Oracle® Fusion Middleware

Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server

11*g* Release 1 (10.3.5)

E16435-03

April 2011

This document explains the use of version 1.1 plug-ins provided for proxying requests to third party administration servers. This document is intended mainly for system administrators who manage the WebLogic Server application platform and its various subsystems.

ORACLE

Oracle Fusion Middleware Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server, 11*g* Release 1 (10.3.5)

E16435-03

Primary Author: Oracle Corporation

Contents

Preface

This preface describes the document accessibility features and conventions used in this guide—*Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server*.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

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.

Convention	Meaning
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 Introduction and Roadmap

This chapter describes the contents and organization of this guide—*Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server*.

1.1 Document Scope and Audience

This document explains use of plug-ins provided for proxying requests to third party administration servers. This document is intended mainly for system administrators who manage the Oracle WebLogic Server application platform and its various subsystems.

1.2 Guide to this Document

This chapter introduces the organization of this guide. The guide is organized as follows:

- Chapter 2, "Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server" describes the plug-ins provided by Oracle for use with WebLogic Server.
- Chapter 3, "Installing and Configuring the Apache HTTP Server Plug-In" describes how to install and configure the Apache HTTP Server plug-in.
- Chapter 4, "Configuring the Plug-In for Oracle HTTP Server" describes how to install and configure the Oracle HTTP Server plug-in.
- Chapter 5, "Installing and Configuring the Microsoft IIS Plug-In" describes how to install and configure the Microsoft Internet Information Server plug-in.
- Chapter 6, "Performing Common Tasks" describe common tasks that you perform for the plug-ins provided by Oracle for use with WebLogic Server.
- Chapter 7, "Parameters for Web Server Plug-Ins" describes the parameters that you use to configure the Apache and Microsoft IIS Web server plug-ins.

1.3 Related Documentation

This document contains information on using Web server plug-ins.

For general information about the available Oracle WebLogic Server documentation, see *Information Roadmap for Oracle WebLogic Server* .

1.4 New and Changed Features in This Release

For a comprehensive listing of the other new Oracle WebLogic Server features introduced in this release, see *What's New in Oracle WebLogic Server*.

2

Using Web Server 1.1 Plug-Ins with Oracle WebLogic Server

The following sections describe the plug-ins provided by Oracle for use with WebLogic Server:

2.1 What Are Plug-Ins?

Plug-ins are small software programs that developers use to extend a WebLogic Server implementation.

The plug-ins allow requests to be proxied from an Apache HTTP Server, Oracle HTTP Server, or Microsoft Internet Information Server (IIS) to WebLogic Server. In this way, plug-ins enable the HTTP server to communicate with applications deployed on the WebLogic Server.

The plug-in enhances an HTTP server installation by allowing WebLogic Server to handle those requests that require dynamic functionality. That is, you typically use a plug-in where the HTTP server serves static pages such as HTML pages, while dynamic pages such as HTTP Servlets or Java Server Pages (JSPs) are served by WebLogic Server.

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 WebLogic Server still appear to be coming from the HTTP server.

In addition, the HTTP-tunneling facility of the WebLogic client-server protocol also operates through the plug-in, providing access to all WebLogic Server services.

2.1.1 Connection Pooling and Keep-Alive

The plug-ins improve performance using a pool of connections from the plug-in to WebLogic Server. The plug-in implements HTTP 1.1 keep-alive connections between the plug-in and WebLogic Server by reusing the same connection for subsequent requests from the same client. If the connection is inactive for more than 20 seconds, (or a user-defined amount of time) the connection is closed. The connection with the client can be reused to connect to the same client at a later time if it has not timed out.

You can disable this feature if desired. For more information, see `KeepAliveEnabled` in Table 7–1.

2.1.2 Proxying Requests

The plug-in proxies requests to WebLogic Server based on a configuration that you specify. You can proxy requests based on either 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 do 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 plug-in.

2.2 Version 1.1 Plug-Ins Available for Download

The version 1.1 plug-ins are not bundled with WebLogic Server but are instead available for download.

See Section 2.6, "Downloading the Version 1.1 Plug-Ins" for instructions on downloading the plug-ins.

The following version 1.1 plug-ins are available for download for use with this release of Oracle WebLogic Server:

- Apache HTTP Server 2.2.x
- Microsoft Internet Information Server (IIS 6.0 and IIS 7.0).

Oracle HTTP Server uses the Apache HTTP Server plug-in, which is bundled with Oracle HTTP Server.

2.2.1 Version 1.0 Plug-Ins Are Deprecated

The version 1.0 plug-ins are deprecated in this release of Oracle WebLogic Server. The version 1.1 plug-ins are the recommended replacement.

The version 1.1 plug-ins are a superset of the version 1.0 plug-ins described in *Using Web Server Plug-Ins with Oracle WebLogic Server* and support all of the existing features, with the exception of the Sun Java System Web Server.

Note: If you need to use a plug-in with Sun Java System Web Server, continue to use the version 1.0 plug-in.

The version 1.1 plug-in supports Apache HTTP Server 2.2.x only. If you need to use Apache1.3.x or Apache 2.0.x, continue to use the version 1.0 plug-in.

As in previous releases, the version 1.0 plug-ins continue to be bundled with Oracle WebLogic Server. However, the version 1.0 plug-ins are not guaranteed to be bundled with future versions of Oracle Weblogic Server. Oracle recommends that you instead download and use the version 1.1 plug-ins as described in Section 2.6, "Downloading the Version 1.1 Plug-Ins".

2.3 Upgrading From the Version 1.0 Plug-Ins

The version 1.1 plug-ins are a superset of the version 1.0 plug-ins described in *Using Web Server Plug-Ins with Oracle WebLogic Server* and support the existing features. However, keep the following considerations in mind when you upgrade:

- The list of supported platforms has changed, as described in Section 2.5, "Plug-In Supported Platforms".
- The version 1.1 plug-ins support most of the existing version 1.0 plug-in features, with the exception of the Sun Java System Web Server. If you need to use a plug-in with Sun Java System Web Server, continue to use the version 1.0 plug-in. If you need to use Apache 1.3.x or Apache 2.0.x, continue to use the version 1.0 plug-in.
- If you have been using 128-bit encryption, you need to change your configuration file to reflect the new naming convention, as described in Section 2.4.1, "Standard Encryption Strength Allows Simplified Naming". For example, you need to change `mod_wl128_22.so` to `mod_wl.so`.

2.4 Features of the Version 1.1 Plug-Ins

This section describes the features of the version 1.1 plug-ins. The following topics are described:

- Section 2.4.1, "Standard Encryption Strength Allows Simplified Naming"
- Section 2.4.2, "Version 1.1 Plug-Ins Use Oracle Security Framework"
- Section 2.4.3, "Version 1.1 Plug-Ins Support IPv6"
- Section 2.4.4, "Version 1.1 Plug-Ins Support Two-Way SSL"

2.4.1 Standard Encryption Strength Allows Simplified Naming

Because the version 1.0 plug-ins supported both 40- and 128-bit encryption standards, the plug-in file names needed to identify which standard was supported. For example, `mod_wl_22.so` indicated 40-bit encryption and `mod_wl128_22.so` indicated 128-bit encryption.

However, the version 1.1 plug-ins support only 128-bit encryption, and the plug-in names are now simplified. For example, `mod_wl.so` is the only file name required.

Note: If you upgrade from the 1.0 plug-ins and had been using 128-bit encryption, you need to change your configuration file to reflect the new naming convention. For example, you need to change `mod_wl128_22.so` to `mod_wl.so`.

2.4.2 Version 1.1 Plug-Ins Use Oracle Security Framework

The version 1.1 plug-ins use the Oracle certified security framework, and can therefore use Oracle wallets to store SSL configuration information.

For this reason, the version 1.1 plug-ins introduce an SSL configuration parameter `WLSSLWallet` to use Oracle wallets.

You can configure the certificates in the Oracle wallet with a command line tool that is provided with the plug-in binary files. See Section 6.1, "Use SSL With Plug-Ins" for information about configuring SSL.

2.4.3 Version 1.1 Plug-Ins Support IPv6

The version 1.1 plug-ins support IPv6. The `WebLogicHost` and `WebLogicCluster` configuration parameters (see Table 7–1) now support IPv6 addresses.

See Section 6.2, "Use IPv6 With Plug-Ins" for additional information.

2.4.4 Version 1.1 Plug-Ins Support Two-Way SSL

The version 1.1 plug-ins provide two-way SSL support for verifying client identity. Two-way SSL is automatically enforced when WebLogic Server requests the client certificate during the handshake process.

See Section 6.1, "Use SSL With Plug-Ins" for configuration information.

2.5 Plug-In Supported Platforms

The version 1.1 plug-ins are supported on the platforms described in `http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html`.

2.6 Downloading the Version 1.1 Plug-Ins

The WebLogic Server version 1.1 plug-ins are available for download via the `http://metalink.oracle.com/` Web site, and also from `http://www.oracle.com/technology/software/products/ias/htdocs/wls_main.html`.

The WebLogic Server 1.1 plug-ins are available in the form of a zip file containing the necessary binary and helper files. You must download and unzip the appropriate file, and then install the plug-in as described in each subsequent plug-in chapter.

For example, the following directories are included in the `mod_wl_so` plug-in distribution. For the Windows version, DLL files are provided.

- `lib/mod_wl.so` or `lib\mod_wl.dll` (Apache plug-in)
- `lib/*.so` or `lib\*.dll` (native libraries)
- `bin/orapki` or `bin\orapki.cmd` (orapki tool)
- `jlib/*.jar` (Java helper libraries for orapki)

3

Installing and Configuring the Apache HTTP Server Plug-In

The following sections describe how to install and configure the Apache HTTP Server Plug-In:

- Section 3.1, "Install the Apache HTTP Server Plug-In"
- Section 3.2, "Configure the Apache HTTP Server Plug-In"

> **Note:** In this release of Oracle WebLogic Server, a single plug-in supports both Apache HTTP Server and Oracle HTTP Server.

3.1 Install the Apache HTTP Server Plug-In

After you have downloaded the Apache HTTP Server Plug-In, as described in Section 2.6, "Downloading the Version 1.1 Plug-Ins", you can install it as an Apache module in your Apache HTTP Server installation and link it as a Dynamic Shared Object (DSO).

A DSO is compiled as a library that is dynamically loaded by the server at runtime, and can be installed without recompiling Apache.

3.1.1 Installation Prerequisites

Before you install the Apache HTTP Server plug-in, you must satisfy the following prerequisites:

- Download the Apache HTTP Server Plug-In, as described in Section 2.6, "Downloading the Version 1.1 Plug-Ins".
- You have extracted the plug-ins zip distribution to the location of your choice on the target system. For example, `/home/myhome/weblogic-plugins-1.1/`.
- Install JDK 6 if you want to use SSL. The JDK 6 installation is required to use the orapki utility. The orapki utility manages public key infrastructure (PKI) elements, such as wallets and certificate revocation lists, for use with SSL.
- You have a supported Apache HTTP Server installation.

  The version 1.1 plug-ins are supported on the Apache platforms described in `http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html`.
- A supported version of WebLogic Server is configured and running on a target system. However, it does not need to be running on the system on which you

extracted the plug-in zip distribution. See http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html for the supported WebLogic Server versions.

3.1.2 Installing the Apache HTTP Server Plug-In as a Dynamic Shared Object

The Apache plug-in is distributed as a shared object (.so) for Unix platforms and a DLL for Windows.

To install the Apache HTTP Server Plug-In as a dynamic shared object:

1. Make sure that the `weblogic-plugins-1.1/lib` folder is included in `LD_LIBRARY_PATH` on Unix systems (and PATH on Windows systems). If you do not do this, you see linkage errors when starting Apache.
2. In the location where you unzipped the downloaded plug-in file, locate `lib/mod_wl.so`, or `lib\mod_wl.dll` for windows . For example, `/home/myhome/weblogic-plugins-1.1/lib/mod_wl.so`.
3. Verify that the `mod_so.c` module is enabled.

   The Apache HTTP Server Plug-In will be installed in your Apache HTTP Server installation as a Dynamic Shared Object (DSO).

   DSO support in Apache is based on module `mod_so.c`, which must be enabled before `mod_wl.so` is loaded.

   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 executing the following command:

   ```
   APACHE_HOME\bin\apachectl -l
   ```

   (Where *APACHE_HOME* is the directory containing your Apache HTTP Server installation.)

   This command lists all enabled modules. If `mod_so.c` is not listed, you must rebuild your Apache HTTP Server, making sure that the following options are configured:

   ```
   ...
   --enable-module=so
   --enable-rule=SHARED_CORE
   ...
   ```

   See Apache 2.2 Shared Object (DSO) Support at http://httpd.apache.org/docs/2.2/dso.html.
4. Make a copy of the ${*APACHE_HOME*}`/bin/httpd.conf` file for backup.
5. Open the `httpd.conf` file.

   The file is located at `APACHE_HOME/conf/httpd.conf` (where *APACHE_HOME* is the root directory of your Apache HTTP server installation). See a sample `httpd.conf` file at Section 3.2, "Configure the Apache HTTP Server Plug-In".
6. Install the Apache HTTP Server Plug-In module for Apache 2.2.x by adding the following line to your `APACHE_HOME/conf/httpd.conf` file. For Windows, specify the .DLL file.

   ```
   LoadModule weblogic_module    /home/myhome/weblogic-plugins-1.1/lib/mod_wl.so
   ```
7. Verify the syntax of the `APACHE_HOME/conf/httpd.conf` file with the following command:

```
APACHE_HOME\bin\apachectl -t
```

(Where *APACHE_HOME* is the directory containing your Apache HTTP Server installation.)

The output of this command reports any errors in your `httpd.conf` file or returns:

```
Syntax OK
```

3.2 Configure the Apache HTTP Server Plug-In

After installing the plug-in in the Apache HTTP Server, configure the WebLogic Server Apache Plug-In and configure the server to use the plug-in.

This section explains how to edit the `httpd.conf` file to proxy requests by path or by MIME type, to enable HTTP tunneling, and to use other WebLogic Server plug-in parameters.

3.2.1 Editing the httpd.conf File

Edit the `httpd.conf` file in your Apache HTTP server installation to configure the Apache HTTP Server Plug-In.

1. Make a copy of the ${*APACHE_HOME*}`/bin/httpd.conf` file for backup.
2. Open the `httpd.conf` file.

   The file is located at `APACHE_HOME/conf/httpd.conf` (where *APACHE_HOME* is the root directory of your Apache HTTP server installation). See a sample `httpd.conf` file at Section 3.2, "Configure the Apache HTTP Server Plug-In".
3. Ensure that the WebLogic Server modules are included for Apache 2.2.x. Add the following line to the `httpd.conf` file if you have not already done so. For Windows, specify the .DLL file.

   ```
   LoadModule weblogic_module   /home/myhome/weblogic-plugins-1.1/lib/mod_wl.so
   ```
4. To proxy requests by MIME type, add an `IfModule` block that defines one of the following:
   - For a non-clustered WebLogic Server: the `WebLogicHost` and `WebLogicPort` parameters.
   - For a cluster of WebLogic Servers: the `WebLogicCluster` parameter.

   For example:

   ```
   <IfModule mod_weblogic.c>
     WebLogicHost my-weblogic-server-com
     WebLogicPort 7001
     Debug ALL
     DebugConfigInfo ON
     WLLogFile /tmp/wl-proxy.log
   </IfModule>
   ```
5. To proxy requests by MIME type, add a `MatchExpression` line to the `IfModule` block. Note that if both MIME type and proxying by path are enabled, proxying by path takes precedence over proxying by MIME type.

   For example, the following `IfModule` block for a non-clustered 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
  Debug ALL
  DebugConfigInfo ON
  WLLogFile /tmp/wl-proxy.log
</IfModule>
```

You can also use multiple `MatchExpressions`, for example:

```
<IfModule mod_weblogic.c>
  WebLogicHost my-weblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
   MatchExpression *.xyz
  Debug ALL
  DebugConfigInfo ON
  WLLogFile /tmp/wl-proxy.log
</IfModule>
```

If you are proxying requests by MIME type to a cluster of WebLogic Servers, use the `WebLogicCluster` parameter instead of the `WebLogicHost` and `WebLogicPort` parameters. For example:

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

6. To proxy requests by path, use the `Location` block and the `SetHandler` statement. `SetHandler` specifies the handler for the Apache HTTP Server Plug-In module. For example the following `Location` block proxies all requests containing /weblogic in the URL:

```
<Location /weblogic>
  SetHandler weblogic-handler
PathTrim /weblogic
</Location>
```

The `PathTrim` parameter specifies a string trimmed from the beginning of the URL before the request is passed to the WebLogic Server instance (see Section 7.2, "General Parameters for Web Server Plug-Ins").

7. Optionally, enable HTTP tunneling for t3 or IIOP.

   a. 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/HTTPClnt>
     SetHandler weblogic-handler
   </Location>
   ```

   b. To enable HTTP tunneling if you are using the IIOP, the only protocol used by the WebLogic Server thin client, `wlclient.jar`, add the following `Location` block to the `httpd.conf` file:

   ```
   <Location /bea_wls_internal/iiop>
     SetHandler weblogic-handler
   </Location>
   ```

8. Define any additional parameters for the Apache HTTP Server Plug-In.

   The Apache HTTP Server Plug-In recognizes the parameters listed in Section 7.2, "General Parameters for Web Server Plug-Ins". To modify the behavior of your Apache HTTP Server Plug-In, define these parameters either:

   - In a `Location` block, for parameters that apply to proxying by path, or
   - In an `IfModule` block, for parameters that apply to proxying by MIME type.

9. Verify the syntax of the *`APACHE_HOME/conf/httpd.conf`* file with the following command:

   ```
   APACHE_HOME\bin\apachectl -t
   ```
   (Where *`APACHE_HOME`* is the directory containing your Apache HTTP Server installation.)

   The output of this command reports any errors in your `httpd.conf` file or returns:

   ```
   Syntax OK
   ```

10. Start the Apache HTTP Server.

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

    (Where *`APACHE_HOME`* is the directory containing your Apache HTTP Server installation.)

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

3.2.1.1 Placing WebLogic Properties Inside Location or VirtualHost Blocks

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

```
<Location /weblogic>
SetHandler weblogic-handler
WebLogicHost myweblogic.server.com
WebLogicPort 7001
</Location>

<Location /weblogic>
SetHandler weblogic-handler
WebLogicCluster w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
</Location>

<VirtualHost apachehost:80>
SetHandler weblogic-handler
WebLogicServer weblogic.server.com
WebLogicPort 7001
</VirtualHost>
```

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

If you want to keep several separate configuration files, you can define parameters in a separate configuration file called `weblogic.conf` file, by using the Apache Include directive in an `IfModule` block in the `httpd.conf` file:

```
<IfModule mod_weblogic.c>
  # Config file for 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.

This section describes how to create `weblogic.conf` files, and includes sample `weblogic.conf` files.

3.2.2.1 Creating weblogic.conf Files

Be aware of the following when 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/`).
- If you want to have only one log file for all the virtual hosts configured in your environment, you can achieve it using global properties. Instead of specifying the same `Debug`, `WLLogFile` and `WLTempDir` properties in each virtual host you can specify them just once in the `<IfModule>` tag.
- Sample `httpd.conf` file:

  ```
  <IfModule mod_weblogic.c>
    WebLogicCluster johndoe02:8005,johndoe:8006
    Debug ON
    WLLogFile             c:/tmp/global_proxy.log
    WLTempDir             "c:/myTemp"
    DebugConfigInfo       On
    KeepAliveEnabled ON
    KeepAliveSecs  15
  </IfModule>

  <Location /jurl>
    SetHandler weblogic-handler
    WebLogicCluster agarwalp01:7001
  </Location>

  <Location /web>
    SetHandler weblogic-handler
    PathTrim/web
    Debug OFF
    WLLogFile c:/tmp/web_log.log
  </Location>

  <Location /foo>
    SetHandler weblogic-handler
    PathTrim/foo
    Debug ERR
    WLLogFile c:/tmp/foo_proxy.log
  ```

```
</Location>
```

- All the requests which match /jurl/* will have Debug Level set to ALL and log messages will be logged to `c:/tmp/global_proxy.log` file. All the requests which match /web/* will have Debug Level set to OFF and no log messages will be logged. All the requests which match /foo/* will have Debug Level set to ERR and log messages will be logged to `c:/tmp/foo_proxy.log` file.
- Oracle recommends that you use the `MatchExpression` statement instead of the `<Files>` block.

3.2.2.2 Sample weblogic.conf Configuration Files

The following 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 Example 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. (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

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

In Example 3–2, the `MatchExpression` parameter syntax for expressing the filename pattern, the WebLogic Server host to which HTTP requests should be forwarded, and various other parameters is 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 WebLogic Server is listening for connection requests, and also activate the Debug option. The second `MatchExpression` specifies the filename pattern *.http and identifies the WebLogicCluster hosts and their ports. The `paramName=value` combination following the pipe symbol specifies the error page for the cluster.

Example 3–2 Example 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 (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

<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 shows an example without WebLogic clusters.

Example 3–3 Example 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 (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)
<IfModule mod_weblogic.c>
  WebLogicHost myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

Example 3–4 shows an example of configuring multiple name-based virtual hosts.

Example 3–4 Example Configuring Multiple Name-Based Virtual Hosts

```
# VirtualHost1 = localhost:80
<VirtualHost 127.0.0.1:80>
DocumentRoot "C:/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 "C:/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 Plug-In parameters will not work as expected.

3.2.2.3 Template for the Apache HTTP Server httpd.conf File

This section contains a sample `httpd.conf` file for Apache 2.2. You can use this sample as a template and modify it to suit your environment and server. Lines beginning with # are comments.

Note that Apache HTTP Server is not case sensitive.

Example 3–5 Sample httpd.conf file for Apache 2.2

```
##################################################
APACHE-HOME/conf/httpd.conf file
##################################################
LoadModule weblogic_module   lhome/myhome/weblogic-plugins-1.1/lib/mod_wl.so
```

```
<Location /weblogic>
 SetHandler weblogic-handler
 PathTrim /weblogic
 ErrorPage http://myerrorpage1.mydomain.com
</Location>

<Location /servletimages>
 SetHandler weblogic-handler
 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>
```

4

Configuring the Plug-In for Oracle HTTP Server

In this release of Oracle WebLogic Server, a single plug-in supports both Apache HTTP Server and Oracle HTTP Server.

You do not have to download and set up the plug-in. Oracle HTTP Server comes pre-bundled with the `mod_wl_ohs.so/dll` binary. This plug-in, although uniquely named, is identical to the Apache plug-in.

The following section describes how to configure the plug-in for Oracle HTTP Server:

- Section 4.1, "Configuring the Plug-In for Oracle HTTP Server"

> **Note:** The definitive source of information for configuring this plug-in is *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*, and specifically the section titled "Configure the mod_wl_ohs Module on Oracle HTTP Server". The information is summarized here for your convenience.

4.1 Configuring the Plug-In for Oracle HTTP Server

To configure the `mod_wl_ohs` module using Fusion Middleware Control, do the following:

1. Select Administration from the Oracle HTTP Server menu.
2. Select `mod_wl_ohs` Configuration from the Administration menu. The `mod_wl_ohs` configuration page appears.
3. If you are using a WebLogic cluster, enter the WebLogic Servers that can be used for load balancing in the WebLogic Cluster field. 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.

   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. 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.

4. Use the WebLogic Host field to enter the WebLogic Server host (or virtual host name as defined in WebLogic Server) to which HTTP requests should be forwarded. If you are using a WebLogic cluster, use the WebLogic Cluster field instead of WebLogic Host.
5. Use the WebLogic Port field to enter the port on which the WebLogic Server host is listening for connection requests from the module (or from other servers). (If you are using SSL between the module and WebLogic Server, set this parameter to the SSL listen port.
6. If you want to use the dynamic cluster list for load balancing requests proxied from the module, then select the Dynamic Server List ON check box. When set to OFF, the module ignores the dynamic cluster list and only uses the static list specified with the WebLogic Cluster parameter. Normally this parameter should be set to ON.
7. You can use the Error Page field to create your own error page that is displayed when your Web server is unable to forward requests to WebLogic Server.
8. Use the Debug field to specify the type of logging performed for debugging operations. The debugging information is written to the /tmp/wlproxy.log file on UNIX systems and c:\TEMP\wlproxy.log on Windows systems. Override this location and filename by setting the Log File parameter to a different directory and file. Ensure that the tmp or TEMP directory has write permission assigned to the user who is logged in to the server.

   The Debug parameter can be set any of the following logging options. Additionally, the HFC, HTW, HFW, and HTC options can be set in combination by entering them separated by commas; for example: HFC,HTW.

   - ON – The module logs informational and error messages.
   - OFF – No debugging information is logged.
   - HFC – The module logs headers from the client, informational, and error messages.
   - HTW – The module logs headers sent to WebLogic Server, and informational and error messages.
   - HFW – The module logs headers sent from WebLogic Server, and informational and error messages.
   - HTC – The module logs headers sent to the client, informational messages, and error messages.
   - ERR – Prints only the Error messages in the module.
   - ALL – The module logs headers sent to and from the client, headers sent to and from WebLogic Server, information messages, and error messages.
9. Use the Log File field to specify the path and file name for the log file that is generated when the Debug parameter is set to ON. You must create this directory before setting this parameter.
10. Use the WebLogic Temp Directory field to specify the directory where a wlproxy.log will be created. If the location fails, the module resorts to creating the log file under c:/temp in Windows and /tmp in all UNIX platforms.

    This also specifies the location of the _wl_proxy directory for post data files. When both WebLogic Temp Directory and Log File are set, Log File will override as to the location of wlproxy.log. WebLogic Temp Directory will still determine the location of the _wl_proxy directory.

11. Use the Exclude Path or Mime Type field to exclude certain requests from proxying. This parameter can be defined locally at the Location tag level as well as globally. When the property is defined locally, it does not override the global property but defines a union of the two parameters.
12. The Match Expression region is used to specify any Expression overrides.

    Example when proxying by MIME type:

    ```
    *.jsp WebLogicHost=myHost|paramName=value
    ```

    It is possible to define a new parameter for Match Expression using the following syntax:

    ```
    *.jsp PathPrepend=/test PathTrim=/foo
    ```

13. The Location region is used to specify any Location overrides.
    a. Click Add Row to create a new row.
    b. Enter the base URI for which following directives become effective.
    c. Complete the WebLogic Cluster, WebLogic Host, and WebLogic Port fields using the definitions supplied earlier in this section.
    d. For the Path Trim field, as per 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 that if you are newly converting an existing third-party server to proxy requests to WebLogic Server using the module, you will need to change application paths to /foo to include weblogic/foo. You can use Path Trim and Path Prepend in combination to change this path.
    e. For the Path Prepend field, as per 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 that if you need to append File Name, use the DefaultFileName module parameter instead of Path Prepend.
    f. Complete the Log File and Debug fields using the definitions supplied earlier in this section.
    g. Click Add Row again to save the new row.

14. Review the settings. If the settings are correct, click Apply to apply the changes. If the settings are incorrect, or you decide to not apply the changes, click Revert to return to the original settings.
15. Restart Oracle HTTP Server.

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

Note: If you are manually editing the `mod_wl_ohs` configuration settings instead of using Fusion Middleware Control, then all directives should be defined within the defined within the `<IfModule weblogic_module>` block of the `mod_wl_ohs.conf` file. `mod_wl_ohs` will continue to work if directives are defined outside of this block, but this could put the `mod_wl_ohs` Configuration page in Fusion Middleware Control in an inconsistent state.

5

Installing and Configuring the Microsoft IIS Plug-In

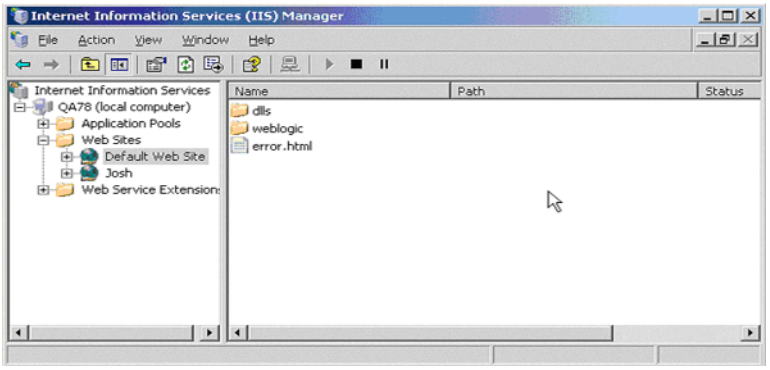
The following sections describe how to install and configure the Microsoft Internet Information Server Plug-In:

5.1 Installing and Configuring the Microsoft Internet Information Server Plug-In

To install the Microsoft Internet Information Server Plug-In:

1. Download the Microsoft Internet Information Server Plug-In, as described in Section 2.6, "Downloading the Version 1.1 Plug-Ins".
2. Copy the iisproxy.dll file into a convenient directory that is accessible to IIS). This directory must also contain the `iisproxy.ini` file that you will create in step 6.
3. Set the user permissions for the `iisproxy.dll` file to include the name of the user who will be running IIS. One way to do this is by right clicking on the `iisproxy.dll` file and selecting Permissions, then adding the username of the person who will be running IIS.
4. If you want to configure proxying by file extension (MIME type) complete this step. (You can configure proxying by path in addition to or instead of configuring by MIME type. See step 5.)
   a. Start the Internet Information Service Manager by selecting it from the Start menu.
   b. In the left panel of the Service Manager, select your Web site (the default is “Default Web Site”).

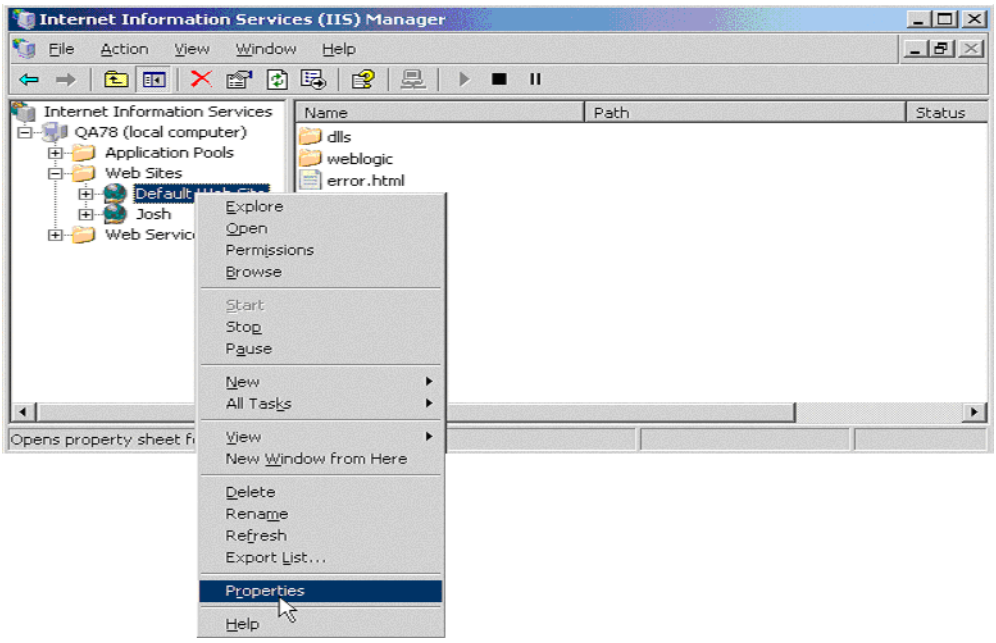
Figure 5–1 Selecting Web Site in Service Manager



c. Click the "Play" arrow in the toolbar to start.

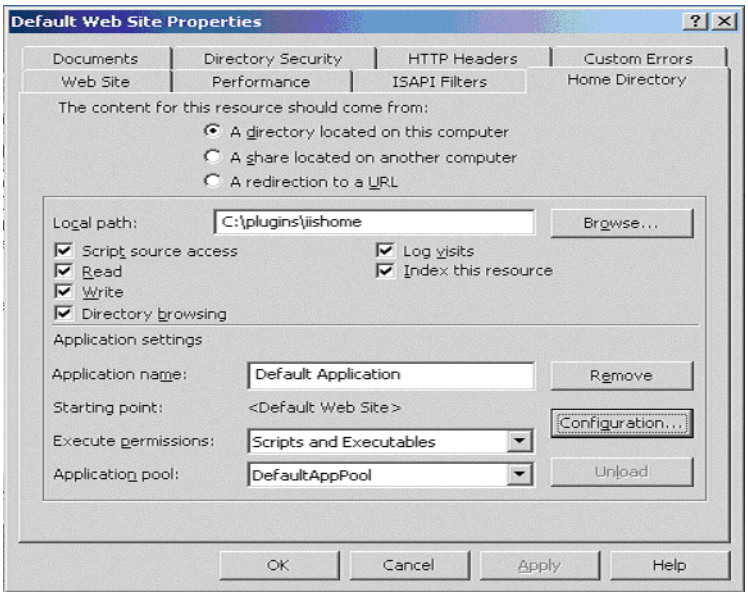
d. Open the properties for the selected Web site by right-clicking the Web site selection in the left panel and selecting Properties.

Figure 5–2 Selecting Properties for Selected Web Site



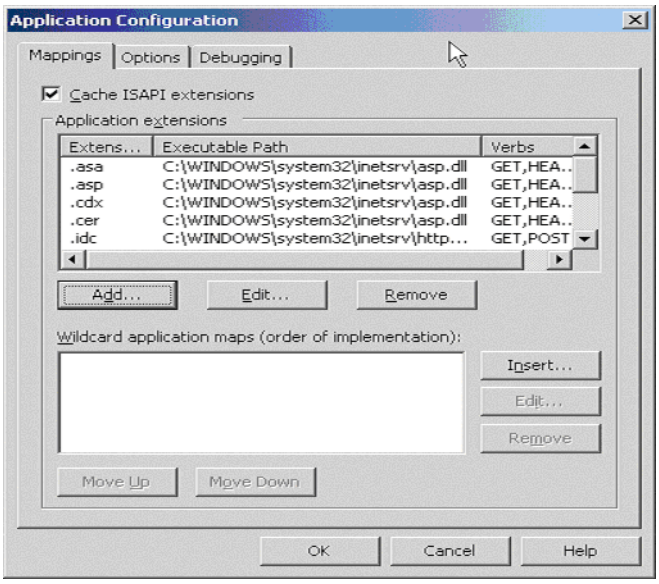
e. In the Properties panel, select the Home Directory tab, and click the Configuration button in the Applications Settings section.

Figure 5–3 Home Directory Tab of the Properties Panel



f. On the Mappings tab, click the Add button to add file types and configure them to be proxied to WebLogic Server.

Figure 5–4 Click the Add Button to Add File Types



g. In the Add dialog box, browse to find the `iisproxy.dll` file.

h. Set the Extension to the type of file that you want to proxy to WebLogic Server.

i. If you are configuring for IIS 6.0 or later, be sure to deselect the "Check that file exists" check box. The behavior of this check has changed from earlier versions of IIS: it used to check that the iisproxy.dll file exists; now it checks that files requested from the proxy exist in the root directory of the Web server. If the check does not find the files there, the iisproxy.dll file will not be allowed to proxy requests to the WebLogic Server.

j. In the Directory Security tab, set the Method exclusions as needed to create a secure installation.

k. When you finish, click the OK button to save the configuration. Repeat this process for each file type you want to proxy to WebLogic.

l. When you finish configuring file types, click the OK button to close the Properties panel.

Note: In the URL, any path information you add after the server and port is passed directly to WebLogic Server. For example, if you request a file from IIS with the URL:

http://myiis.com/jspfiles/myfile.jsp

it is proxied to WebLogic Server with a URL such as http://mywebLogic:7001/jspfiles/myfile.jsp

---

Note: To avoid out-of-process errors, do not deselect the "Cache ISAPI Applications" check box.

---

5. If you want to configure proxying by path, see Section 5.3, "Using Wildcard Application Mappings to Proxy by Path".

6. In WebLogic Server, create the `iisproxy.ini` file.

   The `iisproxy.ini` file contains `name=value` pairs that define configuration parameters for the plug-in. The parameters are listed in Section 7–1, " General Parameters for Web Server Plug-Ins".

   Use the example `iisproxy.ini` file in Section 5.4.1, "Sample iisproxy.ini File" as a template for your `iisproxy.ini` file.

   Note: Changes in the parameters will not go into effect until you restart the “IIS Admin Service” (under services, in the control panel).

   Oracle recommends that you locate the `iisproxy.ini` file in the same directory that contains the `iisproxy.dll` file. You can also use other locations. If you place the file elsewhere, note that WebLogic Server searches for `iisproxy.ini` in the following directories, in the following order:

   a. In the same directory where `iisproxy.dll` is located.

   b. In the home directory of the most recent version of WebLogic Server that is referenced in the Windows Registry. (If WebLogic Server does not find the `iisproxy.ini` file in the home directory, it continues looking in the Windows Registry for older versions of WebLogic Server and looks for the `iisproxy.ini` file in the home directories of those installations.)

   c. In the directory `c:\weblogic`, if it exists.

7. Define the WebLogic Server host and port number to which the Microsoft Internet Information Server Plug-In proxies requests. Depending on your configuration, there are two ways to define the host and port:

- If you are proxying requests to a single WebLogic Server, define the WebLogicHost and WebLogicPort parameters in the `iisproxy.ini` file. For example:

  ```
  WebLogicHost=localhost
  WebLogicPort=7001
  ```

- If you are proxying requests to a cluster of WebLogic Servers, define the WebLogicCluster parameter in the `iisproxy.ini` file. For example:

  ```
  WebLogicCluster=myweblogic.com:7001,yourweblogic.com:7001
  ```

  Where myweblogic.com and yourweblogic.com are instances of Weblogic Server running in a cluster.

8. Optionally, enable HTTP tunneling by following the instructions for proxying by path (see Section 5.3, "Using Wildcard Application Mappings to Proxy by Path") substituting the WebLogic Server host name and the WebLogic Server port number, or the name of a WebLogic Cluster that you wish to handle HTTP tunneling requests.
9. Set any additional parameters in the `iisproxy.ini` file. A complete list of parameters is available in the appendix Section 7.2, "General Parameters for Web Server Plug-Ins".
10. If you are proxying servlets from IIS to WebLogic Server and you are not proxying by path, read the section Section 5.6, "Proxying Servlets from IIS to WebLogic Server".
11. The installed version of IIS with its initial settings does not allow the `iisproxy.dll`. Use the IIS Manager console to enable the Plug-In:
    a. Open the IIS Manager console.
    b. Select Web Service Extensions.
    c. Set "All Unknown ISAPI Extensions" to Allowed.

5.2 Installing and Configuring the Microsoft Internet Information Server Plug-In for IIs 7.0

This section describes differences in how you set up the Microsoft Internet Information Server Plug-In for IIs 7.0.

To set up the Microsoft Internet Information Server Plug-In for IIs 7.0, follow these steps:

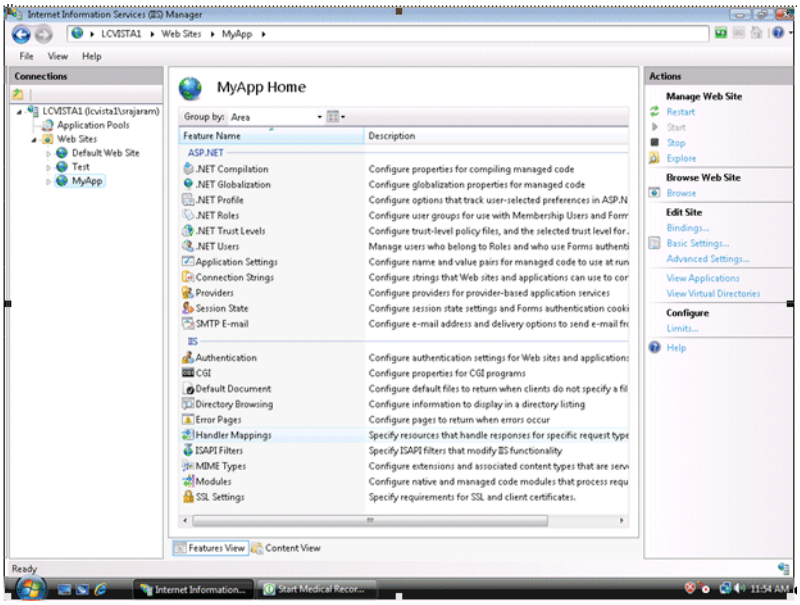
1. Create a web application in IIS Manager by right clicking on Web Sites -> Add Web Site.

   Fill in the Web Site Name with the name you want to give to your web application; for example, `MyApp`. Select the physical path of your web application Port (any valid port number not currently in use).

   Click OK to create the web application.

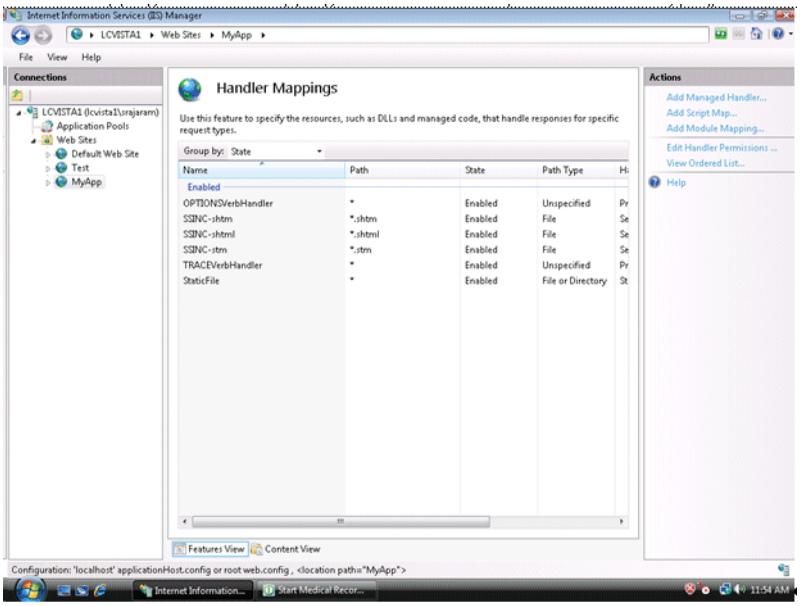
   If you can see the name of your application under Web Sites it means that your application has been created and started running. Click on the MyApp node under Web Sites to see all of the settings related to the MyApp application, which you can change, as shown in Figure 5–5.

Figure 5–5 Application Home Page



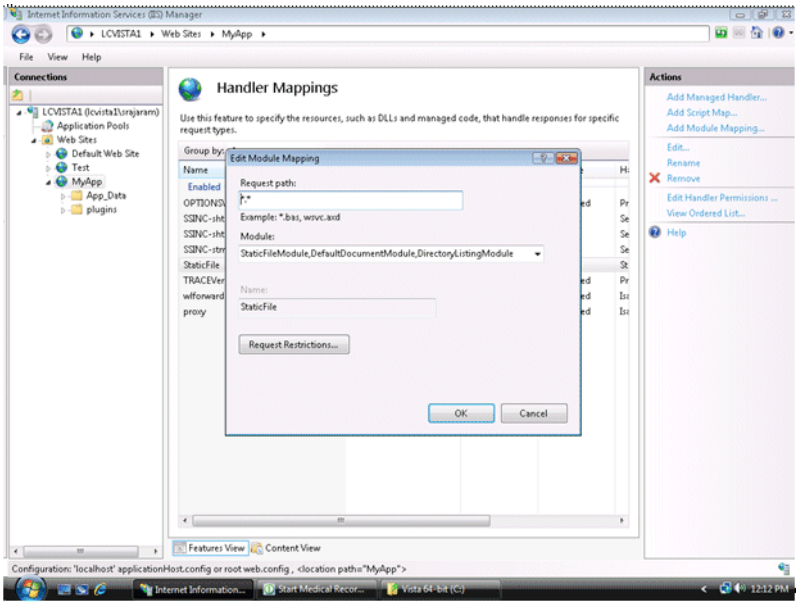
2. Click on "Handler Mappings" to set the mappings to the handler for a particular MIME type.

Figure 5–6 Setting the Handler Mappings



3. Click on the StaticFile and change the Request path from * to *.*. Click OK.

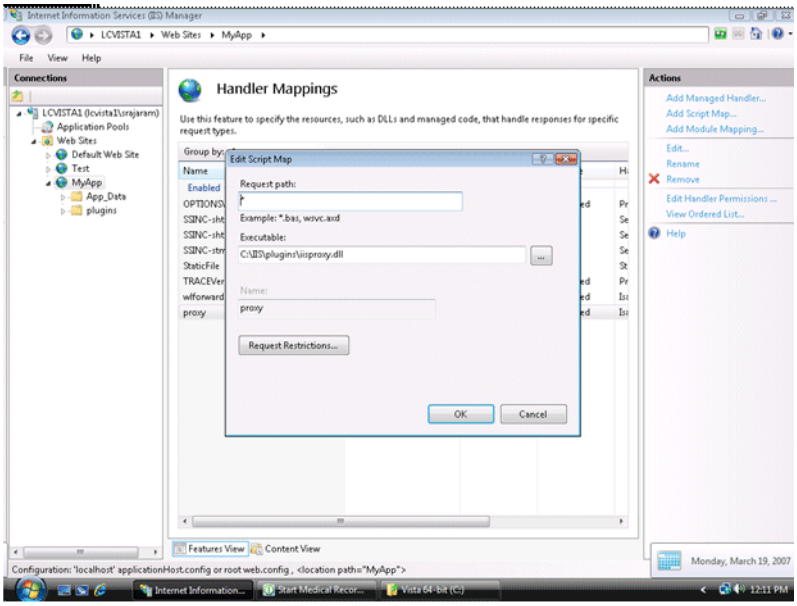
Figure 5–7 Editing the Request Path for Module



4. Click on *MyApp* and then click on "Add Script Map…" on the right-hand side menu options. Enter * for the Request path.

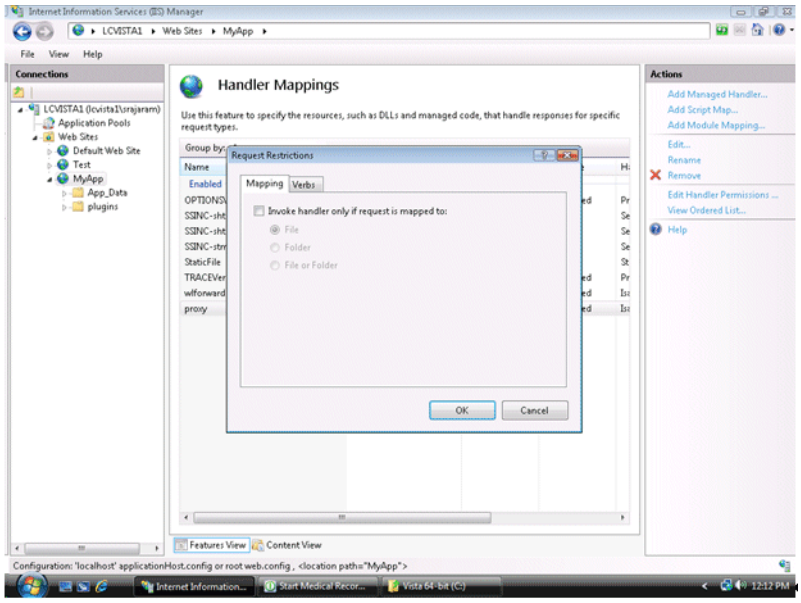
   Browse to the `iisproxy.dll` file and add it as the executable. Name it `proxy`.

Figure 5–8 Editing the Request Path for Script



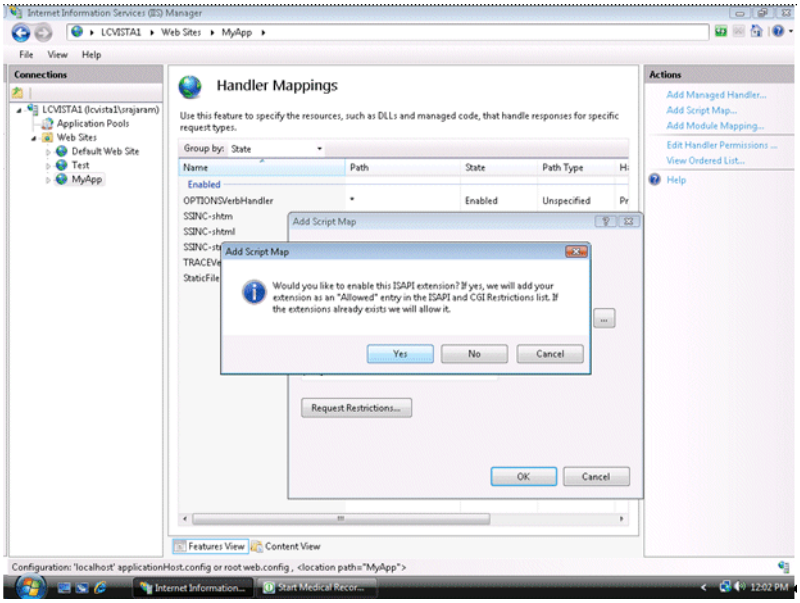
5. Click on the "Request Restrictions…" button and uncheck the box "Invoke handler only if the request is mapped to".

Figure 5–9 Editing the Request Restrictions



6. Click OK to add this Handler mapping. Click Yes on the Add Script Map dialog box.

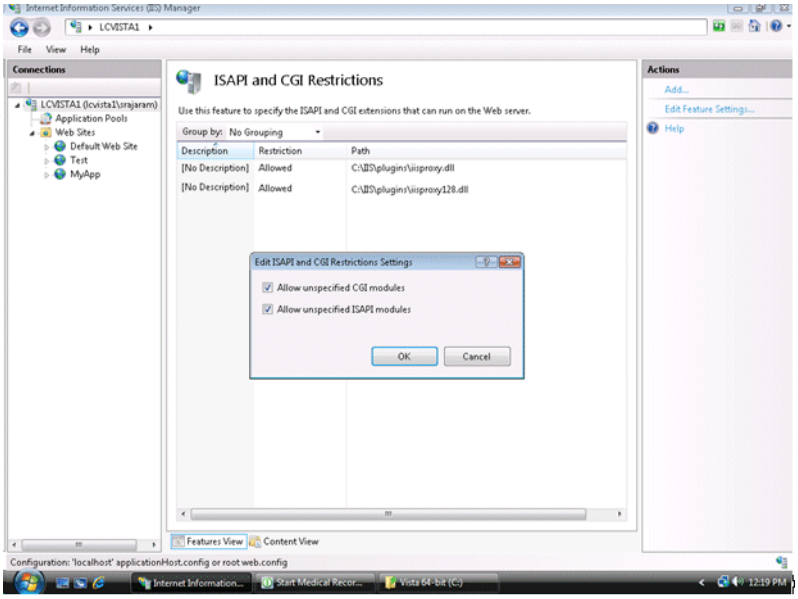
Figure 5–10 Adding the Script Map



7. If you want to configure proxying by path, see Section 5.3, "Using Wildcard Application Mappings to Proxy by Path".

8. Click on the Root node of the IIS Manager tree and click on the ISAPI and CGI Restrictions. Make sure to check the "Allow unspecified ISAPI modules" checkbox.

Figure 5–11 Editing ISAPI and CGI Restrictions



9. Create a file called `iisproxy.ini` with the following contents and place it in the directory with the plug-in:

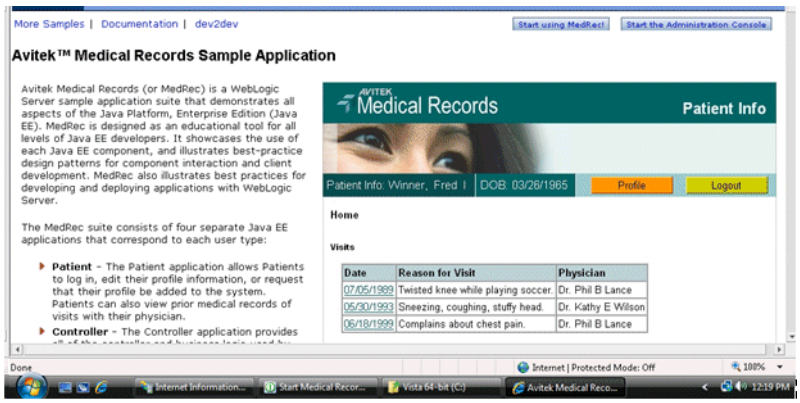
```
WebLogicHost= @hostname@
WebLogicPort= @port@
ConnectRetrySecs=5
ConnectTimeoutSecs=25
Debug=ALL
DebugConfigInfo=ON
KeepAliveEnabled=true

WLLogFile=@Log file name@
SecureProxy=OFF
```

10. Open the Internet Explorer browser and enter `http://<hostname>:<port>`. You should be able to see the Medrec Sample Application from your Weblogic Server.

If you want to run the plug-in in SSL mode, change the value of `WeblogicPort` to the SSL port of your application, and change the `SecureProxy` value to ON.

Figure 5–12 Medrec Sample Application



5.3 Using Wildcard Application Mappings to Proxy by Path

As described in "Installing Wildcard Application Mappings (IIS 6.0)" (http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/5c5ae5e0-f4f9-44b0-a743-f4c3a5ff68ec.mspx?mfr=true), and "Add a Wildcard Script Map" for IIS 7.0 (http://technet.microsoft.com/en-us/library/cc754606(WS.10).aspx), you can configure a Web site or virtual directory to run an Internet Server API (ISAPI) application at the beginning of every request to that Web site or virtual directory, regardless of the extension of the requested file. You can use this feature to insert a mapping to `iisproxy.dll` and thereby proxy requests by path to WebLogic Server.

5.3.1 Installing Wildcard Application Mappings (IIS 6.0)

The following steps summarize the instructions available at "Installing Wildcard Application Mappings (IIS 6.0)" (http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/5c5ae5e0-f4f9-44b0-a743-f4c3a5ff68ec.mspx?mfr=true) for adding a wildcard application mapping to a Web server or Web site in IIS 6.0:

1. In IIS Manager, expand the local computer, expand the Web Sites folder, right-click the Web site or virtual directory that you want, and then click Properties.
2. Click the appropriate tab: Home Directory, Virtual Directory, or Directory.
3. In the Application settings area, click Configuration, and then click the Mappings tab.
4. To install a wildcard application map, do the following:
   a. On the Mappings tab, click Insert.
   b. Type the path to the `iisproxy.dll` DLL in the Executable text box or click Browse to navigate to.
   c. Click OK.

5.3.2 Adding a Wildcard Script Map for IIS 7.0

The following steps summarize the instructions available at "Add a Wildcard Script Map" for IIS 7.0 (http://technet.microsoft.com/en-us/library/cc754606(WS.10).aspx) to add a wildcard script map to do proxy-by-path with ISAPI in IIS 7.0:

1. Open IIS Manager and navigate to the level you want to manage. For information about opening IIS Manager, see "Open IIS Manager" at http://technet.microsoft.com/en-us/library/cc770472(WS.10).aspx. For information about navigating to locations in the UI, see "Navigation in IIS Manager" at http://technet.microsoft.com/en-us/library/cc732920(WS.10).aspx.
2. In Features View, on the server, site, or application Home page, double-click Handler Mappings.
3. On the Handler Mappings page, in the Actions pane, click Add Wildcard Script Map.

4. In the Executable box, type the full path or browse to the `iisproxy.dll` that processes the request. For example, type `systemroot\system32\inetsrv\iisproxy.dll`.
5. In the Name box, type a friendly name for the handler mapping.
6. Click OK.
7. Optionally, on the Handler Mappings page, select a handler to lock or unlock it. When you lock a handler mapping, it cannot be overridden at lower levels in the configuration. Select a handler mapping in the list, and then in the Actions pane, click Lock or Unlock.
8. After you add a wildcard script map, you must add the executable to the ISAPI and CGI Restrictions list to enable it to run. For more information about ISAPI and CGI restrictions, see "Configuring ISAPI and CGI Restrictions in IIS 7" at http://technet.microsoft.com/en-us/library/cc730912(WS.10).aspx.

5.4 Proxying Requests from Multiple Virtual Web Sites to WebLogic Server

To proxy requests from multiple Web sites (defined as virtual directories in IIS) to WebLogic Server:

1. Create a new directory for the virtual directories. This directory will contain `.dll` and `.ini` files used to define the proxy.
2. Extract the contents of the plug-in .zip file to a directory.
3. For each virtual directory you configured, copy the contents of the plug-in `\lib` folder to the directory you created in step 1.
4. Create an `iisproxy.ini` file for the virtual Web sites, as described in Section 2.1.2, "Proxying Requests". Copy this `iispoxy.ini` file to the directory you created in step 1.
5. Copy `iisproxy.dll` to the directory you created in step 1.
6. Create a separate application pool for each virtual directory.

   As described in "Creating Application Pools (IIS 6.)" (http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/93275ef2-2f85-4eb1-8b92-a67545be11b4.mspx?mfr=true), you can isolate different Web applications or Web sites in pools, which are called application pools. In an application pool, process boundaries separate each worker process from other worker processes so that when an application is routed to one application pool, applications in other application pools do not affect that application.

5.4.1 Sample iisproxy.ini File

Here is a sample `iisproxy.ini` file for use with a single, non-clustered WebLogic Server. Comment lines are denoted with the "#" character.

```
# This file contains initialization name/value pairs
# for the IIS/WebLogic plug-in.
WebLogicHost=localhost
WebLogicPort=7001
ConnectTimeoutSecs=20
ConnectRetrySecs=2
```

Here is a sample `iisproxy.ini` file with clustered WebLogic Servers. Comment lines are denoted with the "#" character.

```
# This file contains initialization name/value pairs
# for the IIS/WebLogic plug-in.
WebLogicCluster=myweblogic.com:7001,yourweblogic.com:7001
ConnectTimeoutSecs=20
ConnectRetrySecs=2
```

Note: If you are using SSL between the plug-in and WebLogic Server, the port number should be defined as the SSL listen port.

5.5 Creating ACLs Through IIS

ACLs will not work through the Microsoft Internet Information Server Plug-In if the Authorization header is not passed by IIS. Use the following information to ensure that the Authorization header is passed by IIS.

When using Basic Authentication, the user is logged on with local log-on rights. To enable the use of Basic Authentication, grant each user account the Log On Locally user right on the IIS server. Two problems may result from Basic Authentication's use of local logon:

- If the user does not have local logon rights, Basic Authentication does not work even if the FrontPage, IIS, and Windows NT configurations appear to be correct.
- A user who has local log-on rights and who can obtain physical access to the host computer running IIS will be permitted to start an interactive session at the console.

To enable Basic Authentication, in the Directory Security tab of the console, ensure that the Allow Anonymous option is "on" and all other options are "off".

5.6 Proxying Servlets from IIS to WebLogic Server

You can proxy servlets by path if the `iisforward.dll` is registered as a filter. You would then invoke your servlet with a URL similar to the following:

```
http://IISserver/weblogic/myServlet
```

To proxy servlets if `iisforward.dll` is not registered as a filter, you must configure servlet proxying by file type.To proxy servlets by file type:

1. Register an arbitrary file type (extension) with IIS to proxy the request to the WebLogic Server, as described in step 2 under Section 5.1, "Installing and Configuring the Microsoft Internet Information Server Plug-In".
2. Register your servlet in the appropriate Web Application. For more information on registering servlets, see Creating and Configuring Servlets.
3. Invoke your servlet with a URL formed according to this pattern:

   ```
   http://www.myserver.com/virtualName/anyfile.ext
   ```

   where virtualName is the URL pattern defined in the `<servlet-mapping>` element of the Web Application deployment descriptor (web.xml) for this servlet and ext is a file type (extension) registered with IIS for proxying to WebLogic Server. The anyfile part of the URL is ignored in this context.

Note: If the image links called from the servlet are part of the Web Application, you must also proxy the requests for the images to WebLogic Server by registering the appropriate file types (probably .gif and .jpg) with IIS. You can, however, choose to serve these images directly from IIS if desired.

If the servlet being proxied has links that call other servlets, then these links must also be proxied to WebLogic Server, conforming to the pattern described in step 3.

5.7 Testing the Installation

After you install and configure the Microsoft Internet Information Server Plug-In, follow these steps for deployment and testing:

1. Make sure WebLogic Server and IIS are running.
2. Save a JSP file into the document root of the default Web Application.
3. Open a browser and set the URL to the IIS plus filename.jsp, as shown in this example:

   ```
   http://myii.server.com/filename.jsp
   ```

   If filename.jsp is displayed in your browser, the plug-in is functioning.

6 Performing Common Tasks

The following sections describe common tasks that you perform for the plug-ins provided by Oracle for use with WebLogic Server:

- Section 6.1, "Use SSL With Plug-Ins"
- Section 6.2, "Use IPv6 With Plug-Ins"
- Section 6.3, "Set Up Perimeter Authentication"
- Section 6.4, "Set the WebLogic Plug-in Enabled Control in WebLogic Server"
- Section 6.5, "Understanding Connection Errors and Clustering Failover"

6.1 Use SSL With Plug-Ins

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between the plug-in and WebLogic Server. The SSL protocol provides confidentiality and integrity to the data passed between the plug-in and WebLogic Server.

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

Instead, the plug-in uses SSL parameters that you configure for the plug-in, as described in Section 7.3, "SSL Parameters for Web Server Plug-Ins", to determine when to use SSL. There are two key SSL parameters:

- `WLSSLWallet` -- The version 1.1 plug-ins use Oracle wallets to store SSL configuration information. The plug-ins introduce a new SSL configuration parameter `WLSSLWallet` to use Oracle wallets. The orapki utility is provided in the 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" for information about this tool.

- `SecureProxy` -- The `SecureProxy` parameter determines whether SSL is enabled or not.

In the case of two-way SSL, the plug-in (the SSL client) automatically uses two-way SSL when the WebLogic Server is configured for two-way SSL and requests a client certificate.

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

Note: If you have an Oracle Fusion Middleware 11*g* Release 11 (11.1.1) product installed on the same system as the Apache (including Oracle HTTP) plug-in, the `ORACLE_HOME` variable must point to a valid installation or the plug-in fails to initialize SSL.

For example, if `ORACLE_HOME` is invalid because the product was not cleanly removed, the plug-in fails to initialize SSL.

6.1.1 Configure Libraries for SSL

The plug-ins use Oracle libraries (NZ) to provide SSL support. Because the libraries are large, they are dynamically linked only when SSL is needed. You need to make sure that the library files, located in `lib/*.so*`, are available in the proper locations so that they can be dynamically loaded by the plug-in.

6.1.1.1 Configure Apache Libraries for SSL

To configure the libraries for the Apache plug-in (used for both the Apache HTTP Server and the Oracle HTTP Server) you have a few options:

1. For Windows, the `lib\*.dll` directory must be in the PATH variable, or add the *.dll files to the `Apache/bin` directory.
2. For Unix, copy the binaries to the Apache `lib` folder, or configure `LD_LIBRARY_PATH` to point the the folder containing the binaries.

6.1.2 Configuring a Plug-In for One-Way SSL

After you have installed and configured a plug-in as described in the respective plug-in-specific chapter, you can configure that plug-in to use 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 WebLogic Server is installed. You run the `orapki` commands on the system on which the version 1.1 plug-ins are installed.

Note: This section uses the WebLogic Server demo CA for the purpose of example.

If you are using the plug-in in a production environment, make sure that trusted CAs are properly configured for the plug-in as well as for WebLogic Server.

---

1. Configure WebLogic Server for SSL. For more information, see "Configuring SSL" in *Securing Oracle WebLogic Server*.
2. Configure the WebLogic Server SSL listen port. For more information, see "Configuring SSL" in *Securing Oracle WebLogic Server*.
3. Create an Oracle Wallet with the orapki utility.

See "Using the orapki Utility for Certificate Validation and CRL Management" in *Oracle Fusion Middleware Administrator's Guide* for information about this tool.

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 Apache plug-in because Apache runs as the account SYSTEM on Windows, and as the user who creates it on UNIX. However, for IIS the wallet will not work because the default user is IUSR_<Machine_Name>(IIS6.0 and below) or IUSR (IIS7.0).

If the user who runs the Apache plug-in or IIS plug-in is not the same 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:

IIS6.0 and below:

```
cacls <wallet_path>\cwallet.sso /e /g IUSR_<Machine_
Name>:R
```

IIS7.0:

```
cacls <wallet_path>\cwallet.sso /e /g IUSR:R
```

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

4. Import the *WL_HOME*`\server\lib\`*CertGenCA.der* CA into the Oracle Wallet.

```
orapki wallet add -wallet mywallet -trusted_cert -cert CertGenCA.der -auto_
login_only
```

5. For the Apache Plug-in, in the HTTP Server, edit the `httpd.conf` file as follows:

```
<IfModule mod_weblogic.c>
WebLogicHost my-weblogic.server.com
  WebLogicPort weblogic-server-secure-port
  SecureProxy ON
  WLSSLWallet /home/myhome/mywallet
</IfModule>
```

Where:

- *my-weblogic-server.com* is your WebLogic Server system.
- *weblogic-server-secure-port* is the port used for SSL, typically 7002.
- The `SecureProxy` parameter determines whether SSL is enabled or not.
- `WLSSLWallet` takes the path of an Oracle Wallet as an argument.

6. For the IIS plug-in, edit the Microsoft Internet Information Server `iisproxy.ini` file as follows:

```
WebLogicHost=my-weblogic.server.com
  WebLogicPort=weblogic-server-secure-port
  SecureProxy=ON
  WLSSLWallet=c:\home\myhome\mywallet
```

Where:

- *my-weblogic-server.com* is your WebLogic Server system.
- *weblogic-server-secure-port* is the port used for SSL, typically 7002.
- The SecureProxy parameter determines whether SSL is enabled or not.
- WLSSLWallet takes the path of an Oracle Wallet as an argument.

7. For the Apache Plug-in, set any additional parameters in the httpd.conf file that define information about the SSL connection. For a complete list of the SSL parameters that you can configure for the plug-in, see Section 7.3, "SSL Parameters for Web Server Plug-Ins".
8. For the IIS plug-in, set any additional parameters in the iisproxy.ini file that define information about the SSL connection. For a complete list of the SSL parameters that you can configure for the plug-in, see Section 7.3, "SSL Parameters for Web Server Plug-Ins".
9. Send a request to *http://apache-host:apache-port/mywebapp/my.jsp* from the browser. Validate the response.

6.1.3 Configure Two-Way SSL Between the Plug-In and WebLogic Server

After you have installed and configured a plug-in as described in the respective plug-in-specific chapter, you can configure that plug-in to use two-way SSL.

You configure two-way SSL by importing a user certificate into the Wallet. When WebLogic Server is configured for two-way SSL, the plug-in forwards the user certificate to WebLogic Server. As long as WebLogic Server can validate the user certificate, two-way SSL can be established.

In addition to the steps described in Section 6.1.2, "Configuring a Plug-In for One-Way SSL" to configure SSL, perform the following additional steps to configure two-way SSL between the plug-in and WebLogic Server.

Again, in these steps, you run the keytool commands on the system on which WebLogic Server is installed. You run the orapki commands on the system on which the version 1.1 plug-ins are installed.

1. From the Oracle wallet, generate a certificate request.
2. Use this certificate request to create a certificate via a CA or some other mechanism.
3. Import the user certificate as a trusted certificate in the WebLogic truststore. WebLogic Server needs to trust the certificate.

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

4. Set the WebLogic Server SSL configuration options that require the presentation of client certificates (for two-way SSL). See "Configure two-way SSL" in the *Oracle WebLogic Server Administration Console Help*.

6.1.4 Issues with SSL-Apache Configuration

These known issues arise when you configure the Apache plug-in to use SSL:

- The PathTrim parameter (see Section 7.3, "SSL Parameters for Web Server Plug-Ins") must be configured inside the <Location> tag.

  The following configuration is **incorrect**:

  ```
  <Location /weblogic>
  ```

```
 SetHandler weblogic-handler
</Location>

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

The following configuration is the **correct** setup:

```
<Location /weblogic>
  SetHandler weblogic-handler
  PathTrim /weblogic
</Location>
```

- The current implementation of the WebLogic Server Apache Plug-In does not support the use of multiple certificate files with Apache SSL.

6.2 Use IPv6 With Plug-Ins

The version 1.1 plug-ins support IPv6. Specifically, the `WebLogicHost` and `WebLogicCluster` configuration parameters (see Table 7–1) now support IPv6 addresses. 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.

> **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 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 `mod_wl_ohs.conf` file also improves performance with IPv6. When set to OFF, the plug-in ignores the dynamic cluster list used for load balancing requests proxied from the plug-in and uses the static list specified with the WebLogicCluster parameter.

6.3 Set Up Perimeter Authentication

Use perimeter authentication to secure WebLogic Server applications that are accessed via the plug-in.

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

1. Create a custom Identity Assertion Provider on your 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 (or, if using a cluster, optionally set the `Client Cert Proxy Enabled` attribute to true for the whole cluster on the Administration Console Cluster-->Configuration-->General tab).

   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 WebLogic Server accepts connections only from the machine on which the plug-in is running. See "Using Network Connection Filters" in *Programming Security for Oracle WebLogic Server*.
5. Web server plug-ins require a trusted Certificate Authority file in order to use SSL between the plug-in and WebLogic Server. See Section 6.1, "Use SSL With Plug-Ins" for the steps you need to perform to configure SSL.

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

6.4 Set the WebLogic Plug-in Enabled Control in WebLogic Server

Set the **WebLogic Plug-in Enabled** control in WebLogic Server.

The **WebLogic Plug-in Enabled** control specifies whether the WebLogic Server uses the proprietary WL-Proxy-Client-IP header, which is recommended if the server instance will receive requests from a proxy plug-in.

6.5 Understanding Connection Errors and Clustering Failover

When the plug-in attempts to connect to WebLogic Server, 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. If the plug-in cannot connect or does not receive a response, the plug-in attempts to connect and send the request to other WebLogic Server instances in the cluster. If the connection fails or there is no response from any WebLogic Server in the cluster, an error message is sent.

Figure 6–1 demonstrates how the plug-in handles failover.

6.5.1 Possible Causes of Connection Failures

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

- Physical problems with the host machine
- Network problems
- Other server failures

Failure of all WebLogic Server instances to respond could indicate the following problems:

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

6.5.2 Tuning Apache Plug-In to Reduce Connection_Refused Errors

Under load, an Apache plug-in may receive CONNECTION_REFUSED errors from a back-end WebLogic Server instance. Follow these tuning tips to reduce CONNECTION_REFUSED errors:

- Increase the `AcceptBackLog` setting in the configuration of your WebLogic Server domain.
- Decrease the time wait interval. This setting varies according to the operating system you are using. For example:
  - On Windows NT, set the `TcpTimedWaitDelay` on the proxy and WebLogic Server servers to a lower value. Set the TIME_WAIT interval in Windows NT by editing the registry key under HKEY_LOCAL_MACHINE:

    ```
    SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpTimedWaitDelay
    ```

    If this key does not exist you can create it as a DWORD value. The numeric value is the number of seconds to wait and may be set to any value between 30 and 240. If not set, Windows NT defaults to 240 seconds for TIME_WAIT.
  - On Windows 2000, lower the value of the `TcpTimedWaitDelay` by editing the registry key under HKEY_LOCAL_MACHINE:

    ```
    SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
    ```

  - On Solaris, reduce the setting `tcp_time_wait_interval` to one second (for both the WebLogic Server machine and the Apache machine, if possible):

    ```
    $ndd /dev/tcp
       param name to set - tcp_time_wait_interval
       value=1000
    ```

- 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. For example:

  ```
  #!/bin/sh
  ulimit -S -n 100
  exec httpd
  ```

- On Solaris, increase the values of the following tunables on the WebLogic Server machine:

```
tcp_conn_req_max_q
tcp_conn_req_max_q0
```

6.5.3 Failover with a Single, Non-Clustered WebLogic Server

If you are running only a single WebLogic Server instance the 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 plug-in continues trying to connect to that same WebLogic Server instance for the maximum number of retries as specified by the ratio of `ConnectTimeoutSecs` and `ConnectRetrySecs`.

6.5.4 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.

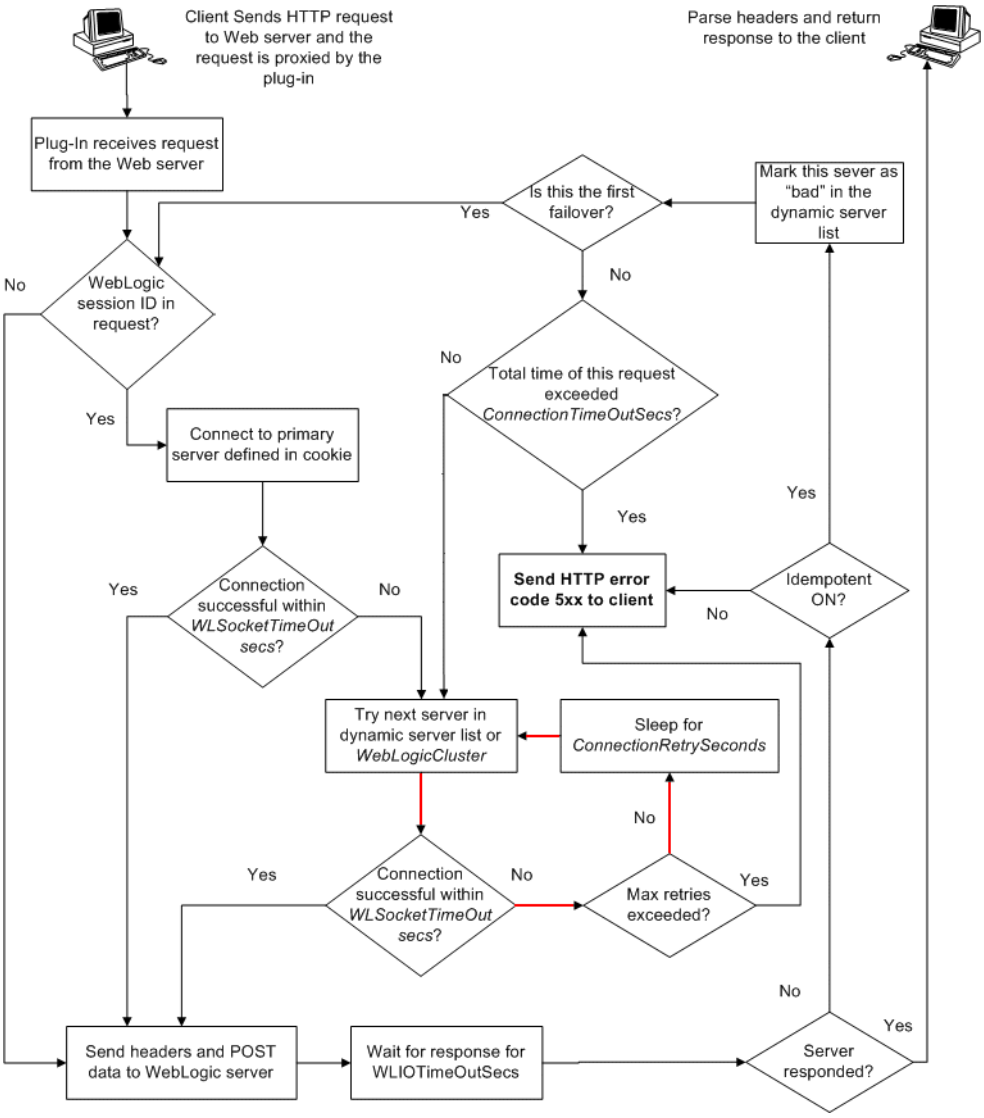
In the case of proxying to clustered managed servers, when you use the `WebLogicCluster` parameter in your `httpd.conf` or `weblogic.conf` file to specify a list of WebLogic Servers, the plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request 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 are no longer part of the cluster or that have failed to respond to requests. This list is updated automatically with the HTTP response when a change in the cluster occurs.

6.5.5 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 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 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 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`.

7

Parameters for Web Server Plug-Ins

The following sections describe the parameters that you use to configure the Apache and Microsoft IIS Web server plug-ins:

- Section 7.1, "Entering Parameters in Web Server Plug-In Configuration Files"
- Section 7.2, "General Parameters for Web Server Plug-Ins"
- Section 7.3, "SSL Parameters for Web Server Plug-Ins"

7.1 Entering Parameters in Web Server Plug-In Configuration Files

You enter the parameters for each Web server plug-in in special configuration files. Each Web server has a different name for this configuration file and different rules for formatting the file. For details, see the following sections on each plug-in:

- Chapter 3, "Installing and Configuring the Apache HTTP Server Plug-In"
- Chapter 5, "Installing and Configuring the Microsoft IIS Plug-In"

7.2 General Parameters for Web Server Plug-Ins

The general parameters for Web server plug-ins are shown in Table 7–1. Parameters are case sensitive.

Table 7–1 General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`WebLogicHost` (Required when proxying to a single WebLogic Server.)	none	WebLogic Server host (or virtual host name as defined in WebLogic Server) to which HTTP requests should be forwarded. If you are using a WebLogic cluster, use the `WebLogicCluster` parameter instead of `WebLogicHost`.	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`WebLogicPort` (Required when proxying to a single WebLogic Server.)	none	Port at which the WebLogic Server host is listening for connection requests from the plug-in (or from other servers). (If you are using SSL between the plug-in and WebLogic Server, set this parameter to the SSL listen port (see Configuring SSL) and set the `SecureProxy` parameter to ON). If you are using a WebLogic Cluster, use the `WebLogicCluster` parameter instead of `WebLogicPort`.	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`WebLogicCluster` (Required when proxying to a cluster of WebLogic Servers, or to multiple non-clustered servers.)	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 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. The method of specifying the parameter, and the required format vary by plug-in. See the examples in: ■ Chapter 5, "Installing and Configuring the Microsoft IIS Plug-In" ■ Chapter 3, "Installing and Configuring the Apache HTTP Server Plug-In" If you are using SSL between the plug-in and WebLogic Server, set the port number to the SSL listen port (see Configuring SSL) and set the `SecureProxy` parameter to ON. The 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 plug-in maintain. WebLogic Server and the 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 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 originally created the cookie.	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`PathTrim`	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 plug-in 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 plug-in for parsing and if `PathTrim` 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:7001/foo Note that if you are newly converting an existing third-party server to proxy requests to WebLogic Server using the 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.	ISAPI, Apache
`PathPrepend`	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 plug-in prepends to the `{PATH}` portion of the original URL, after PathTrim is trimmed and before the request is forwarded to WebLogic Server. Note that if you need to append File Name, use `DefaultFileName` plug-in parameter instead of `PathPrepend`.	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`ConnectTimeoutSecs`	10	Maximum time in seconds that the plug-in should attempt to connect to the 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.	ISAPI and Apache
`ConnectRetrySecs`	2	Interval in seconds that the plug-in should sleep between attempts to connect to the WebLogic Server host (or all of the servers in a cluster). Make this number less than the `ConnectTimeoutSecs`. The number of times the plug-in tries to connect before returning an `HTTP 503/Service Unavailable` response to the client is calculated by dividing `ConnectTimeoutSecs` by `ConnectRetrySecs`. To specify no retries, set `ConnectRetrySecs` equal to `ConnectTimeoutSecs`. However, the plug-in attempts to connect at least twice. You can customize the error response by using the `ErrorPage` parameter.	ISAPI and Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`Debug`	OFF	Sets the type of logging performed for debugging operations. The debugging information is written to the `/tmp/wlproxy.log` file on UNIX systems and `c:\TEMP\wlproxy.log` on Windows NT/2000 systems. Override this location and filename by setting the WLLogFile parameter to a different directory and file. (See the `WLTempDir` parameter for an additional way to change this location.) Ensure that the tmp or TEMP directory has write permission assigned to the user who is logged in to the server. Set any of the following logging options (HFC,HTW,HFW, and HTC options may be set in combination by entering them separated by commas, for example "HFC,HTW"): `ON` - The plug-in logs informational and error messages. `OFF` - No debugging information is logged. `HFC` - The plug-in logs headers from the client, informational, and error messages. `HTW` - The plug-in logs headers sent to WebLogic Server, and informational and error messages. `HFW` - The plug-in logs headers sent from WebLogic Server, and informational and error messages. `HTC` - The plug-in logs headers sent to the client, informational messages, and error messages. `ERR` - Prints only the Error messages in the plug-in. `ALL` - The plug-in logs headers sent to and from the client, headers sent to and from WebLogic Server, information messages, and error messages.	ISAPI and Apache
`WLLogFile`	See the `Debug` parameter	Specifies path and file name for the log file that is generated when the `Debug` parameter is set to `ON`. You must create this directory before setting this parameter.	ISAPI and Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
`WLDNSRefreshInterval`	0 (Lookup once, during startup)	Only applies to Apache. If defined in the proxy configuration, specifies number of seconds interval at which WebLogic Server refreshes DNS name to IP mapping for a server. This can be used in the event that a 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.	Apache plug-in
WLTempDir	See the Debug parameter	Specifies the directory where a `wlproxy.log` will be created. If the location fails, the Plug-In resorts to creating the log file under `C:/temp` in Windows and `/tmp` in all Unix platforms. Also specifies the location of the `_wl_proxy` directory for POST data files. When both `WLTempDir` and `WLLogFile` are set, `WLLogFile` will override as to the location of `wlproxy.log`. `WLTempDir` will still determine the location of `_wl_proxy` directory.	ISAPI and Apache plug-in
DebugConfigInfo	OFF	Enables the special query parameter "__ WebLogicBridgeConfig". Use it to get details about configuration parameters from the plug-in. For example, if you enable "__ WebLogicBridgeConfig" by setting DebugConfigInfo and then send a request that includes the query string ?__ WebLogicBridgeConfig, then the plug-in gathers the configuration information and run-time statistics and returns the information to the browser. The plug-in does not connect to 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.	ISAPI and Apache plug-in

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
StatPath (Not available for the Microsoft Internet Information Server Plug-In)	false	If set to true, the plug-in checks the existence and permissions of the translated path ("Proxy-Path-Translated") of the request before forwarding the request to WebLogic Server. If the file does not exist, an HTTP 404 File Not Found response is returned to the client. If the file exists but is not world-readable, an HTTP 403/Forbidden response is returned to the client. In either case, the default mechanism for the Web server to handle these responses fulfills the body of the response. This option is useful if both the WebLogic Server Web Application and the Web Server have the same document root. You can customize the error response by using the ErrorPage parameter.	Apache plug-in
ErrorPage	none	You can create your own error page that is displayed when your Web server is unable to forward requests to WebLogic Server.	ISAPI, Apache
WLSocketTimeoutSecs	2 (must be greater than 0)	Set the timeout for the socket while connecting, in seconds.	
WLIOTimeoutSecs (new name for HungServerRecoverSecs)	300	Defines the amount of time the plug-in waits for a response to a request from WebLogic Server. The plug-in waits for `WLIOTimeoutSecs` for the server to respond and then declares that server dead, and fails over to the next server. The value should be set to a very large value. If the value is less than the time the servlets take to process, then you may see unexpected results. Minimum value: 10 Maximum value: Unlimited	ISAPI and Apache plug-in
Idempotent	ON	When set to ON and if the servers do not respond within WLIOTimeoutSecs (new name for HungServerRecoverSecs), 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. If you are using the Apache HTTP Server you can set this parameter differently for different URLs or MIME types.	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
WLCookieName CookieName parameter is deprecated	JSESSIONID	If you change the name of the WebLogic Server session cookie in the WebLogic Server Web application, you need to change the WLCookieName parameter in the plug-in to the same value. The name of the WebLogic session cookie is set in the WebLogic-specific deployment descriptor, in the `<session-descriptor>` element.	ISAPI and Apache
DefaultFileName	none	If the URI is "/" then the plug-in performs the following steps: Trims the path specified with the `PathTrim` parameter. Appends the value of `DefaultFileName`. Prepends the value specified with `PathPrepend`. This procedure prevents redirects from WebLogic Server. Set the DefaultFileName to the default welcome page of the Web Application in 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. For more information, see Configuring Welcome Pages. 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.	ISAPI and Apache
MaxPostSize	-1	Maximum allowable size of POST data, in bytes. If the content-length exceeds `MaxPostSize`, the plug-in returns an error message. If set to -1, 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.	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to

MatchExpression (Apache HTTP Server only)	none	When proxying by MIME type, set the filename pattern inside of an IfModule block using the MatchExpression parameter.
		Example when proxying by MIME type:
		`<IfModule weblogic_module>`
		`  MatchExpression *.jsp`
		`WebLogicHost=myHost|paramName`
		`=value`
		`</IfModule>`
		Example when proxying by path:
		`<IfModule weblogic_module>`
		`  MatchExpression /weblogic`
		`WebLogicHost=myHost|paramName`
		`=value`
		`</IfModule>`
		It is possible to define a new parameter for MatchExpression using the following syntax:
		MatchExpression *.jsp PathPrepend=/test PathTrim=/foo
			Apache plug-in

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
FileCaching	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 the 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. Note that 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 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 a WebLogic Server cluster member is identified to serve the request. Then the plug-in reads and immediately sends the POST data to the WebLogic Server in chunks of 8192 bytes. Note that turning `FileCaching OFF` limits failover. If the 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 plug-in will read the data into memory and use it if needed during failover to repeat to the secondary.	ISAPI, Apache
WLExcludePathOrMimeType	none	This parameter allows you make exclude certain requests from proxying. This parameter can be defined locally at the Location tag level as well as globally. When the property is defined locally, it does not override the global property but defines a union of the two parameters.	ISAPI and Apache plug-in

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
KeepAliveSecs	20	The length of time after which an inactive connection between the plug-in and WebLogic Server is closed. You must set `KeepAliveEnabled` to true (ON when using the Apache plug-in) 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 Administration Console on the Server/HTTP tab, or the value set on the server Mbean with the `KeepAliveSecs` attribute.	ISAPI, Apache
KeepAliveEnabled	true (Microsoft IIS plug-in) ON (Apache plug-in)	Enables pooling of connections between the plug-in and WebLogic Server. Valid values for the Microsoft IIS plug-ins are true and false. Valid values for the Apache plug-in are `ON` and `OFF`.	ISAPI, Apache
QueryFromRequest (Apache HTTP Server only)	OFF	When set to `ON`, specifies that the Apache plug-in use `(request_rec *)r->the request` to pass the query string to WebLogic Server. (For more information, see your Apache documentation.) This behavior is desirable in the following situations: ■ When a Netscape version 4.x browser makes requests that contain spaces in the query string ■ If you are using Raven Apache 1.5.2 on HP When set to `OFF`, the Apache plug-in uses `(request_rec *)r->args` to pass the query string to WebLogic Server.	Apache plug-in
MaxSkipTime	10	If a WebLogic Server listed in either the WebLogicCluster parameter or a dynamic cluster list returned from WebLogic Server fails, the failed server is marked as "bad" and the plug-in attempts to connect to the next server in the list. `MaxSkips` sets the amount of time after which the plug-in will retry the server marked as "bad." The 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).	ISAPI, Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
DynamicServerList	ON	When set to `OFF`, the plug-in ignores the dynamic cluster list used for load balancing requests proxied from the 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 plug-in could waste time trying to connect to a dead server, resulting in decreased performance. ■ If you add a new server to the cluster, the plug-in cannot proxy requests to the new server unless you redefine this parameter. WebLogic Server automatically adds new servers to the dynamic server list when they become part of the cluster.	ISAPI and Apache
WLProxySSL	OFF	Set this parameter to `ON` to maintain SSL communication between the plug-in and 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 WebLogic Server proxy plug-ins) ■ The connection between the plug-in and WebLogic Server uses the HTTP protocol When WLProxySSL is set to `ON`, the location header returned to the client from WebLogic Server specifies the HTTPS protocol.	ISAPI and Apache
WLProxyPassThrough	OFF	If you have a chained proxy setup, where a proxy plug-in or `HttpClusterServlet` 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.	ISAPI and Apache

Table 7–1 (Cont.) General Parameters for Web Server Plug-Ins

Parameter Name	Default	Description	Applicable to
WLLocalIP	none	Defines the IP address (on the plug-in's system) to bind to when the plug-in connects to a WebLogic Server instance running on a multihomed machine. If `WLLocalIP` is not set, a random IP address on the multi-homed machine is used.	ISAPI and Apache
WLSendHdrSeparately	ON	When this parameter is set to ON, header and body of the response are sent in separate packets. **Note:** If you need to send the header and body of the response in two calls, for example, in cases where you have other ISAPI filters or programmatic clients that expect headers before the body, set this parameter to ON.	ISAPI plug-in

7.2.1 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 the 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):

1. Environment variable `TMP`
2. Environment variable `TEMP`
3. `C:\Temp`

`/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.

7.3 SSL Parameters for Web Server Plug-Ins

Note: SCG Certificates are not supported for use with WebLogic Server Proxy Plug-Ins. Non-SCG certificates work appropriately and allow SSL communication between WebLogic Server and the plug-in.

KeyStore-related initialization parameters are not supported for use with WebLogic Server Proxy Plug-Ins

The SSL parameters for Web Server plug-ins are shown in Table 7–2. Parameters are case sensitive.

Table 7–2 SSL Parameters for Web Server Plug-Ins

Parameter	Default	Description	Applicable to
SecureProxy	OFF	Set this parameter to ON to enable the use of the SSL protocol for all communication between the plug-in and WebLogic Server. Remember to configure a port on the corresponding 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.	ISAPI and Apache
WLSSLWallet	none	WLSSLWallet performs one-way or two-way SSL based on how WebLogic Server SSL is configured. Requires the path of an Oracle Wallet (containing an SSO wallet file) as argument. For example, `WLSSLWallet "ORACLE_INSTANCE}/config/COMPONENT_TYPE/COMPONENT_NAME/default"`	