTP Server and WebLogic Server can be installed either in same domain or in separate domains.
- If OHS is being used in a WebLogic managed domain, verify that Fusion Middleware Control has been installed and you can access the Enterprise Manager Fusion Middleware Control Console. This is required to configure the WLS OHS Plug-In by using the graphical interface provided by Fusion Middleware Control. The Fusion Middleware Control is available only for WebLogic managed domains.
- To be able to test the configuration, ensure that the required Java applications are deployed to Oracle WebLogic Server—either to a single managed server or to a cluster—and are accessible.

See the following topics:

- Setting the WebLogic Plug-In Enabled Parameter
- Understanding the WebLogic Plug-In Enabled Parameter

Setting the WebLogic Plug-In Enabled Parameter

You must set the WebLogic Plug-In Enabled parameter.

1. Log in to the WebLogic Remote Console.
2. In the **Edit Tree**, go to **Environment**, then:
   - If the server instances to which you want to proxy requests from Oracle HTTP Server are in a cluster, select **Clusters**.
   - Otherwise, select **Servers**.
3. Select the cluster or server to which you want to proxy requests from Oracle HTTP Server.
4. Click **Show Advanced Fields**.
5. Turn on the **WebLogic Plug-In Enabled** option to use WebLogic Plug-Ins with the WebLogic Server.

   If you selected **Servers** (and not Clusters), turn on **WebLogic Plug-In Enabled** for every server to which you want to proxy requests from Oracle HTTP Servers.
   See Understanding the WebLogic Plug-In Enabled Parameter.
6. Click **Save**.

For the change to take effect, you must restart the server instances.

Understanding the WebLogic Plug-In Enabled Parameter

The **WebLogic Plug-In Enabled** drop-down list contains these values:

- **Yes**—**Yes** must be selected if the WLS proxy plug-ins are used with WebLogic Server. When set to **Yes** on the server, it specifies that this server uses the proprietary `WL-Proxy-Client-IP` header, which is recommended if the server instance will receive requests from a proxy plug-in.

  When set to **Yes** on the cluster, it specifies that the cluster will receive requests from a proxy plug-in or `HttpClusterServlet`. A call to `getRemoteAddr` will return the address of the browser client from the proprietary `WL-Proxy-Client-IP` header, instead of the web server.
- **No**—Selecting **No** for the server or cluster disables the `weblogic-plugin-enabled` parameter (`weblogic-plugin-enabled=false`) in the `config.xml` file.
- **Default**—When **Default** is selected for **WebLogic Plug-In Enabled** in the servers page, then the servers will inherit the value selected for **WebLogic Plug-In Enabled** for the cluster. When **Default** is selected for **WebLogic Plug-In Enabled** in the clusters page, then the clusters will inherit the value selected for **WebLogic Plug-In Enabled** for the domain.

Configuring the WLS OHS Plug-In Using Fusion Middleware Control

If OHS is being used in a WebLogic managed domain, you may use Fusion Middleware Control to configure the `mod_wl_ohs` module.

To configure the `mod_wl_ohs` module, complete the following tasks:

Task 1: Navigate to the mod_wl_ohs Configuration Page

The `mod_wl_ohs` configuration page contains the parameters for configuring the WLS OHS Plug-In.

1. Ensure that you have fulfilled the prerequisites listed in Preparing for Configuring the WLS OHS Plug-In.
2. Select **Administration** from the Oracle HTTP Server menu.
3. Select **mod_wl_ohs Configuration** from the Administration menu. The **mod_wl_ohs Configuration** page appears.

   The following table describes the fields in the **mod_wl_ohs** page.

Field	Description
Provide WebLogic Cluster Details	List of Oracle WebLogic clusters that can be used for load balancing. The server or cluster list is a list of `host:port` entries. If a mixed set of clusters and single servers is specified, the dynamic list returned for this parameter will return only the clustered servers. If you are not sure of the correct cluster, you can click the search icon to see a list of all associated clusters. See Using the Search Function. The module does a simple round-robin between all available servers. The server list specified in this property is a starting point for the dynamic server list that the server and module maintain. Oracle WebLogic Server and the module work together to update the server list automatically with new, failed, and recovered cluster members. You can disable the use of the dynamic cluster list by disabling the **Dynamic Server List ON** field. The module directs HTTP requests containing a cookie, URL-encoded session, or a session stored in the POST data to the server in the cluster that originally created the cookie.

Field	Description
Provide WebLogic Server Host and Port Details	• WebLogic Host Oracle WebLogic Server host (or virtual host name as defined in Oracle WebLogic Server) to which HTTP requests should be forwarded. If you are using a WebLogic cluster, use the **WebLogic Cluster** parameter instead of **WebLogic Host**. If you are not sure of the correct server, you can click the search icon to see a list of all associated clusters. See Using the Search Function. • WebLogic Port Port at which the Oracle WebLogic Server host is listening for connection requests from the module (or from other servers). (If you are using SSL between the module and Oracle WebLogic Server, set this parameter to the SSL listen port.)
Dynamic Server List ON | OFF	When set to OFF, the module ignores the dynamic cluster list used for load balancing requests proxied from the module and only uses the static list specified with the WebLogic Cluster parameter. Normally this parameter should be set to ON. There are some implications for setting this parameter to OFF: • If one or more servers in the static list fails, the module could waste time trying to connect to a terminated server, resulting in decreased performance. • If you add a new server to the cluster, the module cannot proxy requests to the new server unless you redefine this parameter. Oracle WebLogic Server automatically adds new servers to the dynamic server list when they become part of the cluster.
Error Page	You can create your own error page to appear when your Web server cannot forward requests to Oracle WebLogic Server.
WebLogic Temp Directory	Specifies the location of the _wl_proxy directory for post data files.
Exclude Path or MIME Type	This parameter allows you exclude certain requests from proxying. This parameter can be defined locally at the Location tag level and globally. When the property is defined locally, it does not override the global property but defines a union of the two parameters.

Match Expressions	Use this region to specify any Expression overrides. For example, if you were proxying by MIME type, you might enter:
	`*.jsp WebLogicHost=myHost|paramName=value`
	You can define a new parameter for Match Expression by using the following syntax:
	`*.jsp PathPrepend=/test|PathTrim=/foo`
	(parameters are separated by a |)

Location	Use this table to specify any location overrides. See Task 3: Configure Expression Overrides or Location Overrides (Optional).

Field	Description
Add Cross Component Wiring	This button appears only if you have installed Oracle HTTP Server in full JRF mode (collocated) and there is a backing database. Selecting this button opens the Service Tables page. A service table provides a way for service providers to publish endpoint information about their services, and clients of these services to query and bind to these services. A service table is a single table in a database schema. There is one row for every endpoint that is published to it. The service table schema is initially created by the Repository Creation Utility. See Wiring Components to Work Together in *Administering Oracle Fusion Middleware*

Task 2: Specify the Configuration Settings

Specify the configuration settings for the WLS OHS Plug-In. In the General section, you can configure `mod_wl_ohs` for a WebLogic cluster or for WebLogic servers.

- If you select the **Provide WebLogic Cluster Details** radio button, then provide values for the WebLogic Cluster, Dynamic Server List ON, Error Page, WebLogic Temp Directory, and Exclude Path or MIME Type fields.
- If you select the **Provide WebLogic Server Host and Port Details** radio button, then provide values for the WebLogic Host, WebLogic Port, Dynamic Server List ON, Error Page, WebLogic Temp Directory, and Exclude Path or MIME Type fields.

Task 3: Configure Expression Overrides or Location Overrides (Optional)

If necessary, you can add expression or location overrides to your configuration.

1. Add any expression overrides in the **Match Expression** field.
2. Add any location overrides in the **Location** table.
   a. Click **Add Row** to create a new row.
   b. Enter the base URI for which the associated directives become effective.
   c. Complete the **WebLogic Cluster**, **WebLogic Host**, and **WebLogic Port** fields. You can automatically complete these fields by clicking **AutoFill** (see Using the AutoFill Function).
   d. Complete the **Path Trim** field.

      According to the RFC specification, generic syntax for URL is:

      `[PROTOCOL]://[HOSTNAME]:{PORT}/{PATH}/{FILENAME};{PATH_PARAMS}/{QUERY_STRING}...`

      Path Trim specifies the string trimmed by the module from the {PATH}/{*FILENAME*} portion of the original URL, before the request is forwarded to WebLogic Server. For example, if the URL:

      `http://myWeb.server.com/weblogic/foo`

      is passed to the module for parsing and if Path Trim has been set to strip off `/weblogic` before handing the URL to WebLogic Server, the URL forwarded to WebLogic Server is:

      `http://myWeb.server.com:7002/foo`

> **Note:**
>
> If you are converting an existing third-party server to proxy requests to WebLogic Server using the module for the first time, you must change application paths to /foo to include weblogic/foo. You can use Path Trim and Path Prepend in combination to change this path

e. Complete the **Path Prepend** field.

According to the RFC specification, generic syntax for URL is:

`[PROTOCOL]://[HOSTNAME]:{PORT}/{PATH}/{FILENAME};{PATH_PARAMS}/{QUERY_STRING}...`

Path Prepend specifies the path that the module prepends to the {*PATH*} portion of the original URL, after Path Trim is trimmed and before the request is forwarded to WebLogic Server.

> **Note:**
>
> If you need to append File Name, use the `DefaultFileName` module parameter instead of Path Prepend.

f. Click **Add Row** again to save the new row.

Task 4: Apply Your Changes

Apply your changes to the **mod_wl_ohs** Configuration Page and restart Oracle HTTP Server.

1. 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.
2. Restart Oracle HTTP Server by selecting **Control** from the Oracle HTTP Server menu, and then selecting **Start Up**.

The `mod_wl_ohs` module configuration is saved and displayed on the **mod_wl_ohs** Configuration page.

Using the Search Function

The search function allows you to search for a particular WebLogic Cluster or WebLogic Host that is available to the selected Oracle HTTP Server instance. By clicking the search icon  , you can see a list of clusters or servers available to the selected Oracle HTTP Server instance. To use the search function, do the following:

1. Click the search icon for either WebLogic Cluster or WebLogic Host. The Select WebLogic Cluster/Server dialog box appears.
2. Select the cluster or server you want to use and click **OK**.

The selected cluster or server name appears in the appropriate field.

Using the AutoFill Function

> **Note:**
>
> The AutoFill function is available only if you are using Oracle WebLogic Server in Full-JRF mode. It is not available if you are using Restricted-JRF.

You can easily add valid WebLogic Server and endpoint locations for a specified Base URL to the Locations table on the Oracle WebLogic Server Proxy Plug-In Configuration screen by using the AutoFill button. To do so:

1. Click **Add** to add a new location,
2. Enter a location name in the Location field.
3. Click **AutoFill**.

Data for any location of the same name will be updated and any new locations will be added to the table.

Configuring the WLS OHS Plug-In Manually

When OHS has been configured in standalone mode, specify directives in the `mod_wl_ohs.conf` file to manually configure the WLS OHS Plug-In.

1. Ensure that you have fulfilled the prerequisites listed in Preparing for Configuring the WLS OHS Plug-In.
2. Open the `mod_wl_ohs.conf` file in a text editor.

   The `mod_wl_ohs.conf` file is located in the following directory:

   `DOMAIN_HOME/config/fmwconfig/components/OHS/componentHome`
3. Add directives within the `<IfModule weblogic_module>` element in the configuration file.

   For examples, see Examples of <IfModule weblogic_module> Element Configurations.

   For information about the other directives that you can specify in the `mod_wl_ohs.conf` file, see Parameters for Oracle WebLogic Server Proxy Plug-Ins.
4. Restart Oracle HTTP Server by using one of the techniques described in Starting Oracle HTTP Server in *Administering Oracle HTTP Server*.

- Examples of <IfModule weblogic_module> Element Configurations

Examples of <IfModule weblogic_module> Element Configurations

The configuration of the predefined `<IfModule weblogic_module>` element determines how requests are sent to Oracle WebLogic Server. These examples demonstrates the different ways in which you can configure this element.

> **Note:**
>
> Oracle recommends that you specify directives within the predefined `<IfModule weblogic_module>` element.
>
> If you specify directives outside the predefined `<IfModule weblogic_module>` element, or in additional `<IfModule weblogic_module>` elements, or in configuration files other than mod_wl_ohs.conf, the Oracle WebLogic Server Proxy Plug-In might work, but the configuration state of the module, as displayed in Fusion Middleware Control, could be inconsistent with the directives specified in the `mod_wl_ohs.conf` configuration file.

To Forward Requests to a Single Oracle WebLogic Server Instance

To forward requests to an application running on a **single Oracle WebLogic Server instance**, specify the details of that destination server within a `<location>` element.

Syntax:

```
<IfModule weblogic_module>
<Location path>
WLSRequest On
WebLogicHost host
WeblogicPort port
</Location>
</IfModule>
```

Example:

With the following configuration, requests for the `/myapp1` URI received at the Oracle HTTP Server listen port will be forwarded to `/myapp1` on the Oracle WebLogic Server with the listen port `localhost:7001`

```
<IfModule weblogic_module>
<Location /myapp1>
WLSRequest On
WebLogicHost localhost
WeblogicPort 7001
</Location>
</IfModule>
```

To Forward Requests to a Cluster of Oracle WebLogic Server Instances

To forward requests to an application running on a **cluster of Oracle WebLogic Server instances**, specify the details of that destination cluster within a new `<location>` element.

Syntax:

```
<IfModule weblogic_module>
<Location path>
WLSRequest On
WebLogicCluster host:port,host:port,...
</Location>
</IfModule>
```

Example:

With the following configuration, requests for the `/myapp2` URI received at the Oracle HTTP Server listen port will be forwarded to `/myapp2` on the Oracle WebLogic Server cluster containing the Managed Servers with the listen ports `localhost:8002` and `localhost:8003`.

```
<IfModule weblogic_module>
<Location /myapp2>
WLSRequest On
WebLogicCluster localhost:8002,localhost:8003
</Location>
</IfModule>
```

To Configure Multiple Destinations

To configure multiple destinations—say, an application running on a single Oracle WebLogic Server instance and another application running on a cluster—you must specify each destination in a distinct `<location>` child element. All the `<location>` child elements should be at the same level within the `<IfModule weblogic_module>` element, as shown in the following syntax:

```
<IfModule weblogic_module>
#For an application running on a single server instance
<Location path1>
WLSRequest On
WebLogicHost host
WeblogicPort port
</Location>

#For an application running on a cluster
<Location path2>
WLSRequest On
WebLogicCluster host:port,host:port,...
</Location>

</IfModule>
```

To Link to Managed Servers

To configure the WLS OHS Plug-In so that it can link to Managed Servers, for example to enable a high availability deployment of Oracle HTTP Server, edit the `mod_wl_ohs.conf` file as follows:

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

<Location /weblogic>
  WLSRequest On
  WebLogicCluster apphost1.mycompany.com:7050,apphost2.com:7050
  DefaultFileName index.jsp
</Location>
```

> **Note:**
>
> If you are using SSL termination and routing requests to WebLogic Server, the following additional configuration is required.
>
> In the WebLogic console, **WebLogic Plugin Enabled** must be set to true, either at the domain, cluster or Managed Server level.
>
> In the Location block which directs requests to the WebLogic Managed Servers, one of the following lines also must be added.
>
> ```
> WLProxySSL ON
> WLProxySSLPassThrough ON
> ```
>
> (To help determine which parameter to use, see SSL Parameters for Oracle WebLogic Server Proxy Plug-Ins.)
>
> For example:
>
> ```
> <Location /weblogic>
>   WLSRequest On
>   WebLogicCluster apphost1.mycompany.com:7050,apphost2.com:7050
>   WLProxySSL On
>   WLProxySSLPassThrough ON
>   DefaultFileName index.jsp
> </Location>
> ```
>
> After enabling the WebLogic plugin, restart the Administration Server. See Terminating SSL Requests in *Administering Oracle HTTP Server*.

These examples show two different ways of routing requests to Oracle WebLogic 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 prefixed by `/weblogic` to the WebLogic Managed Server cluster located on Apphost1 and Apphost2.

To Configure One-way and Two-way SSL

For information about configuring the WLS OHS Plug-In to support one-way and two-way SSL between Oracle HTTP Server and Oracle WebLogic Server, see Using SSL with Proxy Plug-Ins.

Configuring IPv6 with Proxy Plug-Ins

The 14.1.2.0.0 WLS proxy plug-ins support IPv6. Specifically, the `WebLogicHost` and `WebLogicCluster` configuration parameters now support IPv6 addresses.

See WebLogicCluster and WebLogicHost.

For example:

```
<IfModule mod_weblogic.c>
 WebLogicHost [a:b:c:d:e:f]
 WebLogicPort 7002
 ...
</IfModule>
```

or

```
<IfModule mod_weblogic.c>
 WebLogicCluster [a:b:c:d:e:f]:<port>, [g:h:i:j:k:l]:<port>

 ....
</IfModule>
```

You can also use the IPv6 address mapped host name.

For example:

```
<IfModule mod_weblogic.c>
#hostname1 is mapped to IPv6 address in /etc/hosts file
 WebLogicHost hostname1
 WebLogicPort 7002
 ...
</IfModule>
```

Sample entry in the `/etc/hosts` file:

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
::1 hostname1
```

> **Note:**
>
> As of Windows 2008, the DNS server returns the IPv6 address in preference to the IPv4 address. If you are connecting to a Windows 2008 (or later) system using IPv4, the link-local IPv6 address format is tried first, which may result in a noticeable delay and reduced performance. To use the IPv4 address format, configure your system to instead use IP addresses in the configuration files or add the IPv4 addresses to the `etc/hosts` file.
>
> In addition, you may find that, setting the `DynamicServerList` property to `OFF` in the configuration file also improves performance with IPv6. When set to `OFF`, the proxy plug-in ignores the dynamic cluster list used for load balancing requests proxied from the proxy plug-in and uses the static list specified with the `WebLogicCluster` parameter.

Next Steps After Installing the 14.1.2.0.0 WLS OHS Plug-In

After installing the 14.1.2.0.0 WLS OHS Plug-In, to use its new features, complete its configuration as follows.

- Review the following directives, if configured, to enable TLSv1.3:
  - WebLogicSSLVersion
  - WebLogicSSLCiphers

This section includes the following topic:

- About HTTP Header Case Handling

About HTTP Header Case Handling

The WLS OHS Plug-In converts the case of HTTP headers based on the HTTP protocol version configured for the front-end and the back-end connections.

Table 2-1 shows how the case of request and response headers are modified based on the HTTP protocol version configured for the front-end and the back-end connections.

Table 2-1 Case of HTTP Request and Response Headers

HTTP Protocol Version for the Front-End Connection	HTTP Protocol Version for the Back-End Connection	Case of Request Header Sent to the Back End by the Oracle WebLogic Server Proxy Plug-In	Case of Response Header Sent to the Client by the Oracle WebLogic Server Proxy Plug-In
HTTP/1.1	HTTP/1.1	No Conversion	No Conversion

Understanding WLS OHS Plug-In Performance Metrics

Oracle HTTP Server provides performance metrics specific to the WLS OHS Plug-In (`mod_wl_ohs`) module, where a request is proxied to the back-end WebLogic Server.

These metrics are provided through the Oracle Dynamic Monitoring Service (DMS) which enables Oracle Fusion Middleware components to provide administration tools, such as Fusion Middleware Control, with data regarding the component's performance, state and on-going behavior. For the WLS OHS Plug-In module, for example, it could return the number of requests proxied, the number of failed requests, and other specific metrics. For more information on DMS, see Using the Oracle Dynamic Monitoring Service in *Tuning Performance Guide*.

> **Note:**
>
> The WLS OHS Plug-In module metrics are available only for Oracle HTTP Server and Apache Server plug-ins.

This section contains the following information on DMS metrics.

- Configuring DMS Metrics for the WLS OHS Plug-In
- Viewing Performance Metrics for the WLS OHS Plug-In
- DMS State Metrics
- DMS Event Metrics
- DMS PhaseEvent Metrics

Configuring DMS Metrics for the WLS OHS Plug-In

The DMS metrics for the WLS OHS Plug-In are enabled by default in the `admin.conf` file. They are included as part of the regular DMS metrics collection.

Viewing Performance Metrics for the WLS OHS Plug-In

You can view the performance metrics by using either the administration port, WLST commands, or Fusion Middleware Control. For details of each of the performance metrics, see DMS State Metrics, DMS Event Metrics, and DMS PhaseEvent Metrics.

Using the Administration Port:

If administration port is configured, for example, at `127.0.0.1:9999`, then you can view the raw DMS metrics at the URL `http://127.0.0.1/dms/`.

The metrics under the section `/WebLogicProxy [type=OHSWebLogic]` are the metrics coming from WLS OHS Plug-In.

Using WLST (Collocated Mode Only)

Use the WLST command `displayMetricTables` to view performance metrics, for example:

```
displayMetricTables(servertype="OHS", servers=<instancename>)
```

The metrics under the section `/WebLogicProxy [type=OHSWebLogic]` are the metrics coming from Oracle WebLogic Server Proxy Plug-in.

Using Fusion Middleware Control (Collocated Mode Only)

To view performance metrics in Fusion Middleware Control, select Oracle HTTP Server, then Monitoring, then Performance Summary. The metrics towards the bottom of this page will have WLS OHS Plug-In specific metrics. See Viewing Performance Metrics in *Administering Oracle HTTP Server*.

DMS State Metrics

A State metric tracks system status information or to track a metric that is not associated with an event. For a description of the State metrics, see Table 3-4.

Table 2-2 State Metrics for the WLS OHS Plug-In Module

Metric Name	Description
`totalDeclines`	The total number of requests declined (not processed by `mod_wl_24`). This number indicates the requests that are not configured, and/or rejected by the proxy plug-in (for example, custom HTTP methods are always rejected by the proxy plug-in)
`totalErrors`	Number of requests that could not be processed successfully. See Event Metrics for errors.
`totalHandled`	The total number of requests serviced by the `mod_wl_24` proxy plug-in.
`totalRequests`	The total number of requests received by `mod_wl_24`. The number includes all the requests that are targeted to the proxy plug-in, plus the requests that are not targeted to any module (not configured).
`totalRetries`	Number of times a request was retried. Requests are generally retried on failure (depending on configuration). If a request is ever retried, this metric will increment (once per request, irrespective of how many times the request was retried).
`totalSuccess`	The number of requests successfully processed. If the requests are processed successfully (proxied to Oracle WebLogic Server, and sent the response back to client), then this metric will be incremented.

Table 2-2 (Cont.) State Metrics for the WLS OHS Plug-In Module

Metric Name	Description
`websocketActive`	Number of WebSocket upgrade requests currently active.
`websocketClose`	Number of WebSocket upgrade requests closed. If the WebSocket session is terminated (for any reason), then this metric is updated.
`websocketMax`	Maximum number of simultaneous WebSocket requests that can be active. If the `WLMaxWebSocketClients` parameter is configured, the value will be the lower of these: • The configured value, OR • 0.75 of the value of `MaxRequestWorkers` (Apache 2.4) If `WLMaxWebSocketClients` parameter is not configured, the value will be 0.5 of the value of `MaxRequestWorkers` (Apache 2.4). For more information about the `WLMaxWebSocketClients` parameter, see Tuning Oracle HTTP Server for High Throughput for WebSocket Upgrade Requests.
`websocketPercent`	This value is defined by the number of active WebSockets (`websocketActive`) divided by the maximum number of simultaneous WebSocket requests (`websocketMax`) multiplied by 100: (`websocketActive/webocketMax`)`*100`.
`websocketRequests`	The number of WebSocket upgrade requests made. If the request URI is an WebSocket upgrade request, this metric will be incremented.
`websocketSuccess`	Number of WebSocket upgrade requests completed successfully. If Oracle WebLogic Server responds to a WebSocket upgrade request with `101 Switching Protocols`, then this metric is updated.

DMS Event Metrics

A DMS Event metric counts system events. A DMS event tracks system events that have a short duration, or where the duration of the event is not of interest but the occurrence of the event is of interest. For a description of the Event metrics, see Table 3-5.

Table 2-3 Event Metrics for the WLS OHS Plug-In.

Metric Name	Description
`errConnRefused`	The number of `CONNECTION_REFUSED` errors. Indicates the number of times the configured `WebLogicHost` and/or `WebLogicPort` is either not reachable or not listening.
`errNoResources`	The number of `NO_RESOURCES` errors. One scenario where this exception can occur is when SSL is configured in the proxy plug-in, but the corresponding SSL configuration is not defined in the managed server.
`errOthers`	The number of any other errors. For example, `POST` data size is greater than the value of `MaxPostSize`.
`errReadClient`	The number of `READ_ERROR_FROM_CLIENT` errors. Indicates the number of times that the proxy plug-in could not read from the client (browser).

Table 2-3 (Cont.) Event Metrics for the WLS OHS Plug-In.

Metric Name	Description
errReadServer	The number of READ_ERROR_FROM_SERVER errors. Indicates the number of times a read operation could not be successfully performed on Oracle WebLogic Server.
errReadTimeout	The number of READ_TIMEOUT errors. An example is Oracle WebLogic Server not responding within WLIOTimeoutSecs.
errWriteClient	The number of WRITE_ERROR_TO_CLIENT errors. Indicates the number of times that the proxy plug-in could not write to client. This can be seen when the client sends a request but closes the connection before receiving the response.
errWriteWLS	The number of WRITE_ERROR_TO_SERVER errors. Indicates the number of times that the proxy plug-in could not write to Oracle WebLogic Server.
wsClientClose	Number of WebSocket upgrade requests closed by client. If the client sends a WebSocket upgrade request, and client closes the connection, then this metric is updated.
wsErrorClose	Number of WebSocket sessions terminated due to error. If there is any error which causes the WebSocket connection to close, then this metric is updated.
wsNoUpgrade	The number of times the WebSocket upgrade request was rejected. The response to WebSocket upgrade request is not "101 Switching Protocols". This can happen when the upgrade request is sent to Oracle WebLogic Server that does not support WebSockets (Oracle WebLogic Server version 12.1.2 or earlier).
wsServerClose	Number of WebSocket upgrade requests closed by server. If Oracle WebLogic Server initiates a close of WebSocket communication, then this metric is updated. For example, timeout or no communication (by default, 5 minutes) after upgrading the request.

DMS PhaseEvent Metrics

A DMS PhaseEvent metric measures the time spent in a specific section of code that has a beginning and an end. A PhaseEvent tracks time in a method or in a block of code. For each phase event, an "active count", "completed count", "total time", "min time", "max time", and "average time" value is included. For a description of the PhaseEvent metrics, see Table 3-6.

Table 2-4 PhaseEvent Metrics for the WLS OHS Plug-In

Metric Name	Description
websocketPhase	WebSocket communication in progress. The phase (time) between "WebSocket upgrade succeeded" and "WebSocket connection closed"
wlsWait	The phase (time) between "the request sent to Oracle WebLogic Server" and "Waiting for response".

Deprecated Directives for Oracle HTTP Server

The WebLogic Server plug-in logs for the WLS OHS Plug-In are now part of the Web Server error log mechanism. References can be identified with module name as `weblogic`. For example:

```
[2015-05-14T00:43:27.8355-06:00] [OHS] [TRACE:16] [OH99999] [weblogic] [client_id: ::1]
[host_id: XXXXXXXX] [host_addr: XX.XXX.XXX.XXX] [pid: 1240] [tid: 2424] [user: sramavan]
[ecid: 00iT9hK4DrhFw0zobn063z0BvEE3zsYyk0000JO00000H] [rid: 0] [VirtualHost: main]
================New Request: [GET /favicon.ico HTTP/1.1] ================
```

The WLLogFile and Debug directives are deprecated. If the configuration uses these directives, the following note appears in the node manager plug-in log file (ohs_nm.log):

```
<2015-05-14 00:36:25> <INFO> <OHS-0> <[Thu May 14 00:36:25.723286 2015] [weblogic:warn]
[pid 5084:tid 668] The Debug directive is ignored. The web server log level is used
instead.>

<2015-05-14 00:36:25> <INFO> <OHS-0> <[Thu May 14 00:36:25.724263 2015] [weblogic:warn]
[pid 5084:tid 668] The WLLogFile directive is ignored. The web server log file is used
instead.>
```

To enable plug-in logs:

- If **OraLogMode** is set to *ODL-text,* set **OraLogSeverity** to TRACE:16. The logs appear in the directory **OraLogDir** (instance-name.log). This is the default.
- If **OraLogMode** is set to *apache*, set **LogLevel** to *debug*. The directive **ErrorLog** points to the file where the errors are logged.

See Managing Oracle HTTP Server Logs in *Administering Oracle HTTP Server* guide.

3

Installing and Configuring the Oracle WebLogic Server Proxy Plug-In for Apache HTTP Server

To install and configure the Oracle WebLogic Server Proxy Plug-In for Apache HTTP Server (WLS Apache Plug-In), Oracle recommends that you read the information included in this section.

This section includes the following topics:

- Installing the WLS Apache Plug-In
- Configuring the WLS Apache Plug-In
  Edit the `httpd.conf` file to proxy requests by path or by MIME type, to enable HTTP tunneling and to use the other WLS Apache Plug-In parameters.
- Enable and Configure HTTP/2 Support
  To leverage the benefits of the HTTP/2 protocol, HTTP/2 must be used for both front-end connections and back-end connections.
- Configuring IPv6 with Proxy Plug-Ins
  The 14.1.2.0.0 WLS proxy plug-ins support IPv6. Specifically, the `WebLogicHost` and `WebLogicCluster` configuration parameters now support IPv6 addresses.
- Understanding the DMS Metrics for the WLS Apache Plug-In
  The performance metrics for the WLS Apache Plug-In are provided through the Oracle Dynamic Monitoring Service (DMS). For example, it can fetch the number of requests proxied, the number of failed requests, and other specific metrics.
- Support and Patching
- Deprecated Directives for Apache HTTP Server

Installing the WLS Apache Plug-In

After you download the WLS Apache Plug-In, you can load it as a module in your Apache HTTP Server installation.
To download the WLS Apache Plug-In, see Availability of Oracle WebLogic Server Proxy Plug-Ins.

This section includes the following topics:

- Installation Prerequisites
  Review and ensure that you meet the necessary prerequisites.
- Installing the WLS Apache Plug-In
- Next Steps After Installing the WLS Apache Plug-In
  After installing the 14.1.2.0.0 WLS Apache Plug-In, to be able to use its new features, complete its configuration as follows.

Installation Prerequisites

Review and ensure that you meet the necessary prerequisites.

This section includes the following topics:

- Obtaining the WLS Apache Plug-In
- Java Requirements
- Apache HTTP Server Installation
- Oracle WebLogic Server Installation
- Setting the Environment Variables for the WLS Apache Plug-In
- Installing Microsoft Redistributable Package 2015-2022

Obtaining the WLS Apache Plug-In

To obtain the WLS Apache Plug-In:

1. Download the WLS Apache Plug-In, as described in Availability of Oracle WebLogic Server Proxy Plug-Ins.
2. Extract the proxy plug-in ZIP distribution to `PLUGINS_HOME`. For example, `/home/myhome/weblogic-plugins-14.1.2.0.0/`. This is the directory to which the proxy plug-in is extracted.

Table 3-1 lists the files included in the distribution.

Table 3-1 Files Included in the WLS Apache Plug-In Zip File for Linux

(Path)/File Name	Description
`README.txt`	The README file for the proxy plug-in.
`THIRD_PARTY_LICENSES.txt`	The file containing the third-party license related information.
`bin/orapki`	The `orapki` tool for configuring Oracle wallets.
`bin/export_wallet`	The executable file for exporting Oracle wallet to PEM formatted files.
`jlib/*.jar`	The helper Java libraries for `orapki` and the `export_wallet` program.
`lib/mod_wl_24.so`	The Oracle WebLogic Server Proxy Plug-In for Apache HTTP Server 2.4.
`lib/*.so`	The helper libraries.
`lib/nghttp2/*`	The HTTP/2 C library for `nghttp2`.

Java Requirements

Install the required version of Java (JDK). Table 3-2 lists the minimum JDK versions required for certain features when using the 14.1.2.0.0 proxy plug-in for Apache HTTP Server.

Table 3-2 Minimum JDK Requirements

Host on Which JDK Must be Installed	Feature that Requires JDK	Minimum JDK Version Required	Description
Machine on which the Apache HTTP Server is installed.	To use for managing Oracle wallet.	Oracle JDK 17 and 21	JDK is required for using the `orapki` tool (located at *`$PLUGINS_HOME`*`/bin`) to work with Oracle wallets configured for the web server. JDK is also required for `export_wallet` program (located at *`$PLUGINS_HOME`*`/bin`) that exports the content of the Oracle wallets to PEM formatted files on the file system.

Apache HTTP Server Installation

Ensure that you have a supported Apache HTTP Server installation. See Oracle Fusion Middleware Supported System Configurations.

Ensure that you are using Apache Portable Runtime 1.7.0 (apr-1.7.0). Add the path of apr-1.7.0 to the `LD_LIBRARY_PATH` using the following command:

```
export LD_LIBRARY_PATH=<absolute_path_to_the_directory_containing_apr-1.7.0>/
lib:${LD_LIBRARY_PATH}
```

> **Note:**
>
> The 14.1.2.0.0 WLS Apache Plug-In has been tested with Apache Portable Runtime 1.7.0 (apr-1.7.0). Therefore, it is recommended that you use this version of APR at a minimum. It is not known if using earlier versions of the APR with the proxy plug-in will result in the correct behavior.

You can download the APR from `https://apr.apache.org/`.

Oracle WebLogic Server Installation

Ensure that a supported version of Oracle WebLogic Server is configured and running on a target system. This server does not need to be running on the system on which you extracted the proxy plug-in ZIP distribution.

For the list of supported Oracle WebLogic Server versions, see https://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html.

For information about configuring Oracle WebLogic Server, see Planning the Oracle WebLogic Server Installation in *Installing and Configuring Oracle WebLogic Server and Coherence*.

Setting the Environment Variables for the WLS Apache Plug-In

> **Note:**
>
> Oracle recommends that you set the environment variables, such as *PLUGINS_HOME*, *JAVA_HOME*, and *LD_LIBRARY_PATH*.
> For example:
>
> ```
> PLUGINS_HOME=<absolute_path_to_the_directory_where_plugin_zip_is
> extracted_to>
> export PLUGINS_HOME
> ```
>
> You can use the variables set at the time of starting the Apache HTTP Server in the `httpd.conf` file by using the `${VAR_NAME}` syntax. For example:
>
> ```
> LoadModule weblogic_module ${PLUGINS_HOME}/lib/mod_wl_24.so
> ```

Set the following environment variables:

- Set `PLUGINS_HOME` to point to the directory where the proxy plug-ins ZIP file is extracted to, using the following command:

  ```
  export PLUGINS_HOME=<absolute_path_to_the_directory_where_plugin_zip_is
  extracted_to>
  ```

  For example:

  ```
  export PLUGINS_HOME=/home/myhome/weblogic-plugins-14.1.2.0.0/
  ```

- Set `JAVA_HOME` to point to the JDK present on the host where Apache HTTP Server is installed, using the following command:

  > **Note:**
  >
  > Relative path is not allowed for *JAVA_HOME*.

  ```
  export JAVA_HOME=<absolute_path_to_the_JDK_installation_directory>
  ```

  For example:

  ```
  export JAVA_HOME=/home/myhome/JDK_installation_path
  ```

> **Note:**
>
> JAVA_HOME is required only when implementing SSL for managing the Oracle wallet.
>
> For information about the supported JDK versions, see Table 3-2.

- Ensure that $PLUGINS_HOME/lib appears in the LD_LIBRARY_PATH on UNIX systems. To add $PLUGINS_HOME/lib to the LD_LIBRARY_PATH, use the command:

  ```
  export LD_LIBRARY_PATH=$PLUGINS_HOME/lib:$LD_LIBRARY_PATH
  ```

- Review the Third-Party Software Dependencies to determine if additional steps are needed to satisfy the dependency on the nghttp2 library.

Installing Microsoft Redistributable Package 2015-2022

You must install the Microsoft Redistributable Package 2015-2022 on Windows. For more information, see Microsoft Visual C++ Redistributable latest supported downloads.

> **Note:**
>
> If this package is not installed, the Apache Web Server startup fails on Windows after configuring the mod_wl_24.so module, with the following error:
>
> ```
> CANNOT LOAD MODULES/MOD_WL_24.SO
> ```
>
> .

Installing the WLS Apache Plug-In

The WLS Apache Plug-In is distributed as a shared object (.so) file. You can obtain the proxy plug-in from My Oracle Support (https://support.oracle.com/signin) or the Software Delivery Cloud.
To install the WLS Apache Plug-In:

1. Verify that the mod_so.c module is enabled.

   If you installed Apache HTTP Server using the script supplied by Apache, mod_so.c is already enabled. Verify that mod_so.c is enabled by running the following command:

   UNIX/Linux:

   ```
   APACHE_HOME/bin/apachectl -l
   ```

   (APACHE_HOME is the directory that contains the Apache HTTP Server installation.)

This command lists all enabled modules. If `mod_so.c` is not listed, you must rebuild your Apache HTTP Server, ensuring that the following configuration option is specified:

```
...
--enable-module=so
...
```

The output appears as follows:

```
# apachectl -l
Compiled in modules:
...
  mod_so.c
...
```

2. Make a copy of the *APACHE_HOME*`/conf/httpd.conf` file for backup.
3. Open the `httpd.conf` file.
4. Verify the syntax of the `httpd.conf` file by running the following command:

   UNIX/Linux:

   ```
   > APACHE_HOME/bin/apachectl -t
   ```

   If the `httpd.conf` file contains any errors, the output of this command shows the errors; ensure that you get a clean output before continuing. If there are no errors, the command returns the following:

   ```
   Syntax OK
   ```

   > **Note:**
   >
   > Ensure that you have resolved all the configuration errors from Steps 1 through 4 before contacting Oracle Support.

5. The WLS Apache Plug-In modules for Apache 2.4.x are shipped with the 14.1.2.0.0 proxy plug-in distributions. Apache 2.2.x is no longer supported. Therefore, the WLS Apache Plug-In module for Apache 2.2.x is no longer supported. Use the WLS Apache Plug-In module for Apache 2.4.x which continues to be supported.

   > **Note:**
   >
   > If you are using Apache 2.2.x version of the web server, migrate to the Apache 2.4.x version, and then install the Oracle WebLogic Server Proxy Plug-in module for Apache 2.4.x.

   Install the WLS Apache Plug-In module for Apache 2.4.x by adding the following line:

   ```
   LoadModule weblogic_module /home/myhome/weblogic-plugins-14.1.2.0.0/lib/mod_wl_24.so
   ```

6. After installing the WLS Apache Plug-In module, verify the syntax of the `httpd.conf` file by running the following command:

UNIX/Linux:

```
> APACHE_HOME/bin/apachectl -t
```

If the `httpd.conf` file contains any errors, the output of this command shows the errors. Contact Oracle Support for resolving the errors. If there are no errors, the command returns the following:

```
Syntax OK
```

Next Steps After Installing the WLS Apache Plug-In

After installing the 14.1.2.0.0 WLS Apache Plug-In, to be able to use its new features, complete its configuration as follows.

1. Review the minimum JDK requirements and install the supported version of JDK. See Installation Prerequisites.
2. Set the following environment variables:
   - `JAVA_HOME`
   - `PLUGINS_HOME`
   - `LD_LIBRARY_PATH`

   See Installation Prerequisites.

To use the new features of the 14.1.2.0.0 WLS Apache Plug-In, do the following:

- Configuring HTTP/2 Support on Back-End Connections.
- Review the following directives, if configured, to enable TLSv1.3:
  - WebLogicSSLVersion
  - WebLogicSSLCiphers

This section includes the following topics:

- Third-Party Software Dependencies
- About HTTP Header Case Handling
- Unsupported Use Cases

Third-Party Software Dependencies

The 14.1.2.0.0 WLS Apache Plug-In depends on the following third-party software:

- OpenSSL, version 1.1.1x or 3.0.x
- `nghttp2`, version 1.58.0

> **Note:**
>
> For Windows OpenSSL version, by default, the Windows environment latest Apache 2.4 builds come with OpenSSL version 3.x.

Libraries from the above third-party software must be available on the system where the Apache Web Server process (that loads the 14.1.2.0.0 WLS Apache Plug-In module) runs. These libraries are a prerequisite for the Apache Web Server process to start.

There are two flavors of the 14.1.2.0.0 WLS Apache Plug-In which are supported for both OpenSSL 1.1.1x and OpenSSL 3.0.x. For more information, see the Oracle Fusion Middleware Supported System Configurations page. When the Apache Web Server process loads the 14.1.2.0.0 WLS Apache Plug-In module, the OpenSSL libraries provided by the operating system are also loaded into the process address space.

About HTTP Header Case Handling

The WLS Apache Plug-In converts the case of HTTP headers based on the HTTP protocol version configured for the front-end and the back-end connections.

Table 2-1 shows how the case of request and response headers are modified based on the HTTP protocol version configured for the front-end and the back-end connections.

Table 3-3 Case of HTTP Request and Response Headers

HTTP Protocol Version for the Front-End Connection	HTTP Protocol Version for the Back-End Connection	Case of Request Header Sent to the Back End by the Oracle WebLogic Server Proxy Plug-In	Case of Response Header Sent to the Client by the Oracle WebLogic Server Proxy Plug-In
HTTP/2	HTTP/2	Lower Case	Lower Case
HTTP/1.1	HTTP/2	Lower Case	Camel Case
HTTP/2	HTTP/1.1	Camel Case	Lower Case
HTTP/1.1	HTTP/1.1	No Conversion	No Conversion

Unsupported Use Cases

The 14.1.2.0.0 WLS Apache Plug-In is loaded in an Apache Web Server process that loads open source modules such as `mod_ssl.so`, `mod_http2`, and so on, so that they depend on OpenSSL libraries. The 14.1.2.0.0 WLS Apache Plug-In also depends on the OpenSSL libraries. There are two flavors of the proxy plug-ins, which are supported for both OpenSSL 1.1.1 and OpenSSL 3.0.X.

To ensure the current functioning of the Apache Web Server process, Oracle recommends that you ensure that the versions of OpenSSL that different modules within an Apache Web Server process use are binary compatible to prevent symbol version conflicts.

The following use cases are not supported because these lead to an incorrect runtime behavior (process crash) that occurs due to symbol conflicts at runtime:

- Using the 14.1.2.0.0 WLS Apache Plug-In in an Apache Web Server process that uses open source Apache modules statically linked with a version of OpenSSL that is binary incompatible with OpenSSL version used by the 14.1.2.0.0 WLS Apache Plug-In.
- Using the 14.1.2.0.0 WLS Apache Plug-In in an Apache Web Server process that uses the open source Apache modules dynamically linked with a version of OpenSSL that is binary incompatible with OpenSSL version and the OpenSSL library does not support symbol versioning.

Configuring the WLS Apache Plug-In

Edit the `httpd.conf` file to proxy requests by path or by MIME type, to enable HTTP tunneling and to use the other WLS Apache Plug-In parameters.

This section includes the following topics:

Configuring the httpd.conf File

To configure the WLS Apache Plug-In, edit the `httpd.conf` file in your Apache HTTP Server installation. Complete the following tasks:

Task 1: Configure MIME Requests

You can proxy requests by MIME type and/or by path. Open the `httpd.conf` file in a text editor and complete the following steps:

> **Note:**
>
> If both MIME type and proxying by path are enabled, proxying by path takes precedence over proxying by MIME type.

Configuring Proxy Requests by MIME Type

To configure MIME requests by MIME type in the `httpd.conf` file, add a `MatchExpression` line to the `<IfModule>` block:

- **For a non-clustered Oracle WebLogic Server:** Define the `WebLogicHost` and `WebLogicPort` parameters with the `MatchExpression` directive.
  In the example below, a non-clustered Oracle WebLogic Server specifies that all files with MIME type `.jsp` are proxied:

```
<IfModule mod_weblogic.c>
 WebLogicHost my-weblogic.server.com
 WebLogicPort 7001
```

```
 MatchExpression *.jsp
</IfModule>
```

You can use multiple `MatchExpression` as well. For example:

```
<IfModule mod_weblogic.c>
 WebLogicHost my-weblogic.server.com
 WebLogicPort 7001
 MatchExpression *.jsp
 MatchExpression *.xyz
</IfModule>
```

- **For a cluster of Oracle WebLogic Servers:** Define the WebLogicCluster parameter with the `MatchExpression` directive.
  In the example below, a clustered Oracle WebLogic Server specifies that all files with MIME type `.jsp` are proxied:

```
<IfModule mod_weblogic.c>
 WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
 MatchExpression *.jsp
</IfModule>
```

See MatchExpression.

Configuring Proxy Requests by Path

To configure MIME requests by path in the `httpd.conf` file, configure the `PathTrim` parameter inside the `<Location>` tag. The `PathTrim` parameter specifies a string trimmed from the beginning of the URL before the request is passed to the Oracle WebLogic Server instance. See PathTrim.

For example, the following `Location` block proxies all requests that contain `/weblogic` in the URL:

```
<Location /weblogic>
  WLSRequest On
  PathTrim /weblogic
</Location>
```

The `<Location>` directive limits the scope of the enclosed directives by URL. See Apache Location Directive.

Task 2: Define Additional Parameters for the WLS Apache Plug-In

Define any additional parameters for the WLS Apache Plug-In.

The WLS Apache Plug-In recognizes the parameters listed in General Parameters for Oracle WebLogic Server Proxy Plug-Ins. To modify the behavior of your WLS Apache Plug-In, define these parameters either:

- In a `<Location>` block, for parameters that apply to proxying by path, or
- At global or virtual host scope, for parameters that apply to proxying by MIME type.

Task 3: Enable HTTP Tunneling (Optional)

You can enable HTTP tunneling for the T3 protocol by configuring the `<Location>` blocks.

To enable HTTP tunneling if you are using the T3 protocol and `weblogic.jar`, add the following `<Location>` block to the `httpd.conf` file:

```
<Location /bea_wls_internal>
 WLSRequest On
</Location>
```

Task 4: Enable Web Services Atomic Transaction (Optional)

You can enable Web Services Atomic Transaction (WS-AT) by configuring the `<Location>` blocks. The `<wls-wsat>` parameter applies to proxying by path. You can optionally define the parameter to modify the behavior of the Oracle WebLogic Server Proxy Plug-in for Apache HTTP Server.

```
<Location /wls-wsat>
 WLSRequest On
</Location>
```

WebLogic web services enable interoperability with other external transaction processing systems, such as IBM WebSphere, JBoss, Microsoft .NET. For more information about Web Services Atomic Transaction (WS-AtomicTransaction), see https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx.

Task 5: Verify and Apply Your Configuration

Follow these steps to verify the `httpd.conf` configuration and apply it to the Apache HTTP Server.

1. Verify the syntax of the `httpd.conf` file by running the following command (UNIX/Linux):

   ```
   > APACHE_HOME/bin/apachectl -t
   ```

   If the `httpd.conf` file contains any errors, the output of this command shows the errors; otherwise, the command returns the following:

   ```
   Syntax OK
   ```

2. Start the Apache HTTP Server (for UNIX/Linux):

   ```
   > APACHE_HOME/bin/apachectl start
   ```

3. Send a request to `http://apache-host:apache-port/mywebapp/my.jsp` from the browser. Validate the response.

Placing the WebLogic Properties Inside the Location or VirtualHost Blocks

If you choose to not use the `<IfModule>`, you can instead directly place the WebLogic properties inside the `Location` or `<VirtualHost>` blocks. Consider the following examples of the `<Location>` and `<VirtualHost>` blocks:

```
<Location /weblogic>
WLSRequest On
```

```
WebLogicHost myweblogic.server.com
WebLogicPort 7001
</Location>


<Location /weblogic>
WLSRequest On
WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
</Location>


<VirtualHost apachehost:80>
WLSRequest On
WebLogicServer weblogic.server.com
WebLogicPort 7001
</VirtualHost>
```

Default Apache Web Server and WLS Apache Plug-In HTTP Protocol Configuration

In the default Apache Web Server configuration, Apache Web Server supports HTTP/1.1 protocol (only) for both front-end connections and back-end connections. In this default Apache Web Server configuration:

- Clients will create HTTP/1.1 front-end connections to Apache Web Server:
  - Clients that support HTTP/1.1 only will create HTTP/1.1 front-end connections.
  - Clients that support HTTP/2 will negotiate HTTP/1.1 front-end connections.
- WLS Apache Plug-In will create HTTP/1.1 back-end connections to WebLogic Server:
  - If WebLogic Server supports HTTP/1.1 only, then Apache Web Server will create HTTP/1.1 back-end connections.
  - If WebLogic Server supports HTTP/2, then Apache Web Server will negotiate HTTP/1.1 back-end connections.

Example: Configuring the WLS Apache Plug-In

This example demonstrates basic instructions for quickly setting up the WLS Apache Plug-In to proxy requests to a back-end Oracle WebLogic Server.

1. Make a copy of `$APACHE_HOME/conf/httpd.conf` file.
2. Edit the file to add the following code:

```
...
LoadModule weblogic_module /home/myhome/weblogic-plugins-14.1.2.0.0/lib/mod_wl_24.so

<IfModule mod_weblogic.c>
  WebLogicHost wls-host
  WebLogicPort wls-port
</IfModule>

<Location /mywebapp>
  WLSRequest On
</Location>
...
```

3. Include `$PLUGINS_HOME/lib` in the `LD_LIBRARY_PATH`, using the following command:

```
$ export LD_LIBRARY_PATH=/home/myhome/weblogic-plugin-14.1.2.0.0/
lib:$LD_LIBRARY_PATH
```

> **Note:**
>
> You can also update the `LD_LIBRARY_PATH` by copying the 'lib' contents to *APACHE_HOME*`/lib` or by editing the *APACHE_HOME*`/bin/apachectl` to update the `LD_LIBRARY_PATH`.

4. Set `PLUGINS_HOME` to point to the directory where the proxy plug-ins zip file is extracted to, using the following command:

```
export PLUGINS_HOME=/home/myhome/weblogic-plugins-14.1.2.0.0/
```

5. Include the path containing the OpenSSL libraries in the `LD_LIBRARY_PATH`, using the following command:

```
export LD_LIBRARY_PATH=/home/myhome/openssl_installation/
lib:$LD_LIBRARY_PATH
```

6. At the prompt, start the Apache HTTP Server by entering:

```
$ ${APACHE_HOME}/bin/apachectl start
```

7. Send a request to `http://apache-host:apache-port/mywebapp/my.jsp` from the browser and validate the response

Including a weblogic.conf File in the httpd.conf File

To keep several separate configuration files, you can define parameters in a separate configuration file called `weblogic.conf`, by using the Apache HTTP Server `Include` directive in an `<IfModule>` block in the `httpd.conf` file.

```
<IfModule mod_weblogic.c>
 # Config file for Oracle WebLogic Server that defines the parameters
 Include conf/weblogic.conf
</IfModule>
```

The syntax of `weblogic.conf` files is the same as that for the `httpd.conf` file.

The following sections describe how to create the `weblogic.conf` files, and include the sample `weblogic.conf` files:

- Rules for Creating the weblogic.conf Files
- Sample weblogic.conf Configuration Files
- Template for the Apache HTTP Server httpd.conf File

Rules for Creating the weblogic.conf Files

Be aware of the following rules and best practices for constructing a `weblogic.conf` file.

- Enter each parameter on a new line. Do not put "=" between a parameter and its value. For example:

```
PARAM_1 value1
PARAM_2 value2
PARAM_3 value3
```

- If a request matches both a MIME type specified in a `MatchExpression` in an `<IfModule>` block and a path specified in a `Location` block, the behavior specified by the `<Location>` block takes precedence.
- If you use an Apache HTTP Server `<VirtualHost>` block, you must include all configuration parameters (`MatchExpression`, for example) for the virtual host within the `<VirtualHost>` block (see Apache Virtual Host documentation at `http://httpd.apache.org/docs/vhosts/`).
- You should use the `MatchExpression` statement instead of the `<Files>` block.

Here is sample of the `weblogic.conf` file:

Global configuration:

```
<IfModule mod_weblogic.c>
 WebLogicCluster johndoe02:8005,johndoe:8006
 WLTempDir "/tmp"
 DebugConfigInfo ON
 KeepAliveEnabled ON
 KeepAliveSecs 15
</IfModule>
```

Location configuration:

- All the requests that match `/jurl/*` will have the POST data files in `/tmp/jurl` and will reverse proxy the request to `myCluster` and port 7001.

```
<Location /jurl>
 WLSRequest On
 WebLogicCluster myCluster:7001
 WLTempDir "/tmp/jurl"
</Location>
```

- All the requests that match `/web/*` will have the POST data files in `/tmp/web` and will reverse proxy the request to myhost and port 8001.

```
<Location /web>
 WLSRequest On
 PathTrim /web
 WebLogicHost myhost
 WebLogicPort 8001
 WLTempDir "/tmp/web"
</Location>
```

- All the requests that match `/foo/*` will have the POST data files written to `/tmp/foo` and will reverse proxy the request to `myhost02` and port `8090`.

```
<Location /foo>
 WLSRequest On
 WebLogicHost myhost02
 WebLogicPort 8090
 WLTempDir "/tmp/foo"
 PathTrim /foo
</Location>
```

Sample weblogic.conf Configuration Files

These examples of `weblogic.conf` files may be used as templates that you can modify to suit your environment and server. Lines beginning with `#` are comments.

Example 3-1 Using WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks.
<IfModule mod_weblogic.c>
 WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
 ErrorPage http://myerrorpage.mydomain.com
 MatchExpression *.jsp
</IfModule>
#####################################################
```

In the example, the `MatchExpression` parameter syntax for expressing the filename pattern, the Oracle WebLogic Server host to which HTTP requests should be forwarded, and various other parameters are as follows:

```
MatchExpression [filename pattern] [WebLogicHost=host] | [paramName=value]
```

The first MatchExpression parameter below specifies the filename pattern `*.jsp`, and then names the single `WebLogicHost`. The `paramName=value` combinations following the pipe symbol specify the port at which Oracle WebLogic Server is listening for connection requests, and also activate the Debug option. The second `MatchExpression` specifies the filename pattern `*.html` and identifies the WebLogic Cluster hosts and their ports. The `paramName=value` combination following the pipe symbol specifies the error page for the cluster.

Example 3-2 Using Multiple WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks.
<IfModule mod_weblogic.c>
 MatchExpression *.jsp WebLogicHost=myHost|WebLogicPort=7001|Debug=ON
 MatchExpression *.html WebLogicCluster=myHost1:7282,myHost2:7283|ErrorPage=
 http://www.xyz.com/error.html
</IfModule>
```

Example 3-3 Without WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks.
<IfModule mod_weblogic.c>
 WebLogicHost myweblogic.server.com
 WebLogicPort 7001
 MatchExpression *.jsp
</IfModule>
```

Example 3-4 Configuring Multiple Name-Based Virtual Hosts

```
# VirtualHost1 = localhost:80
<VirtualHost 127.0.0.1:80>
DocumentRoot "/test/VirtualHost1"
ServerName localhost:80
<IfModule mod_weblogic.c>
#... WLS parameter ...
WebLogicCluster localhost:7101,localhost:7201
# Example: MatchExpression *.jsp <some additional parameter>
MatchExpression *.jsp PathPrepend=/test2
</IfModule>
</VirtualHost>

# VirtualHost2 = 127.0.0.2:80
<VirtualHost 127.0.0.2:80>
DocumentRoot "/test/VirtualHost1"
ServerName 127.0.0.2:80
<IfModule mod_weblogic.c>
#... WLS parameter ...
WebLogicCluster localhost:7101,localhost:7201
# Example: MatchExpression *.jsp <some additional parameter>
MatchExpression *.jsp PathPrepend=/test2
#... WLS parameter ...
</IfModule>
</VirtualHost>
```

You must define a unique value for `ServerName` or some proxy plug-in parameters will not work as expected.

Template for the Apache HTTP Server httpd.conf File

This section contains a sample `httpd.conf` file for Apache HTTP Server. You can use this sample as a template and modify it to suit your environment and server. Lines beginning with # are comments.

> **Note:**
>
> Apache HTTP Server is not case sensitive.

Sample httpd.conf file for Apache HTTP Server

```
##################################################
# APACHE-HOME/conf/httpd.conf file
##################################################
LoadModule weblogic_module /home/myhome/weblogic-plugins-12.2.1/lib/mod_wl_24.so

<Location /weblogic>
 WLSRequest On
 PathTrim /weblogic
 ErrorPage http://myerrorpage1.mydomain.com
</Location>

<Location /servletimages>
 WLSRequest On
 PathTrim /something
 ErrorPage http://myerrorpage1.mydomain.com
</Location>

<IfModule mod_weblogic.c>
  MatchExpression *.jsp
  WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
</IfModule>
```

About WebSocket Proxy Configurations

The 14.1.2.0.0 WLS Apache Plug-In for Apache HTTP Server 2.4.x can handle WebSocket connection upgrade requests and effectively proxy to WebSocket applications hosted within Oracle WebLogic Server 14c (14.1.1.0.0) and later.

Review following timeout setting for the WebSocket connection:

- If you use the `mod_reqtimeout` module within the Apache HTTP Server, then set the configured `client timeout` value appropriately to consider for the WebSocket connections.
- The default timeout value for HTTP requests in the `mod_reqtimeout` module has changed between Apache HTTP Server 2.2 and 2.4. This change can cause the WebSocket connections to break. Therefore, you will need to use an appropriate `client timeout` value.
- You should configure appropriate `client timeout` values for the WebSocket connections to avoid malicious attacks such as a Denial of Service attack.

> **Note:**
>
> WebSocket is not supported over HTTP/2.

Verifying the Log File

The Oracle WebLogic Server Proxy Plug-in logs are now part of the Apache HTTP Server error log. You can easily identify the references with the prefix `weblogic`:

```
[weblogic:debug] [pid 6571:tid 139894556022528] ApacheProxy.cpp(875): [client
10.184.61.77:53634] <657114316705052> =========New Request: [GET /weblogic/index.html
HTTP/1.1] ======
```

To enable the proxy plug-in logs, set the Apache web server directive `LogLevel` to `debug`. The logs are included in the file pointed to by the Apache web server `ErrorLog` directive.

```
config file name: httpd.conf
setting: LogLevel debug
```

Additionally, a new log file named `wl_exportwallet_log` is created in the same file system path where the web server's log file exists. In case of the Apache web server process, this file is located at `$SERVER_ROOT/logs/`.

The Oracle wallets used in the web server configuration must be exported to PEM formatted files on the file system to enable OpenSSL APIs to access the key and certificates present in the Oracle wallet. This is done by forking a separate process called `export_wallet` from the main web server process. The `export_wallet` process writes to the `wl_exportwallet_log`, and not to the web server's log.

Clustering Failover When Using the WLS Apache Plug-In

When using the WLS Apache Plug-In as the front-end for a cluster, the plug-in uses several configuration parameters to determine how long to wait for connections to the WebLogic Server host and, after a connection is established, how long the plug-in waits for a response:

- Verify the setting of the Apache `idempotent` flag. When `idempotent` is set to ON, and if the servers do not respond within the specified `WLIOTimeoutSecs` value, the plug-ins fail over. The plug-ins also fail over if `idempotent` is set to ON and the servers respond with an error such as `READ_ERROR_FROM_SERVER`. If set to OFF, the plug-ins do not fail over. See Parameters for Web Server Plug-Ins in *Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server*.
- Verify the setting of WebLogic Proxy Plug-in retry mechanism; for example, whether the maximum number of retries allowed is equal to the `ConnectTimeoutSecs` value divided by the `ConnectRetrySecs` value. See Failover, Cookies, and HTTP Sessions in *Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server*.

Enable and Configure HTTP/2 Support

To leverage the benefits of the HTTP/2 protocol, HTTP/2 must be used for both front-end connections and back-end connections.

To support the use of the HTTP/2 protocol for both front-end and back-end connections, all the following are required:

- Apache Web Server must be configured to support HTTP/2 front-end connections.
- WLS Apache Plug-In must be configured to support HTTP/2 back-end connections.
- You must use a version of WebLogic Server that supports HTTP/2, and WebLogic Server must be configured to support HTTP/2.

> **Note:**
>
> Both WebLogic Server 14.1.1 and WebLogic Server 14.1.2 support HTTP/2 and are configured, by default, to support HTTP/2.

In HTTP/2 configurations:

- Clients will create HTTP/2 front-end connections to Apache Web Server, if possible:
  - Clients that support HTTP/1.1 *only* will negotiate HTTP/1.1 front-end connections.
  - Clients that support HTTP/2 will create HTTP/2 front-end connections.
- WLS Apache Plug-In will create HTTP/2 back-end connections to WebLogic Server. HTTP/2 back-end connections will be used, regardless of whether HTTP/1.1 or HTTP/2 front-end connections are being used.

When used with clients that support HTTP/2, such configurations support “end to end” use of HTTP/2 – from clients to Apache Web Server to WebLogic Server. Such configurations will also support HTTP/1.1 front-end connections from clients that support HTTP/1.1 only - the use of HTTP/2 back-end connections is transparent to clients using HTTP/1.1 front-end connections.

The following sections describe how to enable and configure Apache Web Server and the WLS Apache Plug-In to support HTTP/2:

Enabling HTTP2 Support in the Apache Web Server

Use the `mod_http2` module to enable HTTP/2 for front-end connections. `mod_http2` uses the `nghttp2` library (`libnghttp2.so`) as its implementation base. For more information, see the Apache Web Server guide for the HTTP/2 implementation in Apache httpd.

Configuring HTTP/2 Support on Front-End Connections

Use the following steps to configure HTTP/2 support in the Apache Web Server:

1. Load the `http2_mod` module:

   ```
   LoadModule http2_module modules/mod_http2.so
   ```

2. The second directive you need to add to your server configuration is:

   ```
   Protocols h2 http/1.1
   ```

3. This allows h2, the secure variant, to be the preferred protocol on your server connections. When you want to enable all HTTP/2 variants, you write:

   ```
   Protocols h2 h2c http/1.1
   ```

Enabling HTTP2 Support in the WebLogic Apache Plug-In

WLS Apache Plug-In requires the `nghttp2` library (`libnghttp2.so`) for HTTP/2 support to back-end connections. For details, see Enabling HTTP2 Support in the Apache Web Server.

Configuring HTTP/2 Support on Back-End Connections

To configure HTTP/2 on back-end connections, you must set the `WLProtocol` directive in the `weblogic.conf` file.

Example With HTTP/2 Protocol Configured

```
Copy
LoadModule weblogic_module modules/mod_wl_24.so
Listen 4455
<VirtualHost *:4455>
  ServerName vh1.com
  WLSSLWallet /scratch/user/temp/server
  SecureProxy ON
  WLProtocol http/2
  <Location /myApp>
     WLSRequest On
     WebLogicCluster ns1.example.com:7011,ns2.example.com:7011,ns3.example.com:7011
  </Location>
  <Location /myApp2>
     WebLogicHost example.com
     WLSRequest On
     WebLogicPort 7025
  </Location>
</VirtualHost>
```

> **Note:**
>
> When the HTTP/2 protocol is configured, the following directives are ignored:
>
> - KeepAliveEnabled
> - KeepAliveSecs
> - WLMaxWebSocketClients

Server Push Functionality

The server push functionality is one of the key features of the HTTP/2 protocol.

Accessing websites follows the request and response pattern. A user sends a request to a remote server. The server responds with the requested content with some delay. The initial request to a web server is generally for an HTML document. In this scenario, the server replies with the requested HTML resource. The HTML is then parsed by the browser, where references to other assets are discovered, such as style sheets, scripts, and images. Upon their discovery, the browser makes separate requests for those assets, which are then responded to by the server. The problem with this mechanism is that it forces the user to wait for the browser to discover and retrieve critical assets until after an HTML document has been downloaded. This delays rendering time and increases load time.

The server push functionality is a solution to this problem. It lets the server preemptively push website assets to the client without the user having explicitly asked for them.

For example, consider a website where all pages rely on styles defined in an external style sheet named `styles.css`. When the user requests `index.html` from the server, the `styles.css` is sent to the user just after the server starts sending the response for `index.html`.

Rather than waiting for the server to send `index.html` and then waiting for the browser to request and receive `styles.css`, the user only has to wait for the server to respond with both `index.html` and `styles.css` on the initial request. This decreases the rendering time of the page.

To enable the server push functionality, configure the H2Push directive and use link headers either in the web server configuration file or in the response. For configuring the H2Push directive, see H2Push Directive in the Apache HTTP Server Documentation.

The WLS Apache Plug-In supports link headers created using the `Link.Builder` API.

Example of a link header in the web server configuration file:

```
H2Push on
<Location /xxx.html>
    Header add Link "</xxx.css>;rel=preload"
    Header add Link "</xxx.js>;rel=preload"
</Location>
```

Example of a link header in the response:

```
</xxx.css>;rel="preload";type="text/css"
```

To enable the server push functionality, set `H2Push` to `on` in the `httpd.conf` file if link header is sent in the response, as given below:

```
-----config section---
H2Push on
----------------------
```

> **Note:**
>
> `PushBuilder` from `HTTPServeletRequest`, and using 103 early hints along with the link headers are NOT supported.

- Enabling H2Push on Windows Apache
  On the Windows Apache Server, you must ensure that the `headers_module` is loaded which is required for the H2Push to function correctly.

Enabling H2Push on Windows Apache

On the Windows Apache Server, you must ensure that the `headers_module` is loaded which is required for the H2Push to function correctly.

Perform the following steps to enable H2Push:

1. Open the `httpd.conf` file located in your Apache installation. For example, `C:\Apache2\conf`.
2. Uncomment the following line in the `httpd.conf` file by removing `#`:

   ```
   #LoadModule headers_module modules/mod_headers.so
   ```

3. Save the `httpd.conf` file.

4. Restart the Apache Server.

Configuring IPv6 with Proxy Plug-Ins

The 14.1.2.0.0 WLS proxy plug-ins support IPv6. Specifically, the `WebLogicHost` and `WebLogicCluster` configuration parameters now support IPv6 addresses.

See WebLogicCluster and WebLogicHost.

For example:

```
<IfModule mod_weblogic.c>
 WebLogicHost [a:b:c:d:e:f]
 WebLogicPort 7002
 ...
</IfModule>
```

or

```
<IfModule mod_weblogic.c>
 WebLogicCluster [a:b:c:d:e:f]:<port>, [g:h:i:j:k:l]:<port>

 ....
</IfModule>
```

You can also use the IPv6 address mapped host name.

For example:

```
<IfModule mod_weblogic.c>
#hostname1 is mapped to IPv6 address in /etc/hosts file
 WebLogicHost hostname1
 WebLogicPort 7002
 ...
</IfModule>
```

Sample entry in the `/etc/hosts` file:

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
::1 hostname1
```

> **Note:**
>
> As of Windows 2008, the DNS server returns the IPv6 address in preference to the IPv4 address. If you are connecting to a Windows 2008 (or later) system using IPv4, the link-local IPv6 address format is tried first, which may result in a noticeable delay and reduced performance. To use the IPv4 address format, configure your system to instead use IP addresses in the configuration files or add the IPv4 addresses to the `etc/hosts` file.
>
> In addition, you may find that, setting the `DynamicServerList` property to `OFF` in the configuration file also improves performance with IPv6. When set to `OFF`, the proxy plug-in ignores the dynamic cluster list used for load balancing requests proxied from the proxy plug-in and uses the static list specified with the `WebLogicCluster` parameter.

Understanding the DMS Metrics for the WLS Apache Plug-In

The performance metrics for the WLS Apache Plug-In are provided through the Oracle Dynamic Monitoring Service (DMS). For example, it can fetch the number of requests proxied, the number of failed requests, and other specific metrics.

You can configure and view the DMS performance metrics for Oracle WebLogic Server 14.1.2.0.0 Proxy Plug-in.

The DMS metrics that can be returned are described in DMS State Metrics, DMS Event Metrics, and DMS PhaseEvent Metrics.

This section includes the following topics:

Configuring the DMS Metrics for the WLS Apache Plug-In

To configure the DMS metrics for the WLS Apache Plug-In, add the following code to the `httpd.conf` file:

```
# Add the following LoadModule only if it is not already present
# Use mod_wl_24.so for Apache 2.4
LoadModule weblogic_module $PLUGINS_HOME/mod_wl_24.so

<Location /metrics>
    SetHandler dms-handler
</Location>
```

Viewing the Performance Metrics for the WLS Apache Plug-In

You can view the raw metrics using the following URL:

`http://apachehost:apacheport/metrics`

Where, *apachehost* is the host name of the Apache server and *apacheport* is the port number.

The metrics that are coming from the WLS Apache Plug-In can be found under the `/WebLogicProxy [type=WebLogicProxy]` section.

DMS State Metrics

A State metric tracks system status information or to track a metric that is not associated with an event. For a description of the State metrics, see Table 3-4.

Table 3-4 State Metrics for the WLS OHS Plug-In Module

Metric Name	Description
`totalDeclines`	The total number of requests declined (not processed by `mod_wl_24`). This number indicates the requests that are not configured, and/or rejected by the proxy plug-in (for example, custom HTTP methods are always rejected by the proxy plug-in)
`totalErrors`	Number of requests that could not be processed successfully. See Event Metrics for errors.
`totalHandled`	The total number of requests serviced by the `mod_wl_24` proxy plug-in.
`totalRequests`	The total number of requests received by `mod_wl_24`. The number includes all the requests that are targeted to the proxy plug-in, plus the requests that are not targeted to any module (not configured).
`totalRetries`	Number of times a request was retried. Requests are generally retried on failure (depending on configuration). If a request is ever retried, this metric will increment (once per request, irrespective of how many times the request was retried).
`totalSuccess`	The number of requests successfully processed. If the requests are processed successfully (proxied to Oracle WebLogic Server, and sent the response back to client), then this metric will be incremented.
`websocketActive`	Number of WebSocket upgrade requests currently active.
`websocketClose`	Number of WebSocket upgrade requests closed. If the WebSocket session is terminated (for any reason), then this metric is updated.
`websocketMax`	Maximum number of simultaneous WebSocket requests that can be active. If the `WLMaxWebSocketClients` parameter is configured, the value will be the lower of these: • The configured value, OR • 0.75 of the value of `MaxRequestWorkers` (Apache 2.4) If `WLMaxWebSocketClients` parameter is not configured, the value will be 0.5 of the value of `MaxRequestWorkers` (Apache 2.4). For more information about the `WLMaxWebSocketClients` parameter, see Tuning Oracle HTTP Server for High Throughput for WebSocket Upgrade Requests.
`websocketPercent`	This value is defined by the number of active WebSockets (`websocketActive`) divided by the maximum number of simultaneous WebSocket requests (`websocketMax`) multiplied by 100: (`websocketActive/webocketMax`)`*100`.

Table 3-4 (Cont.) State Metrics for the WLS OHS Plug-In Module

Metric Name	Description
websocketRequests	The number of WebSocket upgrade requests made. If the request URI is an WebSocket upgrade request, this metric will be incremented.
websocketSuccess	Number of WebSocket upgrade requests completed successfully. If Oracle WebLogic Server responds to a WebSocket upgrade request with 101 Switching Protocols, then this metric is updated.

DMS Event Metrics

A DMS Event metric counts system events. A DMS event tracks system events that have a short duration, or where the duration of the event is not of interest but the occurrence of the event is of interest. For a description of the Event metrics, see Table 3-5.

Table 3-5 Event Metrics for the WLS OHS Plug-In.

Metric Name	Description
errConnRefused	The number of CONNECTION_REFUSED errors. Indicates the number of times the configured WebLogicHost and/or WebLogicPort is either not reachable or not listening.
errNoResources	The number of NO_RESOURCES errors. One scenario where this exception can occur is when SSL is configured in the proxy plug-in, but the corresponding SSL configuration is not defined in the managed server.
errOthers	The number of any other errors. For example, POST data size is greater than the value of MaxPostSize.
errReadClient	The number of READ_ERROR_FROM_CLIENT errors. Indicates the number of times that the proxy plug-in could not read from the client (browser).
errReadServer	The number of READ_ERROR_FROM_SERVER errors. Indicates the number of times a read operation could not be successfully performed on Oracle WebLogic Server.
errReadTimeout	The number of READ_TIMEOUT errors. An example is Oracle WebLogic Server not responding within WLIOTimeoutSecs.
errWriteClient	The number of WRITE_ERROR_TO_CLIENT errors. Indicates the number of times that the proxy plug-in could not write to client. This can be seen when the client sends a request but closes the connection before receiving the response.
errWriteWLS	The number of WRITE_ERROR_TO_SERVER errors. Indicates the number of times that the proxy plug-in could not write to Oracle WebLogic Server.
wsClientClose	Number of WebSocket upgrade requests closed by client. If the client sends a WebSocket upgrade request, and client closes the connection, then this metric is updated.
wsErrorClose	Number of WebSocket sessions terminated due to error. If there is any error which causes the WebSocket connection to close, then this metric is updated.

Table 3-5 (Cont.) Event Metrics for the WLS OHS Plug-In.

Metric Name	Description
`wsNoUpgrade`	The number of times the WebSocket upgrade request was rejected. The response to WebSocket upgrade request is not "`101 Switching Protocols`". This can happen when the upgrade request is sent to Oracle WebLogic Server that does not support WebSockets (Oracle WebLogic Server version 12.1.2 or earlier).
`wsServerClose`	Number of WebSocket upgrade requests closed by server. If Oracle WebLogic Server initiates a close of WebSocket communication, then this metric is updated. For example, timeout or no communication (by default, 5 minutes) after upgrading the request.

DMS PhaseEvent Metrics

A DMS PhaseEvent metric measures the time spent in a specific section of code that has a beginning and an end. A PhaseEvent tracks time in a method or in a block of code. For each phase event, an "active count", "completed count", "total time", "min time", "max time", and "average time" value is included. For a description of the PhaseEvent metrics, see Table 3-6.

Table 3-6 PhaseEvent Metrics for the WLS OHS Plug-In

Metric Name	Description
`websocketPhase`	WebSocket communication in progress. The phase (time) between "WebSocket upgrade succeeded" and "WebSocket connection closed"
`wlsWait`	The phase (time) between "the request sent to Oracle WebLogic Server" and "Waiting for response".

Support and Patching

When you encounter issues with a proxy plug-in, always report the version of the proxy plug-in you are using. You can find this information in the Apache log.

The version information looks like the following snippet:

```
WebLogic Server Plugin version 14.1.2.0.0 <WLSPLUGINS_XXXX_XXXX_XXXXX.XXXX>
```

> **Note:**
>
> On the Apache Web Server for Linux, you can also obtain the proxy plug-in version by issuing the following command:
>
> ```
> $ strings ${PLUGINS_HOME}/lib/mod_wl_24.so | grep -i wlsplugins
> ```

A patch for a proxy plug-in will typically contain one or more shared objects to be replaced. Ensure to backup your original files as you replace them with those in the patch. Validate that the patch has been correctly updated by checking the version string in the logs.

You can obtain the latest updates for security fixes from the Critical Patch Update (CPU) Patch Advisor for Oracle Fusion Middleware (Doc ID 2806740.2).

Deprecated Directives for Apache HTTP Server

The `WLLogFile` and `Debug` directives are deprecated. If the configuration uses these directives, a note appears during startup.

```
[Thu May 14 23:22:19 2015] [warn] weblogic: The Debug directive is ignored.  The web
server log level is used instead.
```

For information about log files, see Verifying the Log File.

4

Configuring Security

This chapter describes how to work with security for proxy plug-ins.
This chapter includes the following topics:

- Using SSL with Proxy Plug-Ins
  You can use the Transport Layer Security (TLS) or Secure Sockets Layer (SSL) protocols to protect the connection between the proxy plug-in and Oracle WebLogic Server. The TLS/SSL protocol provides confidentiality and integrity to the data passed between the proxy plug-in and Oracle WebLogic Server.
- Configuring Perimeter Authentication
  Use perimeter authentication to secure Oracle WebLogic Server applications that are accessed by using the proxy plug-in.
- About Federal Information Processing Standards
  Federal Information Processing Standards (FIPS) is not supported in the 14.1.2.0.0 WLS proxy plug-ins.

Using SSL with Proxy Plug-Ins

You can use the Transport Layer Security (TLS) or Secure Sockets Layer (SSL) protocols to protect the connection between the proxy plug-in and Oracle WebLogic Server. The TLS/SSL protocol provides confidentiality and integrity to the data passed between the proxy plug-in and Oracle WebLogic Server.

The proxy plug-in does not use the transport protocol (HTTP or HTTPS) specified in the HTTP request (usually by the browser) to determine whether to use TLS/SSL to protect the connection between the proxy plug-in and Oracle WebLogic Server; that is, the proxy plug-in is in no way dependent on whether the HTTP request (again, usually from the browser) uses HTTPS (TLS/SSL).

Instead, the proxy plug-in uses TLS/SSL parameters that you configure for the proxy plug-in, as described in SSL Parameters for Oracle WebLogic Server Proxy Plug-Ins, to determine when to use SSL:

- `WebLogicSSLVersion` - Specifies the TLS/SSL protocol version to use for communication between the proxy plug-in and Oracle WebLogic Server.
- `WLSSLWallet` - The Oracle WebLogic Server 14.1.2.0.0 Proxy Plug-ins use Oracle wallets to store SSL configuration information.
  - Use the `WLSSLWallet` TLS/SSL configuration parameter to configure the wallets. The `orapki` utility is provided in the proxy plug-in distribution for this purpose.
  - The orapki utility manages public key infrastructure (PKI) elements, such as wallets and certificate revocation lists, on the command line so the tasks it performs can be incorporated into scripts. This enables you to automate many of the routine tasks of maintaining a PKI. See Using the orapki Utility for Certificate Validation and CRL Management.
- `SecureProxy` - The `SecureProxy` parameter determines whether SSL is enabled.

> **Note:**
>
> For information about configuring earlier versions of SSL/TLS on the Oracle WebLogic Server side, see Using the weblogic.security.SSL.protocolVersion System Property in *Administering Security for Oracle WebLogic Server*.

In the case of two-way TLS/SSL, the proxy plug-in (the TLS/SSL client) automatically uses two-way TLS/SSL when Oracle WebLogic Server is configured for two-way TLS/SSL and requests a client certificate. For more information about configuring TLS/SSL in Oracle WebLogic Server, see Set Up TLS in *Oracle WebLogic Remote Console Online Help*.

If a client certificate is not requested, the proxy plug-ins default to one-way SSL.

> **Note:**
>
> If an Oracle WebLogic Server 14.1.2.0.0 product is installed on the same system as the WLS OHS Plug-In, the `ORACLE_HOME` variable must point to a valid installation; otherwise, the proxy plug-in fails to initialize SSL.
>
> For example, if `ORACLE_HOME` is invalid because the product was not cleanly removed, the proxy plug-in fails to initialize TLS/SSL.

This section includes the following topics:

Configuring Libraries for SSL

WLS proxy plug-ins have been rewritten to use OpenSSL libraries and removed the dependency on Oracle's NZ libraries. Therefore, some minor changes are necessary for the existing deployments to use SSL for communication with Oracle WebLogic Server as described in Configuring Environment Variables.

> **Note:**
>
> The current implementation of the WLS OHS Plug-In does not support the use of multiple certificate files.

Configuring Environment Variables

The WLS proxy plug-ins use an Oracle Wallet to store SSL information such as private key, user certificate chain, and the list of trusted certificates. OpenSSL APIs cannot read the content of such wallets. Therefore, it is necessary to export the content of a user-supplied wallet to a format that OpenSSL APIs can read. The content of the wallet will be exported as PEM formatted files on the file system. For each user-supplied wallet, three files may be created in the same file system path where the auto-login wallet is present:

- `key.pem`: A pass-phrase protected file containing the private key in PEM format, if a private key is present in the wallet.
- `user.crt`: User certificate in PEM format, if a user certificate is present in the wallet.
- `trusted_certs.crt`: Chain of trusted certificates in PEM format.

Configuring a Proxy Plug-In for One-Way SSL

Perform the following steps to configure one-way SSL.

In these steps, you run the keytool commands on the system on which Oracle WebLogic Server is installed, and you run the `orapki` commands on the system on which the 14.1.2.0.0 WLS proxy plug-ins are installed.

> **Note:**
>
> The examples in this section use the Oracle WebLogic Server demo CA. If you are using the proxy plug-in a production environment, ensure that trusted CAs are properly configured for the proxy plug-in and for Oracle WebLogic Server.

1. Configure Oracle WebLogic Server for SSL. See Configuring SSL in *Administering Security for Oracle WebLogic Server*.
2. Create an Oracle Wallet, by using the `orapki` utility.

   ```
   orapki wallet create -wallet mywallet -auto_login_only
   ```

   See Using the orapki Utility for Certificate Validation and CRL Management in the *Administering Oracle Fusion Middleware*.

> **Note:**
>
> Only the user who creates the wallet (or for Windows, the account SYSTEM) has access to the wallet.
>
> This is typically sufficient for the WLS Apache Plug-In because Apache HTTP Server runs as the account SYSTEM on Windows, and as the user who creates it on UNIX.
>
> If the user who runs the WLS Apache Plug-In is different from the user who creates the wallet (or for Windows, the account `SYSTEM`), you need to grant the user access to the wallet by running the command `cacls` (Windows) or `chmod` (UNIX) after you create the wallet. For example:
>
> ```
> cacls <wallet_path>\cwallet.sso /e /g IUSR:R
> ```

3. Import the Oracle WebLogic Server trust certificate into the Oracle Wallet.

   ```
   orapki wallet add -wallet mywallet -trusted_cert -cert <cert_file_name> -
   auto_login_only
   ```

4. Complete these steps if the version of the Oracle WebLogic Server instances in the back end.

   a. Open the WebLogic Remote Console.

   b. In the **Edit Tree**, go to **Environment**, then:

      - If the server instances to which you want to proxy requests from Apache HTTP Server or Oracle HTTP Server are in a cluster, select **Clusters**.
      - Otherwise, select **Servers**.

   c. Select the server or cluster to which you want to proxy requests from Apache HTTP Server or Oracle HTTP Server.

   d. Click **Show Advanced Fields**.

   e. Do one of the following:

To...	Turn on...
Enable one-way SSL	**WebLogic Plug-In Enabled**
Enable two-way SSL where client certificates are used to authenticate	**Client Cert Proxy Enabled**
Enable two-way SSL with client certificates.	Both

      If you selected **Servers** (and not Clusters), repeat this step for every server to which you want to proxy requests from Apache HTTP Server or Oracle HTTP Server.

   f. Click **Save**.

   For the change to take effect, you must restart the server instances.

5. Send a request to `http://host:port/mywebapp/my.jsp` from the browser and validate the response.

Configuring a Two-Way SSL Between the Proxy Plug-In and Oracle WebLogic Server

When Oracle WebLogic Server is configured for a two-way SSL, the proxy plug-in forwards the user certificate to Oracle WebLogic Server. A two-way SSL can be established as long as Oracle WebLogic Server can validate the user certificate.

In these steps, you run the `keytool` commands on the system on which Oracle WebLogic Server is installed. You run the `orapki` commands on the system on which the 14.1.2.0.0 proxy plug-ins are installed.

To configure a two-way SSL:

1. Perform the steps described in Configuring a Proxy Plug-In for One-Way SSL.
2. Set the Oracle WebLogic Server SSL configuration options that require the presentation of client certificates (for two-way SSL).
3. From the Oracle wallet, generate a certificate request:
   a. Add the certificate request to the Oracle wallet.

      ```
      orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|1024|
      2048
      ```

   b. Export the certificate request from Oracle wallet:

      ```
      orapki wallet export -wallet wallet_location -dn certificate_request_dn
      -request certificate_request_filename
      ```

      See Exporting Certificates and Certificate Requests from Oracle Wallets with orapki in *Administering Oracle Fusion Middleware*.
4. Use the certificate request exported in Step 3 to create a certificate by using a certificate authority (CA) or some other mechanism.
5. Import all trusted certificates in the certificate chain of a user certificate before adding a user certificate. The certificate chain includes the intermediate certificate authorities and the root certificate authority.

   Import the root CA certificate as a trusted certificate by using the following command:

   ```
   orapki wallet add -wallet wallet_location -trusted_cert -cert
   root_certificate_location
   ```

   Import the intermediate CA certificate as a trusted certificate. If there are more than one intermediate CA certificate, execute the following command by changing the location for the `-cert` option for each intermediate CA.

   ```
   orapki wallet add -wallet wallet_location -trusted_cert -cert
   intermediate_certificate_location
   ```

   See Adding Certificates and Certificate Requests to Oracle Wallets with orapki in the *Database Security Guide*.

6. Import the certificate signed by CA as a user certificate to Oracle wallet.

```
orapki wallet add -wallet wallet_location -user_cert -cert
certificate_location
```

7. Import the CA certificate as a trusted certificate in the WebLogic trust store. Oracle WebLogic Server needs to trust the certificate.

```
keytool -file certificate_location -importcert -trustcacerts -keystore
DemoTrust.jks -storepass <passphrase>
```

Replacing Certificates Signed Using the MD5 Algorithm

When using SSL to connect to Oracle WebLogic Server, ensure that any certificate request or certificates signed with MD5 are replaced by SHA-2 signed certificates in the wallet; otherwise, the server fails to start.

This section includes the following topics:

- Checking the Certificate Signing Algorithm
- Creating a New Wallet to Add Certificates Signed with the SHA-2 Algorithm
- Replacing the Existing Certificates with SHA-2 Signed Certificates

Checking the Certificate Signing Algorithm

To check the certificate signing algorithm:

1. Use the `orapki` command to obtain the Distinguished Name (DN) for an SSL certificate.

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet <wallet__location>
```

For example:

- Content of the wallet with a CA-signed user certificate:

```
${PLUGINS_HOME}/bin/orapki  wallet display -wallet /tmp/test_wallet
```

Sample output:

```
Oracle PKI Tool: Version 23.0.0.0.0
Copyright (c) 2004, 2024, Oracle and/or its affiliates. All rights reserved.

Requested Certificates:
User Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
Trusted Certificates:
Subject: CN=im_ca,OU=O,O=FOR TESTING ONLY
Subject: CN=root_ca,OU=O,O=FOR TESTING ONLY
```

In this example, the user certificate is siged with an intermediate CA. Hence, you see a complete trust chain in the trusted certificate.

  – The Distinguished Name for user certificates is "**CN=localhost,O=FOR TESTING ONLY**"

  - The Distinguished Name for intermediate certificates is "**CN=im_ca,OU=O,O=FOR TESTING ONLY**"
  - The Distinguished Name for root certificates is: "**CN=root_ca,OU=O,O=FOR TESTING ONLY**"

- Content of the wallet with a self-signed user certificate:

```
${PLUGINS_HOME}/bin/orapki  wallet display -wallet /tmp/test_wallet
```

Sample output:

```
Oracle PKI Tool : Version 23.0.0.0.0
Copyright (c) 2004, 2024, Oracle and/or its affiliates. All rights reserved.

Requested Certificates:
User Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
Trusted Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
```

The Distinguished Name for the self-singed user certificates is "**CN=localhost,O=FOR TESTING ONLY**"

2. Export the certificates present in the wallet.

```
${PLUGINS_HOME}/bin/orapki  wallet export -wallet <wallet_Location> -dn
'DN_string' -cert <certificate_file>
```

For example:

- Export the user certificate.

```
${PLUGINS_HOME}/bin/orapki  wallet export -wallet <wallet_Location> -dn
'CN=localhost,O=FOR TESTING ONLY' -cert user.crt
```

For more information about this step, see orapki wallet export in the *Database Security Guide*.

- Export the intermediate and root CA certificates.

```
${PLUGINS_HOME}/bin/orapki  wallet export -wallet <wallet_Location> -dn
'CN=im_ca,OU=O,O=FOR TESTING ONLY' -cert im_ca.crt
```

```
${PLUGINS_HOME}/bin/orapki  wallet export -wallet <wallet_Location> -dn
'CN=root_ca,OU=O,O=FOR TESTING ONLY' -cert root_ca.crt
```

3. Check the signature algorithm used to sign `<certificate_file>` using the keytool:

```
$JAVA_HOME/bin/keytool -printcert -file <certificate_file>
```

For example, if the certificate is signed with MD5, the `Signature algorithm name` is set to `MD5withRSA`, as shown in the following sample command output:

```
$JAVA_HOME/bin/keytoolkey -printcert -file  user.crt
```

Sample output:

```
Owner: CN=localhost,OU=O,O=FOR TESTING ONLY
Issuer: CN=localhost,OU=O,O=FOR TESTING ONLY
Serial number: ---
Valid from: ---
Certificate fingerprints:
  MD5:  ---
  SHA1: ---
  SHA256: ---
  Signature algorithm name: MD5withRSA
  Version: 1
```

> **Note:**
>
> If any of the user and trusted certificates in the chain are signed with the MD5 algorithm, you can either create a new wallet with new certificates signed with the SHA-2 algorithm or replace the existing certificates with certificates signed with the SHA-2 signed algorithm.

The list of parameters used in the `orapki` commands:

Table 4-1 Command Parameters

Parameter	Description
`-wallet`	Specifies the wallet location.
`-dn`	Specifies the distinguished name of the certificate.
`-cert`	Specifies the directory location where the tool places the exported certificate.

Creating a New Wallet to Add Certificates Signed with the SHA-2 Algorithm

To create a new wallet:

1. Create a wallet.

```
${PLUGINS_HOME}/bin/orapki  orapki wallet create -wallet
<wallet_location> -auto_login_only
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet create -wallet test_wallet -
auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

Check the content of `test_wallet`:

```
$ls test_wallet
```

```
cwallet.sso
```

For more information about creating wallets with `orapki`, see Creating and Viewing Oracle Wallets with orapki in *Administering Oracle Fusion Middleware*.

2. Add the user certificate to the wallet. User certificates can be self-signed or CA-signed. For production, Oracle recommends to use a CA-signed certificate.

   a. Add a self-signed user certificate.

      i. Run the following command:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet <wallet_Location> -dn
'DN_string' -keysize 512|1024|2048|4096|8192|16384 -sign_alg sha256
-self_signed -validity 9125 [-pwd <pwd>] | [-auto_login_only]
```

      For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -dn
'CN=localhost,O=FOR TESTING ONLY' -keysize 2048  -sign_alg sha256 -
self_signed -validity 9125 -auto_login_only
```

      Sample output:

```
Operation is successfully completed.
```

      ii. List the content of the wallet after adding self-signed certificate to the wallet:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet <wallet_location>
```

      For example:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

      Sample output:

```
Requested Certificates:
User Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
Trusted Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
```

      For more information about adding certificates to a wallet, see Adding a Root Certificate to an Oracle Wallet in *Administering Oracle Fusion Middleware*.

   b. Add a CA-signed user certificate.

      i. Add the certificate request to the Oracle wallet.

```
orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|
1024|2048 [-pwd <pwd>] | [-auto_login_only]
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -dn
"CN=localhost,O=testing_only" -keysize 2048 -auto_login_only
```

Sample output:

```
Operation is successfully completed

Wallet content after adding certificate request
${PLUGINS_HOME}/bin/orapki wallet display -wallet /scratch/shichoud/
test_wallet

Requested Certificates:
Subject:        CN=localhost,O=testing_only
User Certificates:
Trusted Certificates:
...
```

ii. Export the certificate request from Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn
certificate_request_dn -request certificate_request_filename
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet export -wallet test_wallet -dn
"CN=localhost,O=testing_only" -request user.csr
```

Sample output:

```
Operation is successfully completed.
```

To view the content of the certificate, run the following command:

```
cat user.csr
```

Sample output:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICcDCCAVgCAQAwKzEVMBMGA1UECgwMdGVzdGluZ19vbmx5MRIwEAYDVQQDEwls
...
...
WnDd1cweMAH+1/D1C4Gi7Gvhi2Axw18H60lmZcU3JXv2bhu8QxZI9N6sI1DjU2Mg
l6EH2w==
```

See Exporting Certificates and Certificate Requests from Oracle Wallets with orapki in *Administering Oracle Fusion Middleware*.

iii. Use the certificate request exported in Step 3 to create a certificate by using a certificate authority (CA) or some other mechanism.

iv. Import all the trusted certificates in the certificate chain of a user certificate before adding a user certificate. The certificate chain includes the intermediate certificate authorities and the root certificate authority.

- Import the root CA certificate as a trusted certificate by using the following command:

  ```
  orapki wallet add -wallet wallet_location -trusted_cert -cert
  root_certificate_location [-pwd <pwd>] | [-auto_login_only]
  ```

  For example:

  ```
  ${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
  trusted_cert -cert Root_CA.crt -auto_login_only
  ```

  Sample output:

  ```
  Operation is successfully completed.
  ```

- Import the intermediate CA certificate as a trusted certificate. If there are more than one intermediate CA certificate, execute the following command by changing the location for the `-cert` option for each intermediate CA.

  ```
  orapki wallet add -wallet wallet_location -trusted_cert -cert
  intermediate_certificate_location [-pwd <pwd>] | [-
  auto_login_only]
  ```

  For example:

  ```
  ${PLUGINS_HOME}/bin/orapki  wallet add -wallet test_wallet -
  trusted_cert -cert IM_CA.crt -auto_login_only
  ```

  Sample output:

  ```
  Operation is successfully completed.
  ```

Display the wallet after importing the root CA and the intermediate CA:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
Subject:        CN=localhost,O=testing_only
User Certificates:
Trusted Certificates:
Subject:        CN=im_ca,OU=O,O=oracle,C=IN
Subject:        CN=root_ca,OU=O,O=oracle,C=IN
```

v. Import the certificate signed by CA as a user certificate to the Oracle wallet.

```
orapki wallet add -wallet wallet_location -user_cert -cert
certificate_location [-pwd <pwd>] | [-auto_login_only]
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
user_cert -cert user_1.crt -auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

Wallet content after adding the user certificate signed from the intermediate CA:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
User Certificates:
Subject:        CN=localhost,O=testing_only
Trusted Certificates:
Subject:        CN=im_ca,OU=O,O=oracle,C=IN
Subject:        CN=root_ca,OU=O,O=oracle,C=IN
```

3. Add the back-end server certificate as a trusted certificate.
   a. If the back-end server certificate is a self-singed certificate, then import it as a trusted certificate.
      i. View the back-end server certificate:

```
${PLUGINS_HOME}/bin/orapki cert display -cert
<backend_server_certificate> -complete
```

For example:

```
${PLUGINS_HOME}/bin/orapki  cert display -cert backend.crt -complete
```

Sample output:

```
{ fingerprint = ... holder = ... CN=Backend_Server,O=testing_only, issuer =
CN=Backend_Server,O=testing_only, ...
]} } }
```

      ii. Import the back-end server certificate as a trusted certificate to the wallet:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet --
trusted_cert -cert <back_end_sever_crt> -auto_login_only
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet --
trusted_cert -cert <back_end_sever_crt> -auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

iii. Display the wallet content after adding the back-end server certificate as a trusted certificate:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet <wallet_path>
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
User Certificates:
...
Trusted Certificates:
...
Subject: CN=Backend_Server,O=testing_only
```

b. If the back-end server certificate is signed with a CA authority, then import the trust chain:

i. If the back-end server certificate is signed by the intermediate CA, then import the root CA and the intermediate CA certificates as trusted certificates to the wallet:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet <wallet_Location> -
trusted_cert -cert <CA_certificate> -auto_login_only
```

- Example 1:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
trusted_cert -cert Root_CA.crt -auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

- Example 2:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
trusted_cert -cert IM_CA.crt -auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

ii. Display the wallet content after adding the back-end server trust chain:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet <wallet_location>
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
User Certificates:
...
Trusted Certificates:
...
Subject: CN=im_ca,OU=O,O=oracle,C=IN
Subject: CN=root_ca,OU=O,O=oracle,C=IN
```

The list of parameters used in the `orapki` commands:

Table 4-2 Command Parameters

Parameter	Description
`-wallet`	Specifies the wallet location.
`-dn`	Specifies the distinguished name of the certificate.
`-trusted_cert`	Specifies that it is a trusted certificate.
`-user_cert`	Specifies that it is a user certificate.
`-pwd`	Specifies the wallet password if the wallet is password protected.
`-auto_login_only`	Specifies if the wallet is auto_login_only or not.
`-request`	Specifies the location of the certificate request for the certificate you are creating.
`-cert`	Specifies the directory location of the certificate.
`-keysize`	Specifies the key size for the certificate.
`-self_signed`	Causes the tool to create a root certificate.
`-validity`	Specifies the number of days, starting from the current date, that the root certificate will be valid.
`-sign_alg`	Specifies the sign algorithm to be used.

Replacing the Existing Certificates with SHA-2 Signed Certificates

If the wallet has a mix of certificates which are signed either with the MD5 or the SHA-2 algorithm, you may want to remove only those certificates which are signed with the MD5 algorithm and keep the certificates that are signed with the SHA-2 algorithm.

For example:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
User Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
Trusted Certificates:
```

```
Subject: CN=localhost,O=FOR TESTING ONLY
Subject: CN=im_ca,OU=O,O=oracle,C=IN
Subject: CN=root_ca,OU=O,O=oracle,C=IN
Subject: CN=test_SHA2_signed_cert,OU=O,O=oracle,C=IN

test_wallet contains following certificates signed with MD5 algorithm :
Self-signed user certificate : Subject: CN=localhost,O=FOR TESTING ONLY
Trusted certificates :
Subject: CN=im_ca,OU=O,O=oracle,C=IN
Subject: CN=root_ca,OU=O,O=oracle,C=IN
```

After you have identified the certificate request by which the user and trusted certificates are signed with MD5, complete the following steps to remove them from wallet:

1. Remove the CA-signed or the self-signed user certificate:
   a. Check whether the certificate is self-signed or CA-signed:
      i. Display the wallet content and get the Distinguished Name:

```
${PLUGINS_HOME}/bin/orapki   wallet display -wallet
<wallet_location>
```

For example:

```
${PLUGINS_HOME}/bin/orapki   wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
User Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
Trusted Certificates:
Subject: CN=localhost,O=FOR TESTING ONLY
Subject: CN=im_ca,OU=O,O=oracle,C=IN
Subject: CN=root_ca,OU=O,O=oracle,C=IN
Subject: CN=test_SHA2_singed_cert,OU=O,O=oracle,C=IN
```

The Distinguished Name for the user certificates is "**CN=localhost,O=FOR TESTING ONLY**"

The `display -wallet` command shows the user certificate and the trusted certificate present in the wallet.

      ii. Export the user certificate to a file.

```
${PLUGINS_HOME}/bin/orapki  wallet export -wallet <wallet_Location>
-dn 'DN_string' -cert <certificate_file>
```

For example:

```
${PLUGINS_HOME}/bin/orapki  wallet export -wallet <wallet_Location>
-dn 'CN=localhost,O=FOR TESTING ONLY' -cert user.crt
```

      iii. View the user certificate.

```
${PLUGINS_HOME}/bin/orapki  cert display -cert <user_cert>
```

For example:

- For a self-signed certificate, the `Subject` and `Issuer` names are same, as given below:

```
${PLUGINS_HOME}/bin/orapki  cert display -cert user.crt
```

Sample output:

```
Subject:        CN=localhost,O=FOR TESTING ONLY
Issuer:         CN=localhost,O=FOR TESTING ONLY
Valid Until:    Thu Oct 07 15:15:55 UTC 2117
```

- For a CA-signed certificate, the `Subject` and `Issuer` names are different, as given below:

```
${PLUGINS_HOME}/bin/orapki  cert display -cert user.crt
```

Sample output:

```
Subject:  CN=localhost,O=FOR TESTING ONLY
Issuer:    CN=im_ca,OU=O,O=FOR TESTING ONLY
Valid Until:    Thu Oct 07 15:15:55 UTC 2117
```

b. Remove the self-signed certificate from the trusted and user certificate lists and also remove the certificate request associated with the self-signed certificate:

```
${PLUGINS_HOME}/bin/orapki wallet remove -wallet < wallet_location > -
dn 'DN_string' -trusted_cert [-pwd <pwd>] | [-auto_login_only]
```

```
${PLUGINS_HOME}/bin/orapki wallet remove -wallet < wallet_location > -
dn 'DN_string' -user_cert [-pwd <pwd>] | [-auto_login_only]
```

```
${PLUGINS_HOME}/bin/orapki wallet remove -wallet < wallet_location > -
dn 'DN_string' -cert_req [-pwd <pwd>] | [-auto_login_only]
```

c. If a user certificate is CA-signed, then remove the user certificate:

```
${PLUGINS_HOME}/bin/orapki wallet remove -wallet <wallet_location> -dn
'DN_string' -user_cert [-pwd <pwd>] | [-auto_login_only]
```

d. Remove the trusted certificate signed using the MD5 algorithm:

```
${PLUGINS_HOME}/bin/orapki wallet remove -wallet < wallet_location > -
dn 'DN_string' -trusted_cert [-pwd < pwd >] | [-auto_login_only]
```

e. Remove the certificate request signed using the MD5 algorithm:

```
${PLUGINS_HOME}/bin/orapki wallet remove -wallet < wallet_location > -
dn 'DN_string' -cert_req [-pwd <pwd>] | [-auto_login_only]
```

2. Create and import the certificates to the wallet:

a. Add a self-signed user certificate signed with the SHA-2 algorithm:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet <wallet_Location> -dn
'DN_String' -keysize 2048 -sign_alg sha256 -self_signed -validity 9125
[-pwd <pwd>] | [-auto_login_only]
```

b. Add a CA-signed user certificate signed with the SHA-2 algorithm:

i. Add the certificate request to the Oracle wallet.

```
orapki wallet add -wallet wallet_location -dn user_dn -keySize 512|
1024|2048 [-pwd <pwd>] | [-auto_login_only]
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -dn
"CN=localhost,O=testing_only" -keysize 2048 -auto_login_only
```

Sample output:

```
Operation is successfully completed

Wallet content after adding certificate request
${PLUGINS_HOME}/bin/orapki wallet display -wallet /scratch/shichoud/
test_wallet
Requested Certificates:
Subject:        CN=localhost,O=testing_only
User Certificates:
Trusted Certificates:
```

ii. Export the certificate request from Oracle wallet:

```
orapki wallet export -wallet wallet_location -dn
certificate_request_dn -request certificate_request_filename
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet export -wallet test_wallet -dn
"CN=localhost,O=testing_only" -request user.csr
```

Sample output:

```
Operation is successfully completed.
```

To view the content of the certificate, run the following command:

```
cat user.csr
```

Sample output:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICcDCCAVgCAQAwKzEVMBMGA1UECgwMdGVzdGluZ19vbmx5MRIwEAYDVQQDEwls
...
...
WnDd1cweMAH+1/D1C4Gi7Gvhi2Axw18H60lmZcU3JXv2bhu8QxZI9N6sI1DjU2Mg
l6EH2w==
```

See Exporting Certificates and Certificate Requests from Oracle Wallets with orapki in *Administering Oracle Fusion Middleware*.

iii. Use the certificate request exported in Step 3 to create a certificate by using a certificate authority (CA) or some other mechanism.

iv. Import all trusted certificates in the certificate chain of a user certificate before adding a user certificate. The certificate chain includes the intermediate certificate authorities and the root certificate authority.

- Import the root CA certificate as a trusted certificate by using the following command:

```
orapki wallet add -wallet wallet_location -trusted_cert -cert
root_certificate_location [-pwd <pwd>] | [-auto_login_only]
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
trusted_cert -cert Root_CA.crt -auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

- Import the intermediate CA certificate as a trusted certificate. If there are more than one intermediate CA certificate, execute the following command by changing the location for the `-cert` option for each intermediate CA.

```
orapki wallet add -wallet wallet_location -trusted_cert -cert
intermediate_certificate_location [-pwd <pwd>] | [-
auto_login_only]
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
trusted_cert -cert IM_CA.crt -auto_login_only
```

Display the wallet after importing the root CA and the intermediate CA:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
Subject:        CN=localhost,O=testing_only
User Certificates:
Trusted Certificates:
Subject:        CN=im_ca,OU=O,O=oracle,C=IN
Subject:        CN=root_ca,OU=O,O=oracle,C=IN
```

v. Import the certificate signed by CA as a user certificate to the Oracle wallet.

```
orapki wallet add -wallet wallet_location -user_cert -cert
certificate_location [-pwd <pwd>] | [-auto_login_only]
```

For example:

```
${PLUGINS_HOME}/bin/orapki wallet add -wallet test_wallet -
user_cert -cert user_1.crt -auto_login_only
```

Sample output:

```
Operation is successfully completed.
```

Wallet content after adding the user certificate signed from the intermediate CA:

```
${PLUGINS_HOME}/bin/orapki wallet display -wallet test_wallet
```

Sample output:

```
Requested Certificates:
User Certificates:
Subject:        CN=localhost,O=testing_only
Trusted Certificates:
Subject:        CN=im_ca,OU=O,O=oracle,C=IN
Subject:        CN=root_ca,OU=O,O=oracle,C=IN
```

The list of parameters used in the `orapki` commands:

Table 4-3 Command Parameters

Parameter	Description
`-wallet`	Specifies the wallet location.
`-dn`	Specifies the distinguished name of the certificate.
`-trusted_cert`	Specifies that it is a trusted certificate.
`-user_cert`	Specifies that it is a user certificate.
`-cert_req`	Specifies that it is a certificate request.
`-pwd`	Specifies the wallet password if the wallet is password protected.
`-auto_login_only`	Specifies if the wallet is auto_login_only or not.
`-request`	Specifies the location of the certificate request for the certificate you are creating.
`-cert`	Specifies the directory location of the certificate.
`-keysize`	Specifies the key size for the certificate.
`-self_signed`	Causes the tool to create a root certificate.
`-validity`	Specifies the number of days, starting from the current date, that the root certificate will be valid.
`-sign_alg`	Specifies the sign algorithm to be used.

Certificates Signed with MD5 Algorithm Not Supported

Certificates signed using MD5 algorithm are not recommended due to compromised security.

These certificates are no longer supported with the 14.1.2.0.0 WLS proxy plug-ins. The proxy plug-ins refuse to start if MD5 certificates are present in the Oracle Wallet.

Using Certificates Signed with RSASSA-PSS Signature Algorithm

Certificates signed with RSASSA-PSS signature algorithm are very secure and are supported in the Oracle WebLogic Server 14.1.2.0.0 Proxy Plug-ins.

Certificates signed with RSASSA-PSS signature algorithm and private keys generated using the RSASSA-PSS algorithm can be deployed when using TLSv1.3 for communication between the web server and the back-end Oracle WebLogic Server.

With Oracle WebLogic Server 14.1.2.0.0 Proxy Plug-ins, you can configure an RSA certificate with a signature algorithm of RSASSA-PSS as a user certificate for the Apache Web Server. You can then use this certificate to function as a client certificate when the Oracle WebLogic Server requires one for client authentication.

If you have configured Oracle WebLogic Server to use a certificate with RSASSA-PSS signature, the Oracle WebLogic Server 14.1.2.0.0 Proxy Plug-ins support such certificates during an SSL handshake.

Configuring Perimeter Authentication

Use perimeter authentication to secure Oracle WebLogic Server applications that are accessed by using the proxy plug-in.

A WebLogic Identity Assertion Provider authenticates tokens from outside systems that access your Oracle WebLogic Server application, including users who access your Oracle WebLogic Server application through the proxy plug-in. Create an Identity Assertion Provider that will safely secure your proxy plug-in as follows:

1. Create a custom Identity Assertion Provider on your Oracle WebLogic Server application. See How to Develop a Custom Identity Assertion Provider in *Developing Security Providers for Oracle WebLogic Server*.
2. Configure the custom Identity Assertion Provider to support the Cert token type and make Cert the active token type. See How to Create New Token Types in *Developing Security Providers for Oracle WebLogic Server*.
3. Set clientCertProxy to True in the web.xml deployment descriptor file for the Web application.

   The `clientCertProxy` attribute can be used with a third party proxy server, such as a load balancer or an SSL accelerator, to enable 2-way SSL authentication. For more information about the `clientCertProxy` attribute, see context-param in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.
4. Once you have set `clientCertProxy`, be sure to use a connection filter to ensure that Oracle WebLogic Server accepts connections only from the machine on which the proxy plug-in is running. See Using Network Connection Filters in *Developing Applications with the WebLogic Security Service*.
5. The Oracle WebLogic Server Proxy Plug-ins require a trusted Certificate Authority file to use SSL between the proxy plug-in and Oracle WebLogic Server. See Using SSL with Proxy Plug-Ins for the steps you need to perform to configure SSL.

See Identity Assertion Providers in *Developing Security Providers for Oracle WebLogic Server*.

About Federal Information Processing Standards

Federal Information Processing Standards (FIPS) is not supported in the 14.1.2.0.0 WLS proxy plug-ins.

If FIPS is configured directly at the Oracle WebLogic Server side, and if a request is made through the WLS proxy plug-in with a front-end Apache HTTP Server, the request will fail.

5

Parameters for Oracle WebLogic Server Proxy Plug-Ins

Learn about the parameters that you can use to configure Oracle HTTP Server.

> **Note:**
>
> The parameters for the WLS proxy plug-ins should be specified in special configuration files, which are named and formatted uniquely for each web server. For information about the configuration files specific to the proxy plug-ins for Oracle HTTP Server, see Configuring the Plug-In for Oracle HTTP Server.

- General Parameters for Oracle WebLogic Server Proxy Plug-Ins
  The names of the general parameters for the WLS proxy plug-ins are case sensitive.
- SSL Parameters for Oracle WebLogic Server Proxy Plug-Ins
  The names of the SSL parameters for WLS proxy plug-ins are case sensitive.

General Parameters for Oracle WebLogic Server Proxy Plug-Ins

The names of the general parameters for the WLS proxy plug-ins are case sensitive.

This section includes the following topics:

- ConnectRetrySecs
- ConnectTimeoutSecs
- DebugConfigInfo
- DefaultFileName
- DynamicServerList
- ErrorPage
- FileCaching
- Idempotent
- KeepAliveEnabled
- KeepAliveSecs
- MatchExpression
- MaxPostSize
- MaxSkipTime
- PathPrepend
- PathTrim
- QueryFromRequest

ConnectRetrySecs

Default: 2

To specify no retries, set `ConnectRetrySecs` equal to `ConnectTimeoutSecs`. However, the proxy plug-in attempts to connect at least twice.

You can customize the error response by using the `ErrorPage` parameter.

ConnectTimeoutSecs

Default: 10

Maximum time in seconds that the proxy plug-in should attempt to connect to the Oracle WebLogic Server host. Make the value greater than `ConnectRetrySecs`. If `ConnectTimeoutSecs` expires without a successful connection, even after the appropriate retries (see `ConnectRetrySecs`), an `HTTP 503/Service Unavailable` response is sent to the client.

You can customize the error response by using the `ErrorPage` parameter.

DebugConfigInfo

Default: OFF

Enables the special query parameter "__WebLogicBridgeConfig". Use it to get details about configuration parameters from the proxy plug-in.

For example, if you enable "__WebLogicBridgeConfig" by setting `DebugConfigInfo` and then send a request that includes the query string ?__WebLogicBridgeConfig, then the proxy plug-in gathers the configuration information and run-time statistics and returns the information to the browser. The proxy plug-in does not connect to Oracle WebLogic Server in this case.

This parameter is strictly for debugging and the format of the output message can change with releases. For security purposes, keep this parameter turned OFF in production systems.

DefaultFileName

Default: none

If the URI is "/" then the proxy plug-in performs the following steps:

1. Trims the path specified with the `PathTrim` parameter.
2. Appends the value of `DefaultFileName`.
3. Prepends the value specified with `PathPrepend`.

This procedure prevents redirects from Oracle WebLogic Server.

Set the DefaultFileName to the default welcome page of the Web application in Oracle WebLogic Server to which requests are being proxied. For example, If the `DefaultFileName` is set to welcome.html, an HTTP request like "http://somehost/weblogic" becomes "http://somehost/weblogic/welcome.html". For this parameter to function, the same file must be specified as a welcome file in all the Web Applications to which requests are directed. See Configuring Welcome Files in *Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

Note for Apache users: If you are using Stronghold or Raven versions, define this parameter inside of a `Location` block, and not in an `IfModule` block.

DynamicServerList

Default: ON

When set to `OFF`, the proxy plug-in ignores the dynamic cluster list used for load balancing requests proxied from the proxy plug-in and only uses the static list specified with the `WebLogicCluster` parameter. Normally this parameter should remain set to `ON`.

There are some implications for setting this parameter to `OFF`:

- If one or more servers in the static list fails, the proxy plug-in could waste time trying to connect to a terminated server, resulting in decreased performance.
- If you add a new server to the cluster, the proxy plug-in cannot proxy requests to the new server unless you redefine this parameter. Oracle WebLogic Server automatically adds new servers to the dynamic server list when they become part of the cluster.

ErrorPage

Default: none

You can create your own error page that is displayed when your Web server cannot forward requests to Oracle WebLogic Server.

The proxy plug-in redirects to an error page when the back-end server returns an HTTP 503/ Service Unavailable response and there are no servers for failover.

FileCaching

Default: ON

When set to `ON`, and the size of the POST data in a request is greater than 2048 bytes, the POST data is first read into a temporary file on disk and then forwarded to Oracle WebLogic Server in chunks of 8192 bytes. This preserves the POST data during failover, allowing all necessary data to be repeated to the secondary if the primary goes down.

When `FileCaching` is `ON`, any client that tracks the progress of the POST will see that the transfer has completed even though the data is still being transferred between the WebServer and WebLogic. So, if you want the progress bar displayed by a browser during the upload to reflect when the data is actually available on the Oracle WebLogic Server, you might not want to have `FileCaching ON`.

When set to `OFF` and the size of the POST data in a request is greater than 2048 bytes, the reading of the POST data is postponed until an Oracle WebLogic Server cluster member is identified to serve the request. Then the proxy plug-in reads and immediately sends the POST data to Oracle WebLogic Server in chunks of 8192 bytes.

Turning `FileCaching OFF` limits failover. If the Oracle WebLogic Server primary server goes down while processing the request, the POST data already sent to the primary cannot be repeated to the secondary.

Finally, regardless of how `FileCaching` is set, if the size of the POST data is 2048 bytes or less the proxy plug-in will read the data into memory and use it if needed during failover to repeat to the secondary.

- Location of POST Data Files

Location of POST Data Files

When the `FileCaching` parameter is set to `ON` and the size of the POST data in a request is greater than 2048 bytes, the POST data is first read into a temporary file on disk, and then forwarded to Oracle WebLogic Server in chunks of 8192 bytes. This preserves the POST data during failover.
The temporary POST file is located under `/tmp/_wl_proxy` for UNIX. For Windows it is located as follows (if `WLTempDir` is not specified):

- Environment variable `TMP`
- Environment variable `TEMP`
- `C:\Temp`

Idempotent

Default: ON

Applies to: Oracle HTTP Server, Apache HTTP Server, Microsoft IIS

Prior to WebLogic Plug-in 14.1.2.0.0, when Idempotent parameter is set to `ON` and if the back-end server do not respond within `WLIOTimeoutSecs`, the proxy plug-ins failover if the method is Idempotent. However, from WebLogic Plug-in version 14.1.2 for Oracle HTTP Server and Apache HTTP Server, if the back-end server do not respond within `WLIOTimeoutSecs`, the failover is controlled by `WLRetryOnTimeout` parameter.

The proxy plug-ins also failover if Idempotent is set to `ON` and the servers respond with an error such as `READ_ERROR_FROM_SERVER`.

If Idempotent is set to `OFF`, the proxy plug-ins do not failover. If you are using the Apache HTTP Server, you can set this parameter differently for different URLs or MIME types.

Idempotent only takes effect if the request is successfully sent to Oracle WebLogic Server and the proxy plug-in is now waiting for a response from the back-end server.

POST requests are not retried even if marked as Idempotent.

KeepAliveEnabled

Default: `ON`

This directive enables pooling of connections between the proxy plug-in and Oracle WebLogic Server. Valid values are `ON` and `OFF`.

While using Apache prefork mpm, Apache web server might fail. Set `KeepAliveEnabled` to `OFF` when using prefork mpm or use worker mpm in Apache.

> **Note:**
>
> If both `KeepAliveEnabled` and HTTP/2 are configured for a back-end connection, the following message is generated:
> `KeepAliveEnabled option will be ignored since HTTP/2 connection is enabled`

KeepAliveSecs

Default: `20`

The length of time after which an inactive connection between the proxy plug-in and Oracle WebLogic Server is closed. You must set `KeepAliveEnabled` to `true` (`ON` when using the Apache HTTP Server) for this parameter to be effective.

The value of this parameter must be less than or equal to the value of the `Duration` field set in the Remote Console on the **Server > Protocols > HTTP** page, or the value set on the server MBean with the `KeepAliveSecs` attribute.

MatchExpression

Default: none

Use this parameter to modify the values of existing parameters or add a new parameter for a particular configuration.

The MatchExpression parameter supports only the `*` and `?` regular expressions

- `*` which matches 0 or more characters
- `?` which matches exactly one character

This parameter can be configured for two scenarios.

Proxying by MIME type:

You can use this parameter in the following format to set other parameters for a particular MIME type.

Syntax:

```
MatchExpression <file_extension> <param=value>|<param-value>|…
```

For example, the following configuration proxies `*.jsp` to `myHost:8080`:

```
<IfModule weblogic_module>
MatchExpression *.jsp WebLogicHost=myHost|WebLogicPort=8080
</IfModule>
```

Proxying by path:

You can also use this parameter in the following format to set other parameters for a particular path.

Syntax:

```
MatchExpression <path> <param=value>|<param-value>|…
```

For example, the following configuration proxies the URIs beginning with `/weblogic` to `myHost:9090`:

```
<IfModule weblogic_module>
MatchExpression /weblogic WebLogicHost=myHost|WebLogicPort=9090
</IfModule>
```

You can also use `MatchExpression` to override the parameter values, as shown above. It can also be used to define new parameters (this is, those that have not been used in the configuration).

For example, the configuration below proxies all the requests to my`Host:8080`. The URIs that match the type `jpg` will be proxied to `myHost:8080/images` and others will be proxied to my`Host:8080`.

```
<IfModule weblogic_module>
WLSRequest On
WebLogicHost myHost
WebLogicPort 8080
MatchExpression *.jpg PathPrepend=/images
</IfModule>
```

MaxPostSize

Default: 0

Maximum allowable size of POST data, in bytes. If the content-length exceeds `MaxPostSize`, the proxy plug-in returns an error message. If set to 0, the size of POST data is not checked. This is useful for preventing denial-of-service attacks that attempt to overload the server with POST data.

MaxSkipTime

Default: 10

If Oracle WebLogic Server listed in either the `WebLogicCluster` parameter or a dynamic cluster list returned from Oracle WebLogic Server fails, the failed server is marked as "bad" and the proxy plug-in attempts to connect to the next server in the list.

`MaxSkipTime` sets the amount of time after which the proxy plug-in will retry the server marked as "bad." The proxy plug-in attempts to connect to a new server in the list each time a unique request is received (that is, a request without a cookie).

> **Note:**
>
> If the weighted least connection routing algorithm is chosen, Oracle recommends the value of MaxSkipTime to be the "no of servers" times ConnectRetrySecs as shown below:
>
> ```
> MaxSkipTime = (no of servers) * ConnectRetrySecs
> ```

PathPrepend

Default: null

As per the RFC specification, generic syntax for URL is:

```
[PROTOCOL]://[HOSTNAME]:{PORT}/{PATH}/{FILENAME};{PATH_PARAMS}/{QUERY_STRING}...
```

`PathPrepend` specifies the path that the proxy plug-in prepends to the `{PATH}` portion of the original URL, after PathTrim is trimmed and before the request is forwarded to Oracle WebLogic Server.

If you must append a File Name, use `DefaultFileName` parameter instead of `PathPrepend`.

PathTrim

Default: null

As per the RFC specification, generic syntax for URL is:

```
[PROTOCOL]://[HOSTNAME]:{PORT}/{PATH}/{FILENAME};{PATH_PARAMS}/{QUERY_STRING}...
```

`PathTrim` specifies the string trimmed by the proxy plug-in from the `{PATH}/{FILENAME}` portion of the original URL, before the request is forwarded to Oracle WebLogic Server. For example, if the `http://myWeb.server.com/weblogic/foo` URL is passed to the proxy plug-in for parsing

and if `PathTrim` has been set to strip off `/weblogic`, before handing the URL to Oracle WebLogic Server, the URL forwarded to Oracle WebLogic Server is `http://myWeb.server.com:7001/foo`.

If you are newly converting an existing third-party server to proxy requests to Oracle WebLogic Server using the proxy plug-in, you will need to change application paths to `/foo` to include `weblogic/foo`. You can use `PathTrim` and `PathPrepend` in combination to change this path.

Configure the `PathTrim` parameter inside the `<Location>` tag.

The following configuration is **incorrect** because the `PathTrim` parameter is not configured inside the `<Location> tag`:

```
<Location /weblogic>
WLSRequest On
</Location>

<IfModule mod_weblogic.c>
WebLogicHost localhost
WebLogicPort 7001
PathTrim /weblogic
</IfModule>
```

The following configuration is **correct**:

```
<Location /weblogic>
 WLSRequest On
 PathTrim /weblogic
</Location>
```

The `<Location>` directive limits the scope of the enclosed directives by URL. See Apache Location Directive.

QueryFromRequest

Default: OFF

When set to `ON`, specifies that the Apache HTTP Server use

```
(request_rec *)r->the_request
```

to pass the query string to Oracle WebLogic Server. (For more information, see the Apache documentation.) This behavior is desirable when a Netscape version 4.x browser makes requests that contain spaces in the query string

When set to `OFF`, the Apache HTTP Server uses `(request_rec *)r->args` to pass the query string to Oracle WebLogic Server.

RoutingAlgorithm

Specifies how new requests are routed to Oracle WebLogic Server back-end servers.

Default: `Round-Robin`

Supported Values:

- `Round-Robin` - Select this algorithm to route new requests to the next server in a cluster.
- `Weighted-Least-Connection` - Select this algorithm to route new requests according to intelligent load balancing which is based on the weighted least connection algorithm. This algorithm selects the next server based on its current load capacity as determined by a health score provided by WebLogic Server. For more information, see Support for Intelligent Load Balancing.

Sample configuration:

```
<Location /app>
 WLSRequest On
 WebLogicCluster <Weblogic Managed Server 1>,<Weblogic Managed Server
2>,<Weblogic Managed Server 3> ...
 RoutingAlgorithm Weighted-Least-Connection
</Location>
```

> **Note:**
>
> Configure the `WeightUpdateInterval` directive to specify how frequently the WebLogic Server Proxy Plug-in requests health scores from the back-end server.

WebLogicCluster

Required when proxying to a cluster of Oracle WebLogic Servers, or to multiple non-clustered servers.

Default: none

The `WebLogicCluster` parameter is required to proxy a list of back-end servers that are clustered, or to perform load balancing among non-clustered managed server instances.

List of Oracle WebLogic Servers that can be used for load balancing. The server or cluster list is a list of `host:port` entries. If a mixed set of clusters and single servers is specified, the dynamic list returned for this parameter will return only the clustered servers.

For the syntax for specifying the value of this parameter for Oracle HTTP Server, see Configuring the Plug-In for Oracle HTTP Server.

If you are using SSL between the proxy plug-in and Oracle WebLogic Server, set the port number to the SSL listen port and set the SecureProxy parameter to ON.

The proxy plug-in does a simple round-robin between all available servers. The server list specified in this property is a starting point for the dynamic server list that the server and proxy plug-in maintain. Oracle WebLogic Server and the proxy plug-in work together to update the server list automatically with new, failed, and recovered cluster members.

You can disable the use of the dynamic cluster list by setting the DynamicServerList parameter to `OFF`.

The proxy plug-in directs HTTP requests containing a cookie, URL-encoded session, or a session stored in the POST data to the server in the cluster that created the cookie.

WebLogicHost

Required when proxying to a single Oracle WebLogic Server.

Default: none

Oracle WebLogic Server host (or virtual host name as defined in Oracle WebLogic Server) to which HTTP requests should be forwarded. If you are using a Oracle WebLogic cluster, use the `WebLogicCluster` parameter instead of `WebLogicHost`.

WebLogicPort

Required when proxying to a single Oracle WebLogic Server.

Default: none

The port at which Oracle WebLogic Server host is listening for connection requests from the proxy plug-in (or from other servers). (If you are using SSL between the proxy plug-in and Oracle WebLogic Server, set this parameter to the SSL listen port and set the `SecureProxy` parameter to `ON`).

If you are using a Oracle WebLogic Cluster, use the `WebLogicCluster` parameter instead of `WebLogicPort`.

WeightUpdateInterval

Specifies how often (in seconds) the Oracle WebLogic Server Proxy Plug-in requests updates on server weight to determine a server's capacity for new traffic.

Default: `1`

When intelligent load balancing is enabled, the `WeightUpdateInterval` directive determines how frequently Oracle WebLogic Server Proxy Plug-in requests updates from WebLogic Server on the weight of each server in a cluster through the header, `X-WebLogic-Request-Server-Health-Score`.

To enable intelligent load balancing, set the `RoutingAlgorithm` directive to `Weighted-Least-Connection` .

The weight of the servers are updated when one of the following conditions is satisfied:

- The time elapsed between now and the previous weight update exceeds the value set by `WeightUpdateInterval`
- The entire cluster of servers is changed

Sample configuration:

```
<Location /app>
 WLSRequest On
 WebLogicCluster <Weblogic Managed Server 1>,<Weblogic Managed Server
2>,<Weblogic Managed Server 3> ...
 RoutingAlgorithm Weighted-Least-Connection
 WeightUpdateInterval 3
</Location>
```

For improved performance, consider lowering the frequency of weight update requests.

WLCookieName

Default: `JSESSIONID`

If you change the name of the Oracle WebLogic Server session cookie in the Oracle WebLogic Server Web application, then you must change the `WLCookieName` parameter in the proxy plug-in to the same value. The name of the Oracle WebLogic session cookie is set in the WebLogic-specific deployment descriptor, in the `<session-descriptor>` element in `weblogic.xml`.

WLDNSRefreshInterval

Default: 0 (Lookup once, during startup)

If defined in the proxy configuration, specifies number of seconds interval at which Oracle WebLogic Server refreshes DNS name to IP mapping for a server. This can be used if an Oracle WebLogic Server instance is migrated to a different IP address, but the DNS name for that server's IP remains the same. In this case, at the specified refresh interval the DNS<->IP mapping will be updated.

WLExcludePathOrMimeType

Default: none

This parameter allows you to exclude certain requests from proxying.

This parameter can be defined locally at the Location tag level and globally. When the property is defined locally, it does not override the global property but defines a union of the two parameters.

WLForwardUriUnparsed

Default: OFF

When set to `ON`, the Oracle WebLogic Server Proxy Plug-in will forward the original URI from the client to Oracle WebLogic Server. When set to `OFF` (default), the URI sent to Oracle WebLogic Server is subjected to modification by `mod_rewrite` or other Web Server Plug-in modules.

WLIOTimeoutSecs

New name for `HungServerRecoverSecs`.

Default: 120

Defines the amount of time the proxy plug-in waits for a response to a request from Oracle WebLogic Server. The proxy plug-in waits for `WLIOTimeoutSecs` for the server to respond, and then declares that the server is dead, and fails over to the next server. You must set the value to a large value. If the value is less than the time the servlets take to process, you might see unexpected results.

Minimum value: 10

Maximum value: 2147483647

WLLocalIP

Default: none

Defines the IP address (on the proxy plug-in's system) to bind to when the proxy plug-in connects to an Oracle WebLogic Server instance running on a multihomed machine.

If `WLLocalIP` is not set, the TCP/IP stack will choose the source IP address.

WLMaxWebSocketClients

Default: Windows: Half of `ThreadsPerChild`, Non-Windows: Half of `MaxRequestWorkers`

Limits the number of active WebSocket connections at any instant of time.

> **Note:**
>
> The maximum value you can set for this parameter is 75 percent of `ThreadsPerChild` (Windows) or 75 percent of `MaxRequestWorkers` (non-Windows). If the value specified for this parameter is greater than the maximum allowed, it will be automatically lowered to that maximum.

WLProtocol

Default: `http/1.1`

Applies to: Apache HTTP Server

Scope: Location, Server context

Supported Values:

- `http/2` - for HTTP/2 protocol

  > **Note:**
  >
  > WLS Apache Plug-In *only*.

- `http/1.1` - for HTTP/1.1 protocol

This directive specifies the protocol to be used by the WLS Apache Plug-In to communicate with the back-end server.

If the `WLProtocol` directive is not configured, then the WLS Apache Plug-In sends requests to the back-end server over HTTP/1.1. If the `WLProtocol` is configured as HTTP/2 and the back-end server does not support the HTTP/2 protocol, the WLS Apache Plug-In does not fallback to HTTP/1.1 and the 503 error is returned.

WLProxyPassThrough

Default: OFF

If you have a chained proxy setup, where a proxy plug-in is running behind some other proxy or load balancer, you must explicitly enable the `WLProxyPassThrough` parameter. This parameter allows the header to be passed through the chain of proxies.

WLProxySSL

Default: OFF

Set this parameter to `ON` to maintain SSL communication between the proxy plug-in and Oracle WebLogic Server when the following conditions exist:

- An HTTP client request specifies the HTTPS protocol.
- The request is passed through one or more proxy servers (including the Oracle WebLogic Server Proxy Plug-in).
- The connection between the proxy plug-in and Oracle WebLogic Server uses the HTTP protocol.

When `WLProxySSL` is set to `ON`, the location header returned to the client from Oracle WebLogic Server specifies the HTTPS protocol.

WLProxySSLPassThrough

Default: OFF

If a load balancer or other software deployed in front of the web server and proxy plug-in is the SSL termination point, and that product sets the `WL-Proxy-SSL` request header to `true` or `false` based on whether the client connected to it over SSL, set `WLProxySSLPassThrough` to `ON` so that the use of SSL is passed on to the Oracle WebLogic Server.

If the SSL termination point is in the web server where the proxy plug-in operates, or the load balancer does not set `WL-Proxy-SSL`, set `WLProxySSLPassThrough` to `OFF` (default).

WLRetryOnTimeout

Default: IDEMPOTENT

Applies to: Oracle HTTP Server, Apache HTTP Server

Tells the WebLogic Plug-in whether to retry requests (including POST requests) when a timeout occurs before the WebLogic server sends the status line. Valid arguments are:

- `ALL`: All requests are retried.
- `IDEMPOTENT`: Only requests that use idempotent methods are retried.
- `NONE`: No requests are retried.
- `ALL_NOREAD`: All requests are retried, where after sending the request nothing has been received.
- `IDEMPOTENT_NOREAD`: Only requests that use idempotent methods are retried, where after sending the request nothing has been received.

Prior to Oracle HTTP Server 14.1.2.0.0, the parameter Idempotent played a role where `IDEMPOTENT ON` caused a retry in case of response timeout. However, from Oracle HTTP Server 14.1.2.0.0 the retry due to response timeout is controlled only by the parameter `WLRetryOnTimeout`.

WLRetryAfterDroppedConnection

Default: ALL

Tells the Oracle WebLogic Server Proxy Plug-ins which requests to retry when a connection is lost before Oracle WebLogic Server sends the status line. Valid arguments are:

- `ALL`: All requests will be retried.
- `IDEMPOTENT`: Only requests using idempotent methods will be retried.
- `NONE`: No requests will be retried.

WLServerInitiatedFailover

Default: ON

This controls whether a 503 error response from Oracle WebLogic Server triggers a failover to another server. Normally, the proxy plug-in will attempt to failover to another server when a 503 error response is received. When `WLServerInitiatedFailover` is set to `OFF`, the 503 error response will be returned to the client immediately.

WLSocketTimeoutSecs

Default: 2 (must be greater than 0)

Set the timeout for the socket while connecting, in seconds. See `ConnectTimeoutSecs` and `ConnectRetrySecs` for additional details.

WLSRequest

Default: OFF

This is an alternative to the WLSRequest On mechanism of identifying requests to be forwarded to Oracle WebLogic Server. For example,

```
<Location /weblogic>
 WLSRequest ON
 PathTrim /weblogic
</Location>
```

The use of `WLSRequest` ON instead of `SetHandler weblogic-handler` has the following advantages:

- Lower web server processing overhead in general
- Resolves substantial performance degradation when the web server `DocumentRoot` is on a slow filesystem
- Resolves 403 errors for URIs which cannot be mapped to the filesystem due to the filesystem length restrictions

WLTempDir

Default: `/tmp`

For Apache HTTP Server, this directive specifies the location of the `_wl_proxy` directory for the POST data files.

SSL Parameters for Oracle WebLogic Server Proxy Plug-Ins

The names of the SSL parameters for WLS proxy plug-ins are case sensitive.

> **Note:**
>
> The SCG certificates are not supported for use with WLS proxy plug-ins. The non-SCG certificates work appropriately and allow SSL communication between Oracle WebLogic Server and the proxy plug-in.
>
> KeyStore-related initialization parameters are not supported for use with Oracle WebLogic Server Proxy Plug-in.

This section includes the following topics:

SecureProxy

Default: OFF

Set this parameter to ON to enable the use of the SSL protocol for all communication between the proxy plug-in and Oracle WebLogic Server. Remember to configure a port on the corresponding Oracle WebLogic Server for the SSL protocol before defining this parameter.

This parameter may be set at two levels: in the configuration for the main server and—if you have defined any virtual hosts—in the configuration for the virtual host. The configuration for the virtual host inherits the SSL configuration from the configuration of the main server if the setting is not overridden in the configuration for the virtual host.

WebLogicSSLCiphers

Ciphers Supported:

- TLSv1.3 ciphers:
  - `TLS_AES_256_GCM_SHA384`
  - `TLS_AES_128_GCM_SHA256`
  - `TLS_CHACHA20_POLY1305_SHA256`
- TLSv1.2 ciphers:
  - `ECDHE-RSA-AES256-GCM-SHA384`

- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-ECDSA-AES128-GCM-SHA256

Deprecated (yet available) TLSv1.2 ciphers:

- AES128-GCM-SHA256
- AES256-GCM-SHA384
- AES128-SHA256
- AES256-SHA256
- AES256-SHA
- AES128-SHA

Default: All supported ciphers

Scope: Server, VirtualHost

Applies to: Apache HTTP Server, Oracle HTTP Server

This directive accepts a space separated list of ciphers to be used between Oracle WebLogic Server Proxy Plug-in and Oracle WebLogic Server.

Note:

- If `WebLogicSSLCiphers` is set to TLSv1.3 ciphers and `WebLogicSSLVersion` is set to TLSv1.2, or vice versa, the following error message is thrown during the server startup:

  ```
  Error: No available SSL version. Possible Mismatch between the
  configured protocol(s) and cipher(s)
  ```

- If `WebLogicSSLCiphers` is omitted, then the default list of ciphers is selected (that is, all supported ciphers).
- If `WebLogicSSLVersion` is omitted, then the default list of protocols is selected, which is TLSv1.2 and TLSv1.3.
- If you want to remove a particular cipher, you must explicitly set a list of ciphers by omitting that cipher. Only the ciphers specified with the `WebLogicSSLCiphers` directive will be enabled.
- A web server starts if there is at least one supported cipher in the list of ciphers configured with the `WebLogicSSLCiphers` directive. If the list contains any unsupported cipher, a warning message is displayed and the unsupported cipher is ignored.
- Both TLSv1.2 (and below) and TLSv1.3 ciphers can be configured using the `WebLogicSSLCiphers` directive. For example:

  ```
  WebLogicSSLCiphers TLS_AES_256_GCM_SHA384 ECDHE-ECDSA-AES128-GCM-
  SHA256
  ```

WebLogicSSLVersion

Default: TLSv1.2, TLSv1.3

Specifies the SSL protocol version to use for communication between the proxy plug-in and the Oracle WebLogic Server. This setting need not match that of the web server's `ssl.conf` file. The proxy plug-in can have its own SSL version to communicate with Oracle WebLogic Server.

The following values are accepted:

- `TLSv1_2` or `TLSv1.2`: Uses TLS v1.2
- `TLSv1_3` or `TLSv1.3`: Uses TLS v1.3

For example:

```
WebLogicSSLVersion TLSv1_2 TLSv1_3
```

You can define multiple protocols by using a space-separated list. The SSL protocol version chosen is used for all the connections from the proxy plug-in to Oracle WebLogic Server. Hence, define this parameter at the global scope.

If not configured, the proxy plug-in uses the best protocol supported by both the proxy plug-in and Oracle WebLogic Server.

> **Note:**
>
> - The default minimum version of the Transport Layer Security (TLS) protocol configured is TLSv1.2. Oracle recommends that you use TLS v1.2 or later in a production environment.
> - If `WebLogicSSLCiphers` is set to TLSv1.3 ciphers and `WebLogicSSLVersion` is set to TLSv1.2, or vice versa, the following error message is thrown during the server startup:
>
>   ```
>   Error: No available SSL version. Possible Mismatch between the
>   configured protocol(s) and cipher(s)
>   ```

WLSSLCheckCn

Default: OFF

Scope: Location, Server context

Set this parameter to `ON` to enable the host name verification. Before you do that, ensure that the certificate meets the following requirement:

- The host name configured using the `WebLogicHost` or `WebLogicCluster` directive must match the `Common Name` attribute of the SSL certificate's `Distinguished Names` or the `subjectAltName` extension.
- The SSL certificate referred to here is the certificate configured for the Oracle WebLogic Server Managed Server serving the request.

WLSSLWallet

Default: none

Scope: Server context, Virtual Host context

The `WLSSLWallet` performs one-way or two-way SSL based on how SSL is configured for Oracle WebLogic Server. The export_wallet program exports the wallet into PEM formatted files on the file system. For each user-supplied wallet, three files may be created in the same file system path where only auto_login_only wallet is supported.

- `key.pem`: A pass-phrase protected file containing the private key in PEM format, if a private key is present in the wallet.
- `user.crt`: User certificate in PEM format, if a user certificate is present in the wallet.
- `trusted_certs.crt`: Chain of trusted certificates in PEM format.

Set the path of an Oracle Wallet (containing an SSO wallet file) as an argument. For example:

```
WLSSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/${COMPONENT_TYPE}/
instances/${COMPONENT_NAME}/keystores/default"
```

The WLSSLWallet directive is allowed in the Global Server context and <VirtualHost> context but not allowed in <Location> context. For example,

```
<IfModule weblogic_module>
 WLSSLWallet [Directory_Path_Of_Wallet]
...
</IfModule>
...
<Location /console>
 WLSRequest ON
 WebLogicHost [HOSTNAME]
 WebLogicPort [PORT]
 SecureProxy On
</Location>
```

6

Troubleshooting and Tuning Oracle WebLogic Server Proxy Plug-Ins

You might encounter some problems when using the WLS proxy plug-ins. Descriptions of how to solve these problems are provided.

This chapter includes the following topics:

- Tuning Oracle HTTP Server for High Throughput for WebSocket Upgrade Requests
  Oracle WebLogic Server 14c (14.1.2.0.0) supports deploying WebSocket applications. The 14.1.2.0.0 WLS OHS Plug-In can handle such WebSocket connection upgrade requests and effectively proxy to WebSocket applications hosted within Oracle WebLogic Server 14c (14.1.1.0.0) and later.
- Understanding Connection Errors and Clustering Failover
  When the proxy plug-in attempts to connect to Oracle WebLogic Server, the proxy plug-in uses several configuration parameters to determine how long to wait for connections to the Oracle WebLogic Server host and, after a connection is established, how long the proxy plug-in waits for a response.
- Oracle WebLogic Server Session Issues
- NO_RESOURCES Errors
  Occasionally, under stress conditions, a few requests might fail with the error logged in the error log file.
- POST Data Files Issues

Tuning Oracle HTTP Server for High Throughput for WebSocket Upgrade Requests

Oracle WebLogic Server 14c (14.1.2.0.0) supports deploying WebSocket applications. The 14.1.2.0.0 WLS OHS Plug-In can handle such WebSocket connection upgrade requests and effectively proxy to WebSocket applications hosted within Oracle WebLogic Server 14c (14.1.1.0.0) and later.

As a result of adding this support, a new configuration parameter `WLMaxWebSocketClients` is introduced.

The `WLMaxWebSocketClients` parameter limits the number of active WebSocket connections at any instant of time. The maximum value you can set for this parameter is 75 percent of `ThreadsPerChild` (Windows) or 75 percent of `MaxRequestWorkers` (non-Windows). Hence, to tune your HTTP Server for maximum WebSocket connection upgrade requests, set `MaxRequestWorkers`/`ThreadsPerChild` to a value that can accommodate WebSocket connections as well. Also, ensure that `WLMaxWebSocketClients` is set to 75 percent of `MaxRequestWorkers`/`ThreadsPerChild`.

Understanding Connection Errors and Clustering Failover

When the proxy plug-in attempts to connect to Oracle WebLogic Server, the proxy plug-in uses several configuration parameters to determine how long to wait for connections to the Oracle WebLogic Server host and, after a connection is established, how long the proxy plug-in waits for a response.

If the proxy plug-in cannot connect or does not receive a response, the proxy plug-in attempts to connect and send the request to the other Oracle WebLogic Server instances in the cluster. If the connection fails or there is no response from any Oracle WebLogic Server in the cluster, an error message is sent. For an illustration of how the proxy plug-in handles failover, see Figure 6-1.

This section includes the following topics:

Possible Causes of Connection Failures

Failure of the Oracle WebLogic Server host to respond to a connection request could indicate the following problems:

- Physical problems with the host machine (such as power outages, hardware malfunction, operating system crash, and so on).
- Network problems.
- Other server failures.

Failure of a Oracle WebLogic Server instance to respond could indicate the following problems:

- Oracle WebLogic Server is not running or is unavailable.
- A hung server.
- A database problem.
- An application-specific failure.

Tips for Reducing CONNECTION_REFUSED Errors

Under load, a proxy plug-in may receive CONNECTION_REFUSED errors from a back-end Oracle WebLogic Server instance. For example, the following error is logged in the log file:

```
weblogic: Trying GET /uri at backend host 'xx.xx.xx.xx/port; got exception
'CONNECTION_REFUSED [os error=xxx, line xxxx of URL.cpp]: apr_socket_connect call failed
with error=xxx, host=xx.xx.xx.xx, port=xxxx'
```

Oracle WebLogic Server might have reached the maximum allowed backlog connections. Follow these tuning tips to reduce CONNECTION_REFUSED errors:

- Increase the `AcceptBackLog` setting in the configuration of your Oracle WebLogic Server domain.
- Decrease the time wait interval. This setting varies according to the operating system you are using. For example, on Linux, set the `net.ipv4.tcp_fin_timeout` parameter to a lower value in the `/etc/sysctl.conf` file.
- Increase the open file descriptor limit on your machine. This limit varies by operating system. Using the limit (.csh) or ulimit (.sh) directives, you can make a script to increase the limit.

Failover with a Single, Non-Clustered Oracle WebLogic Server

If you run only a single Oracle WebLogic Server instance, the proxy plug-in only attempts to connect to the server defined with the `WebLogicHost` parameter. If the attempt fails, an HTTP 503 error message is returned. The proxy plug-in continues trying to connect to that same Oracle WebLogic Server instance for the maximum number of retries as specified by the ratio of `ConnectTimeoutSecs` and `ConnectRetrySecs`.

The Dynamic Server List

The `WebLogicCluster` parameter is required to proxy to a list of back-end servers that are clustered, or to perform load balancing among non-clustered managed server instances.

In the case of proxying to clustered managed servers, when you use the `WebLogicCluster` parameter to specify a list of Oracle WebLogic Servers, the proxy plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request is routed to one of these servers, a dynamic server list is returned containing an updated list of servers in the cluster.

The updated list adds any new servers in the cluster and deletes any that have been shut down, or are being suspended, or are no longer part of the cluster or that have failed to respond to requests. This feature can be controlled by using `DynamicServerList`. For example, to disable this feature, set `DynamicServerList` to `OFF`.

`DynamicServerList ON` is a preferred performance tuning parameter. It is useful, for example, if a member of a cluster is temporarily down for maintenance or if administrators decide they want to add another member, and not need to restart the web server.

> **Note:**
>
> If `DynamicServerList` is set to `ON`, and the list of the back-end Oracle WebLogic Servers specified in `WebLogicCluster` is not in a cluster, then the behavior would be undefined.

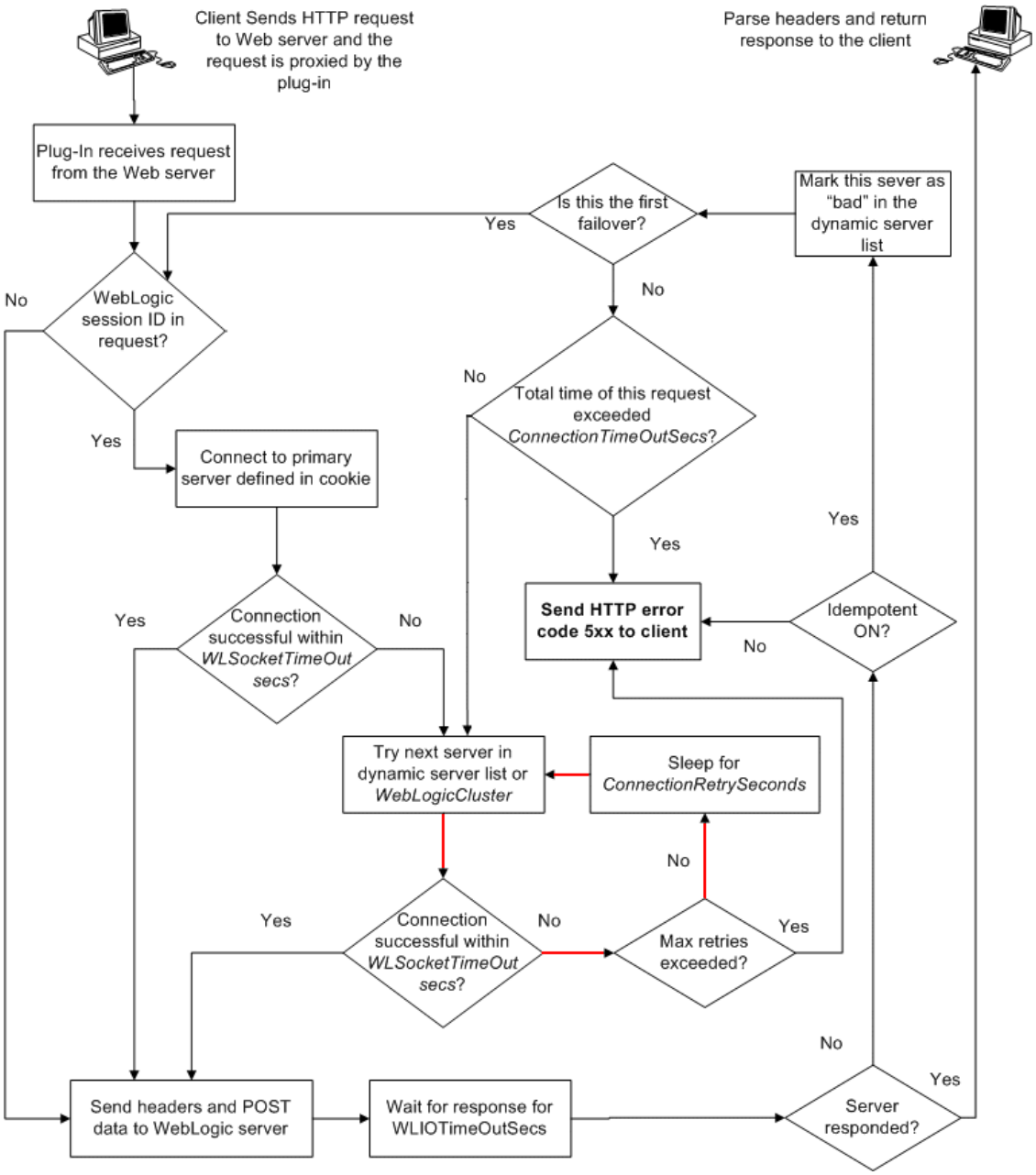
Failover, Cookies, and HTTP Sessions

When a request contains session information stored in a cookie or in the POST data, or encoded in a URL, the session ID contains a reference to the specific server instance in which the session was originally established (called the primary server). A request containing a cookie attempts to connect to the primary server. If that attempt fails, the proxy plug-in attempts to make a connection to the next available server in the list in a round-robin fashion.

That server retrieves the session from the original secondary server and makes itself the new primary server for that same session. See Figure 6-1.

> **Note:**
>
> If the POST data is larger than 64K, the proxy plug-in will not parse the POST data to obtain the session ID. Therefore, if you store the session ID in the POST data, the proxy plug-in cannot route the request to the correct primary or secondary server, resulting in possible loss of session data.

Figure 6-1 Connection Failover



In this figure, the Maximum number of retries allowed in the red loop is equal to `ConnectTimeoutSecs/ConnectRetrySecs`.

Failover Behavior When Using Firewalls and Load Directors

In some configurations that use combinations of firewalls and load-directors, any one of the servers (firewall or load-directors) can accept the request and return a successful connection while the primary instance of Oracle WebLogic Server is unavailable. After attempting to direct the request to the primary instance of Oracle WebLogic Server (which is unavailable), the request is returned to the proxy plug-in as "connection reset."

Requests running through combinations of firewalls (with or without load-directors) are handled by Oracle WebLogic Server. In other words, responses of connection reset fail over to a secondary instance of Oracle WebLogic Server. Because responses of connection reset fail over in these configurations, servlets must be idempotent. Otherwise duplicate processing of transactions may result.

Oracle WebLogic Server Session Issues

The WLS proxy plug-in routes the requests to back-end Oracle WebLogic Server or cluster. Oracle WebLogic Server maintains sessions so that subsequent requests from the same client are routed to the same server. However, due to various reasons, if the WLS proxy plug-in cannot communicate with the Oracle WebLogic Server server, the request is handled in the following ways:

- If the request is routed to a single Oracle WebLogic Server instance, the WLS proxy plug-in continues trying to connect to that same Oracle WebLogic Server instance for the maximum number of retries as specified by the ratio of `ConnectTimeoutSecs` and `ConnectRetrySecs`. If all attempts fail, an HTTP 503 error message is returned back to the client.
- If the request is routed to the WebLogic cluster, the current Oracle WebLogic Server is marked as bad, and the request is routed to the next available Oracle WebLogic Server. If all attempts fail, an HTTP 503 error message is returned back to the client.

In addition to sending a HTTP 503 error message, the following is displayed as a response in the HTTP client:

```
Failure of Web Server bridge:
No backend server available for connection: timed out after xx seconds or idempotent set
to OFF or method not idempotent.
```

NO_RESOURCES Errors

Occasionally, under stress conditions, a few requests might fail with the error logged in the error log file.

The following error is logged in the log file:

```
weblogic: *******Exception type [NO_RESOURCES] (apr_socket_connect call failed with
error=70007, host=xx.xx.xx.xx, port=xxxx) raised at line xxxx of URL.cpp
```

This usually occurs if Oracle WebLogic Server is too busy to respond to the connect request from the WLS proxy plug-in. This can be resolved by setting `WLSocketTimeoutSecs` to a higher value. This allows the WLS proxy plug-in to wait longer for the connect request to be responded to by the Oracle WebLogic Server.

POST Data Files Issues

The temporary POST file is located under `/tmp/_wl_proxy` for UNIX. For Windows it is located as follows (if `WLTempDir` is not specified):

- Environment variable `TMP`
- Environment variable `TEMP`
- `C:\Temp`

The `/tmp/_wl_proxy` is a fixed directory and is owned by the HTTP Server user. When there are multiple HTTP Servers installed by different users, some HTTP Servers might not be able to write to this directory. This condition results in an error.

To correct this condition, use the `WLTempDir` parameter to specify a different location for the `_wl_proxy` directory for POST data files.