Oracle® Fusion Middleware

Using Web Server Plug-Ins with Oracle WebLogic Server

11*g* Release 1 (10.3.1)

E14395-02

October 2009

This document explains the use of 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 Plug-Ins with Oracle WebLogic Server, 11*g* Release 1 (10.3.1)

E14395-02

Primary Author: Oracle Corporation

Contents

4 Installing and Configuring the Microsoft IIS Plug-In

5 Installing and Configuring the Sun Java System Web Server Plug-In

Preface

This preface describes the document accessibility features and conventions used in this guide—*Oracle Fusion Middleware Using Web Server 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.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

Conventions

The following text conventions are used in this document:

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

1 Introduction and Roadmap

rss.This section describes the contents and organization of this guide—*Oracle Fusion Middleware Using Web Server 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.

1.3 Related Documentation

This document contains information on using Web server plug-ins. For information on using a proxy plug-in, see *Oracle Fusion Middleware Using Clusters for Oracle WebLogic Server* .

1.4 New and Changed Features in This Release

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

2

Understanding Using Web Server 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. Plug-ins enable WebLogic Server to communicate with applications deployed on Oracle HTTP Server, Apache HTTP Server, Sun Java System Web Server, or Microsoft's Internet Information Server. Typically, WebLogic Server handles the application requests that require dynamic functionality, the requests that can best be served with dynamic HTML pages or JSPs (Java Server Pages).

2.2 Plug-Ins Included with Oracle WebLogic Server

WebLogic Server includes plug-ins for the following Web servers:

- Apache HTTP Server
- Microsoft Internet Information Server
- Sun Java System Web Server

2.3 Plug-In Versions

This release of *Oracle Fusion Middleware Using Web Server Plug-Ins with Oracle WebLogic Server* documents the following plug-ins:

- Apache HTTP Server, version 1.0.1211636
- Microsoft Internet Information Server, version 1.0.1211636
- Sun Java System Web Server, version 1.0.1211636

2.4 Oracle HTTP Server Plug-In Support

Plug-in support is also available for Oracle HTTP Server. These plug-ins are packaged with the Oracle HTTP Server distribution. See *Oracle Fusion Middleware Administrator's Guide for Oracle HTTP Server*.

2.5 Plug-In Two-Way SSL Support

WebLogic Server Plug-Ins do not support two-way SSL. However, the Plug-Ins can be set up to require the client certificate and pass it on to WebLogic Server. For example:

```
apache ssl
SSLVerifyClient require
SSLVerifyDepth 10
SSLOptions +FakeBasicAuth +ExportCertData +CompatEnvVars +StrictRequire
```

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:

3.1 Overview of the Apache HTTP Server Plug-In

The Apache HTTP Server Plug-In allows requests to be proxied from an Apache HTTP Server to WebLogic Server. The plug-in enhances an Apache installation by allowing WebLogic Server to handle requests that require the dynamic functionality of WebLogic Server.

The plug-in is intended for use in an environment where an Apache Server serves static pages, and another part of the document tree (dynamic pages best generated by HTTP Servlets or JavaServer Pages) is delegated to WebLogic Server, which 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 same source.

HTTP-tunneling, a technique which allows HTTP requests and responses access through a company's firewall, can also operate through the plug-in, providing non-browser clients access to WebLogic Server services.

The Apache HTTP Server Plug-In operates as an Apache module within an Apache HTTP Server. An Apache module is loaded by Apache Server at startup, and then certain HTTP requests are delegated to it. Apache modules are similar to HTTP servlets, except that an Apache module is written in code native to the platform.

For information on configurations on which the Apache HTTP Server Plug-In is supported, see http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html.

Note: Apache 2.0 Plug-In was deprecated in the WebLogic Server 10.0 release.

3.1.1 Keep-Alive Connections in Apache Version 2.0

Version 2.0 of the Apache HTTP Server Plug-In improves performance by using a reusable 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 in the pool 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 and removed from the pool. You can disable this feature if desired. For more information, see `KeepAliveEnabled` in Table 7–1.

3.1.2 Proxying Requests

The plug-in proxies requests to WebLogic Server based on a configuration that you specify. You can proxy requests based on the URL of the request (or a portion of the URL). This is called proxying by *path*. You can also proxy requests based on the *MIME type* of the requested file. Or you can use a combination of the two methods. If a request matches both criteria, the request is proxied by path. You can also specify additional parameters for each type of request that define additional behavior of the plug-in. For more information, see "Configuring the Apache HTTP Server Plug-In" on page 3-5.

3.1.3 Apache 2.2

Although this document refers to Apache 2.0, you can apply the same instructions to use Apache 2.2 with the libraries shown in Table 3–2.

3.1.4 Certifications

The Apache HTTP Server Plug-In is supported on AIX, Linux, Solaris, Windows, and HPUX11 platforms. For information on support for specific versions of Apache, see `http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html`.

3.2 Installing the Apache HTTP Server Plug-In

You can install the Apache HTTP Server Plug-In 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 run time, and can be installed without recompiling Apache.

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

The Apache plug-in is distributed as a shared object (.so) for Solaris, Linux, AIX, Windows, and HPUX11 platforms.

Note: The WebLogic Server version 10.3 installation did not include the Apache HTTP server plug-ins. The Apache HTTP Server plug-ins are available in a separate zip file, available from the Oracle download and support sites. These plug-ins contain a fix for the security advisory described at:

http://www.oracle.com/technology/deploy/security/alerts/alert_cve2008-3257.html

After downloading the zip file, extract the zip to a directory of your choice on disk.

Table 3–1 shows the directories created after extracting the zip file to the directory you specified. These directories contain shared object files for various platforms.

Table 3–2 identifies the WebLogic Server Apache Plug-In modules for different versions of Apache HTTP Server and different encryption strengths.

Table 3–1 Locations of Plug-In Shared Object Files

Operating System	Shared Object Location
AIX	aix/ppc
Solaris	solaris/sparc solaris/sparc/largefile[1] solaris/x86 solaris/x86/largefile[2]
Linux	linux/i686 linux/i686/largefile[3] linux/ia64 linux/x86_64
Windows (Apache 2.0 and 2.2, 32-bit)	win\32
HPUX11	hpux11/IPF64 hpux11/PA_RISC **Note:** If you are running Apache 2.0.x server on HP-UX11, set the environment variables specified immediately below before you build the Apache server. Because of a problem with the order in which linked libraries are loaded on HP-UX, a core dump can result if the load order is not preset as an environment variable before building. Set the following environment variables before proceeding with the Apache configure, make, and make install steps, (described in Apache HTTP Server documentation at http://httpd.apache.org/docs-2.1/install.html#configure): `export EXTRA_LDFLAGS="-lstd -lstream -lCsup -lm -lcl -ldld -lpthread"`

[1] See "Support for Large Files in Apache 2.0" on page 3-5
[2] See "Support for Large Files in Apache 2.0" on page 3-5
[3] See "Support for Large Files in Apache 2.0" on page 3-5

Choose the appropriate version of the plug-in shared object from the following table:

Table 3–2 Apache Plun-In Shared Object File Versions

Apache Version	Regular Strength Encryption	128-bit Encryption
Standard Apache Version 2.0.x	mod_wl_20.so	mod_wl128_20.so
Standard Apache Version 2.2.x	mod_wl_22.so	mod_wl128_22.so

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

1. In the location where you unzipped the downloaded plug-in file, locate the shared object directory for your platform using Table 3-1.

   Note: Before making your selection, please review "Support for Large Files in Apache 2.0" on page 3-5.

2. Identify the plug-in shared object file for your version of Apache in Table 3–2.
3. Verify that the WebLogic Server Apache HTTP Server Plug-In 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_20.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 2.0 Shared Object (DSO) Support at `http://httpd.apache.org/docs/2.0/dso.html`.

4. Install the Apache HTTP Server Plug-In module for Apache 2.0.x by copying the *mod_wl_20.so* file to the *APACHE_HOME\modules* directory and adding the following line to your *APACHE_HOME/conf/httpd.conf* file manually:

   ```
   LoadModule weblogic_module     modules/mod_wl_20.so
   ```

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

   The Apache HTTP Server Plug-In recognizes the parameters listed in "General Parameters for Web Service Plug-Ins" on page 7-1. To modify the behavior of your Apache HTTP Server Plug-In, define these parameters:

   - 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.

6. Verify the syntax of the *APACHE_HOME\conf\httpd.conf* file with the following command:

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

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

   ```
   Syntax OK
   ```

7. Restart Weblogic Server.
8. Start (or restart if you have changed the configuration) Apache HTTP Server.
9. Test the plug-in by opening a browser and setting the URL to the Apache Server plus "/weblogic/", which should bring up the default WebLogic Server HTML page, welcome file, or default servlet, as defined for the default Web Application on WebLogic Server. For example:

   ```
   http://myApacheserver.com/weblogic/
   ```

3.2.2 Support for Large Files in Apache 2.0

The version of Apache 2.0 that ships with some operating systems, including some versions of Solaris and Linux, is compiled with the following flags:

```
-D_LARGEFILE_SOURCE
-D_FILE_OFFSET_BITS=64
```

These compilation flags enable support for files larger than 2 GB. If you want to use a WebLogic Server Web server plug-in on such an Apache 2.0 Web server, you must use plug-ins compiled with the same compilation flags, which are available in the largefile subdirectory for your operating system. For example:

```
C:\WL_HOME\server\plugin\solaris\sparc\largefile
```

Note: Apache 2.2 does not require special compilation flags to support files larger than 2 GB. Therefore, you do not need to use a specially compiled Web server plug-in if you are running Apache 2.2.

3.3 Configuring 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 Apache *httpd.conf* file to instruct the Apache server to load the WebLogic Server library for the plug-in as an Apache module, and to specify the application requests that should be handled by the module.

3.3.1 Editing the httpd.conf File

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

This section explains how to locate and edit the *httpd.conf* file, to configure the server to use the WebLogic Server Apache Plug-In, to proxy requests by path or by MIME type, to enable HTTP tunneling, and to use other WebLogic Server plug-in parameters.

1. 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 "Setting Up Perimeter Authentication" on page 3-11.

2. Ensure that the WebLogic Server modules are included for Apache 2.0.x, manually add the following line to the *httpd.conf* file:

```
LoadModule weblogic_module   modules\mod_wl_20.so
```

3. 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 myweblogic.server.com
  WebLogicPort 7001
</IfModule>
```

4. 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 myweblogic.server.com
  WebLogicPort 7001
  MatchExpression *.jsp
</IfModule>
```

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

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

5. 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 "General Parameters for Web Service Plug-Ins" on page 7-1).

6. 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 /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 /iiop>
     SetHandler weblogic-handler
   </Location>
   ```

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

   The Apache HTTP Server Plug-In recognizes the parameters listed in "General Parameters for Web Service Plug-Ins" on page 7-1. 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.

3.3.1.1 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>
```

3.3.2 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.

3.3.2.1 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.

3.3.2.2 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.

3.3.2.3 Template for the Apache HTTP Server httpd.conf File

This section contains a sample *httpd.conf* file for Apache 2.0. 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 2.0

```
###################################################
APACHE-HOME/conf/httpd.conf file
###################################################
```

```
LoadModule weblogic_module   libexec/mod_wl_20.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>
```

3.4 Setting Up Perimeter Authentication

Use perimeter authentication to secure WebLogic Server applications that are accessed via the Apache 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 Apache HTTP Server 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 *Oracle Fusion Middleware 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 *Oracle Fusion Middleware 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 *Oracle Fusion Middleware 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 Apache Plug-In is running. See Using Network Connection Filters in *Oracle Fusion Middleware 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. Use Sun Microsystems' keytool utility to export a trusted Certificate Authority file from the *DemoTrust.jks* keystore file that resides in *WL_HOME/server/lib*.
   a. To extract the wlsdemoca file, for example, use the command:

      ```
      keytool -export -file trustedcafile.der -keystore DemoTrust.jks -alias
      wlsdemoca
      ```

Change the alias name to obtain a different trusted CA file from the keystore.

To look at all of the keystore's trusted CA files, use:

```
keytool -list -keystore DemoTrust.jks
```

Press enter if prompted for password.

b. To convert the Certificate Authority file to pem format: java utils.der2pem trustedcafile.der

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

3.5 Using SSL with the Apache Plug-In

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

The Apache HTTP Server 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 the SSL protocol is used to protect the connection between the Apache HTTP Server Plug-In and WebLogic Server.

Although two-way SSL can be used between the HTTP client and Apache HTTP server, note that one-way SSL is used between Apache HTTP Server and WebLogic Server.

3.5.1 Configuring SSL Between the Apache HTTP Server Plug-In and WebLogic Server

To use the SSL protocol between Apache HTTP Server Plug-In and WebLogic Server:

1. Configure WebLogic Server for SSL. For more information, see Configuring SSL.
2. Configure the WebLogic Server SSL listen port. For more information, see Configuring SSL.
3. In the Apache Server, set the `WebLogicPort` parameter in the *httpd.conf* file to the WebLogic Server SSL listen port configured in Step 2.
4. In the Apache Server, set the *SecureProxy* parameter in the *httpd.conf* file to ON.
5. 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 "SSL Parameters for Web Server Plug-Ins" on page 7-14.

3.5.2 Issues with SSL-Apache Configuration

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

- To prepare the plugin configuration, double click the lock and go to the certificates path:
  - Select the root CA (at the top).
  - Display it.
  - Detail and then copy this certificate to a file using the Coded "Base 64 X509" option.

  - Save the file, for example, to "MyWeblogicCAToTrust.cer" (which is also a PEM file).

- The `PathTrim` parameter (see "SSL Parameters for Web Server Plug-Ins" on page 7-14) 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.

3.6 Connection Errors and Clustering Failover

When the Apache HTTP Server 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 3–1 demonstrates how the plug-in handles failover.

3.6.1 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

3.6.2 Tuning 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.
- On Apache 2.0.x, set the `KeepAlive` directive in the *httpd.conf* file to On. For example:

```
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive On
```

  See Apache HTTP Server 2.0 documentation at http://httpd.apache.org/docs-project/.
- 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
```

3.6.3 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*.

3.6.4 The Dynamic Server List

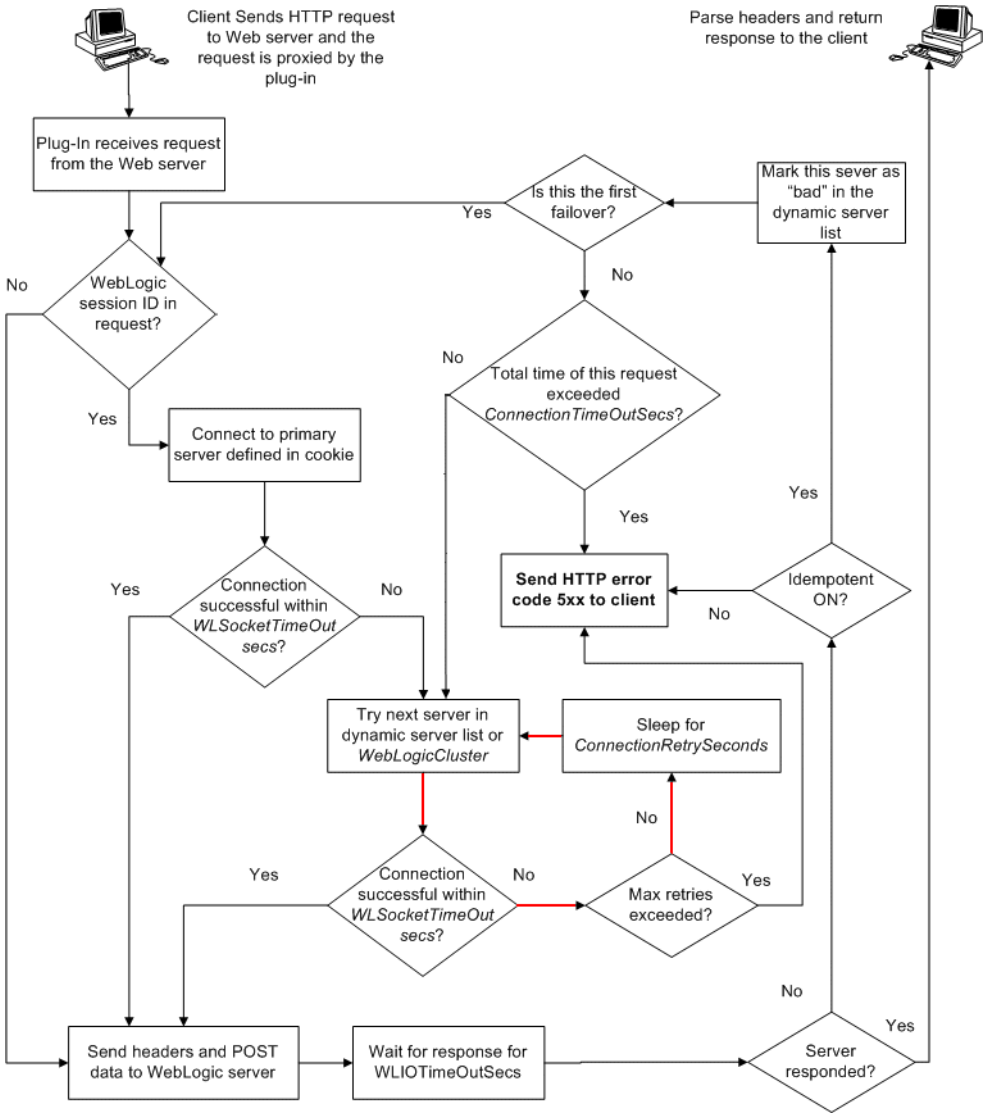
When you use the WebLogicCluster parameter in your *httpd.conf* or *weblogic.conf* file to specify a list of WebLogic Servers, the plug-in uses that list as a starting point for load balancing among the members of the cluster. After the first request is routed to one of these servers, a dynamic server 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.

3.6.5 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 3–1.

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 3–1 Connection Failover



In this figure, the Maximum number of retries allowed in the red loop is equal to `ConnectTimeoutSecs/ConnectRetrySecs`.

4

Installing and Configuring the Microsoft IIS Plug-In

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

4.1 Overview of the Microsoft Internet Information Server Plug-In

The Microsoft Internet Information Server Plug-In allows requests to be proxied from a Microsoft Internet Information Server (IIS) to WebLogic Server. The plug-in enhances an IIS installation by allowing WebLogic Server to handle those requests that require the dynamic functionality of WebLogic Server.

You use the Microsoft Internet Information Server Plug-In in an environment where the Internet Information Server (IIS) serves static pages such as HTML pages, while dynamic pages such as HTTP Servlets or JavaServer Pages 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 IIS. The HTTP-tunneling facility of the WebLogic client-server protocol also operates through the plug-in, providing access to all WebLogic Server services.

4.1.1 Connection Pooling and Keep-Alive

The Microsoft Internet Information Server Plug-In improves 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 re-using the same connection for subsequent requests from the same client. If the connection is inactive for more than 30 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.

4.1.2 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. For more information, see "Using Wildcard Application Mappings to Proxy by Path" on page 4-2 and "Installing and Configuring the Microsoft Internet Information Server Plug-In" on page 4-4.

4.2 Certifications

For the latest information on operating system and IIS version compatibility with the Microsoft Internet Information Server Plug-In, see `http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html`.

4.3 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.

You may find this to be a more convenient method of proxying by path than using *iisforward.dll* as described in "Installing and Configuring the Microsoft Internet Information Server Plug-In" on page 4-4.

4.3.1 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.

4.3.2 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:

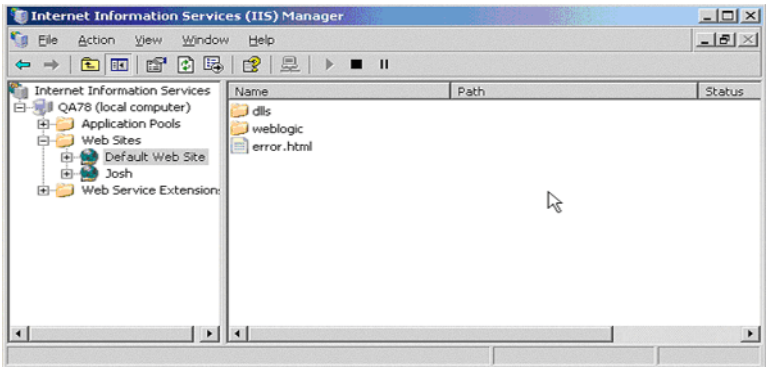
1. Open IIS Manager and navigate to the level you want to manage. For information about opening IIS Manager, see "Open IIS Manager" at http://technet.microsoft.com/en-us/library/cc770472(WS.10).aspx. For information about navigating to locations in the UI, see "Navigation in IIS Manager" at http://technet.microsoft.com/en-us/library/cc732920(WS.10).aspx.
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.

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

To install the Microsoft Internet Information Server Plug-In:

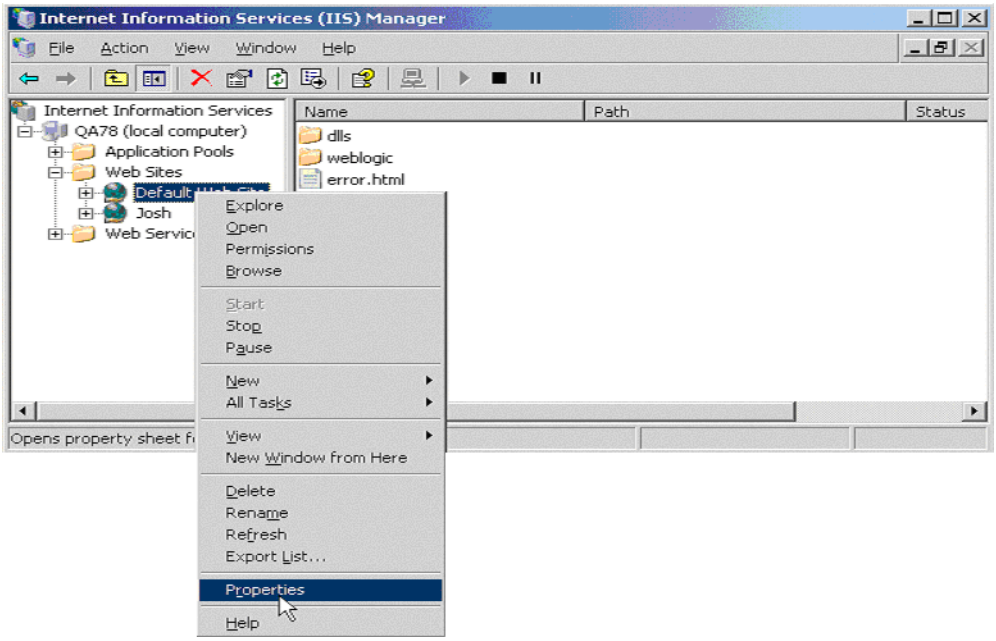
1. Copy the iisproxy.dll file from the *WL_HOME/server/plugin/win/32* or *WL_HOME/server/plugin/win/64* directory of your WebLogic Server installation (where *WL_HOME* is the top-level directory for the WebLogic Platform and Server and contains the WebLogic Server installation files into a convenient directory that is accessible to IIS). This directory must also contain the *iisproxy.ini* file that you will create in step 4. 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.
2. 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 3.)
   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 website (the default is "Default Web Site").

Figure 4–1 Selecting Web Site in Service Manager



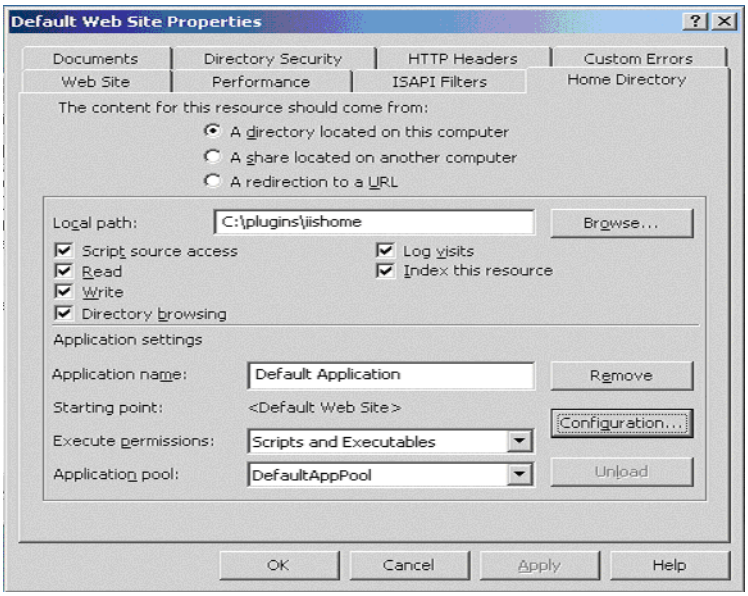
   c. Click the "Play" arrow in the toolbar to start.
   d. Open the properties for the selected website by right-clicking the website selection in the left panel and selecting Properties.

Figure 4–2 Selecting Properties for Selected Web Site



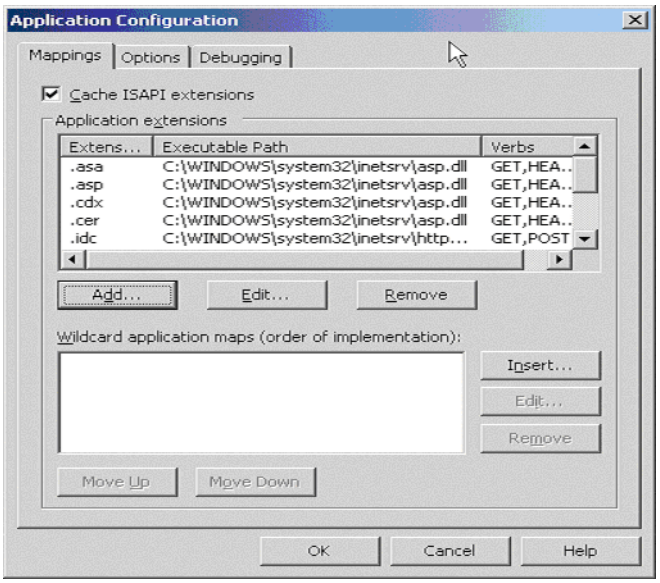
e. In the Properties panel, select the Home Directory tab, and click the Configuration button in the Applications Settings section.

Figure 4–3 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 4–4 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.

---

3. If you want to configure proxying by path complete this step. (In addition to proxying by file type, you can configure the Microsoft Internet Information Server

Plug-In to serve files based on their path by specifying some additional parameters in the *iisproxy.ini* file.) Proxying by path takes precedence over proxying by MIME type.

You can also proxy multiple websites defined in IIS by path. For more information, see "Proxying Requests from Multiple Virtual Websites to WebLogic Server" on page 4-14.

To configure proxying by path:

a. Start the Internet Information Service Manager by selecting it from the Start menu.

b. Place the *iisforward.dll* file in the same directory as the *iisproxy.dll* file and add the *iisforward.dll* file as a filter service in IIS (WebSite Properties ->ISAPI Filters tab -> Add the iisforward dll). Set the user permissions for the *iisforward.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.

c. Register .wlforward as a special file type to be handled by *iisproxy.dll* in IIS.

d. Define the property `WlForwardPath` in *iisproxy.ini*. `WlForwardPath` defines the path that is proxied to WebLogic Server, for example: `WlForwardPath=/weblogic`.

e. Set the `PathTrim` parameter to trim off the `WlForwardPath` when necessary. For example, using

```
WlForwardPath=/weblogic
PathTrim=/weblogic
```

trims a request from IIS to Weblogic Server. Therefore, /weblogic/session is changed to /session.

f. If you want requests that do not contain extra path information (in other words, requests containing only a host name), set the `DefaultFileName` parameter to the name of the welcome page of the Web Application to which the request is being proxied. The value of this parameter is appended to the URL.

g. If you need to debug your application, set the Debug=ON parameter in *iisproxy.ini*. A *c:\tmp\iisforward.log* is generated containing a log of the plug-in's activity that you can use for debugging purposes.

4. 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 " General Parameters for Web Service Plug-Ins" on page 7-1.

Use the example *iisproxy.ini* file in "Sample iisproxy.ini File" on page 4-15 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.

5. 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.

6. Optionally, enable HTTP tunneling by following the instructions for proxying by path (see step 8 above), 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.

   a. If you are using weblogic.jar and the T3 protocol, set `WlForwardPath` to this URL pattern:

      ```
      WlForwardPath=*/HTTPClnt*
      ```

   b. If you are using IIOP, which is the only protocol used by the WebLogic Server thin client, *wlclient.jar*, set the value of WlForwardPath to */iiop*:

      ```
      WlForwardPath=*/iiop*
      ```

   You do not need to use the `PathTrim` parameter.

7. Set any additional parameters in the *iisproxy.ini* file. A complete list of parameters is available in the appendix "General Parameters for Web Service Plug-Ins" on page 7-1.

8. If you are proxying servlets from IIS to WebLogic Server and you are not proxying by path, read the section "Proxying Servlets from IIS to WebLogic Server" on page 4-17.

9. 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.

4.5 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:

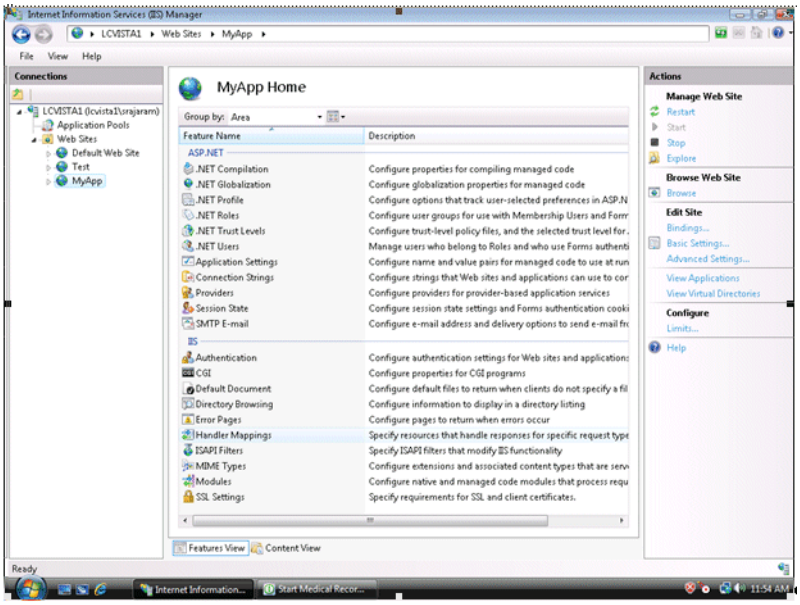
1. Create a web application in IIS Manager by right clicking on Web Sites -> Add Web Site.

   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.

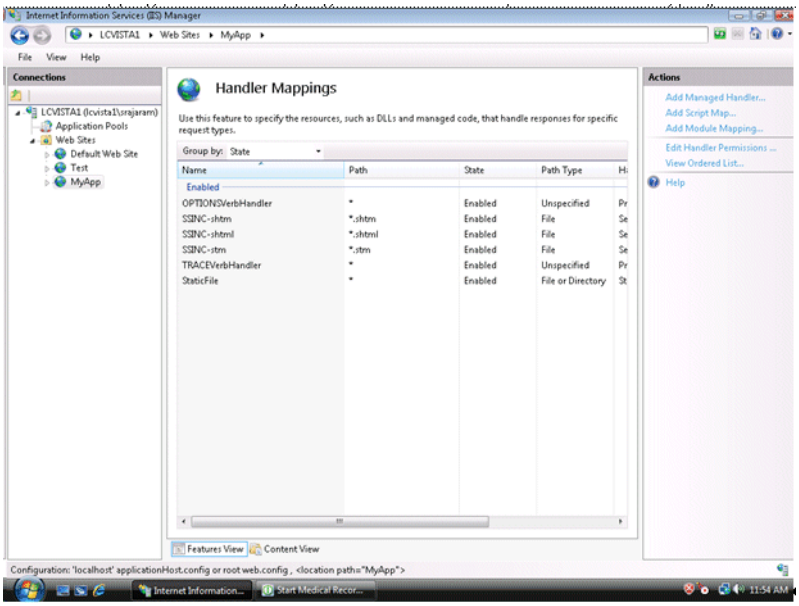
   If you can see the name of your application under Web Sites it means that your application has been created and started running. Click on the MyApp node under Web Sites to see all of the settings related to the MyApp application, which you can change, as shown in Figure 4–5.

Figure 4–5 Application Home Page



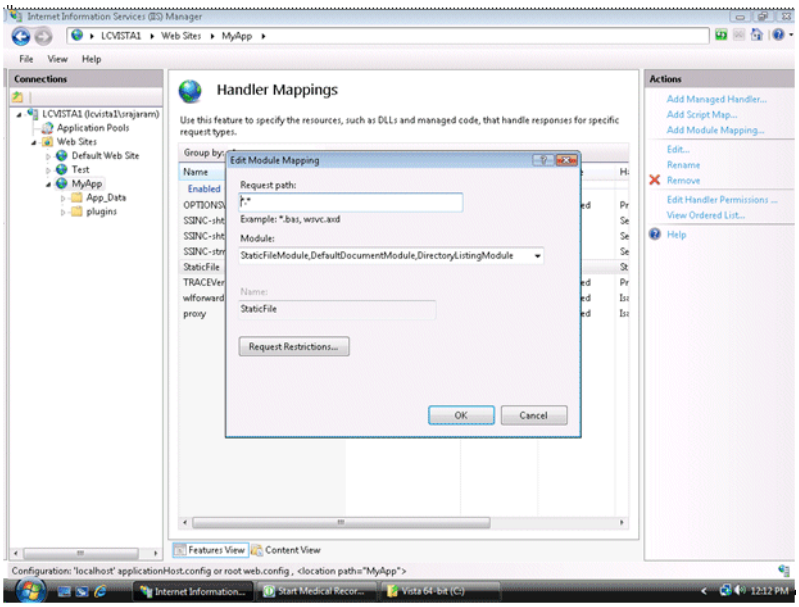
2. Click on "Handler Mappings" to set the mappings to the handler for a particular MIME type.

Figure 4–6 Setting the Handler Mappings



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

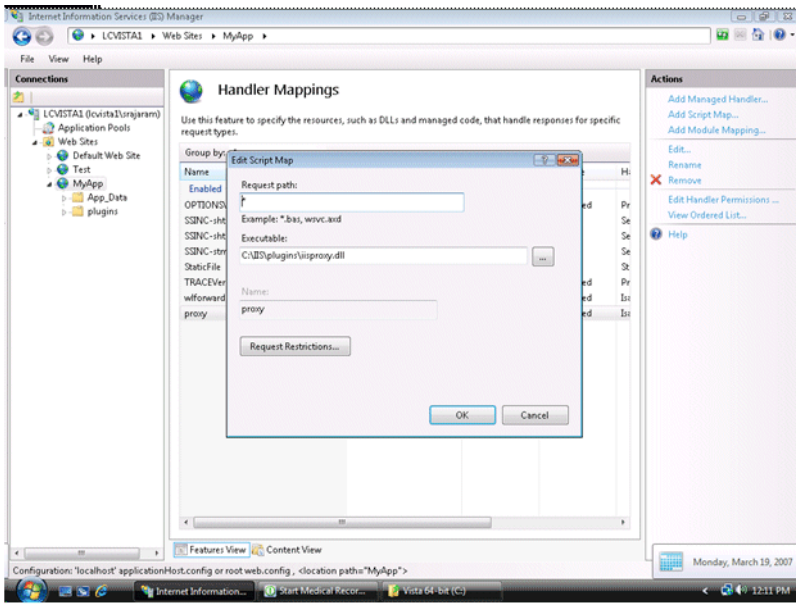
Figure 4–7 Editing the Request Path for Module



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

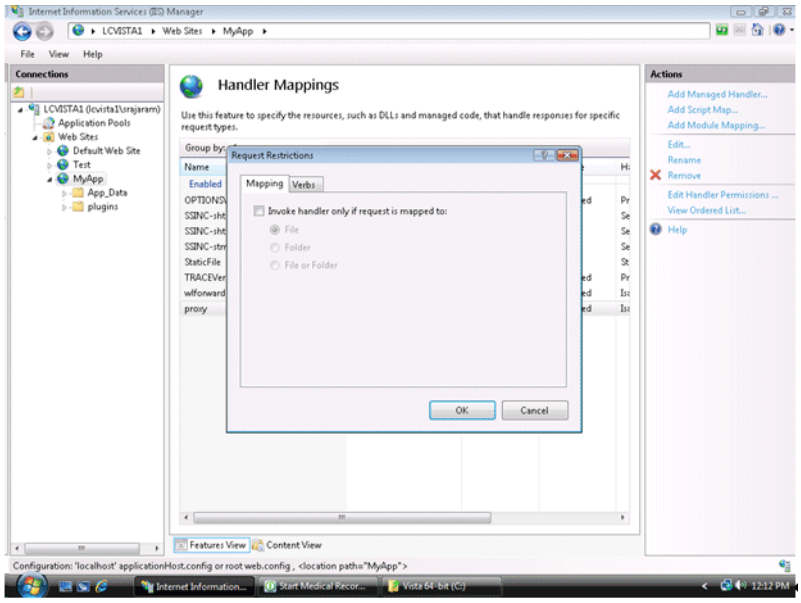
   Browse to the *iisproxy.dll* file and add it as the executable. Name it *proxy*.

Figure 4–8 Editing the Request Path for Script



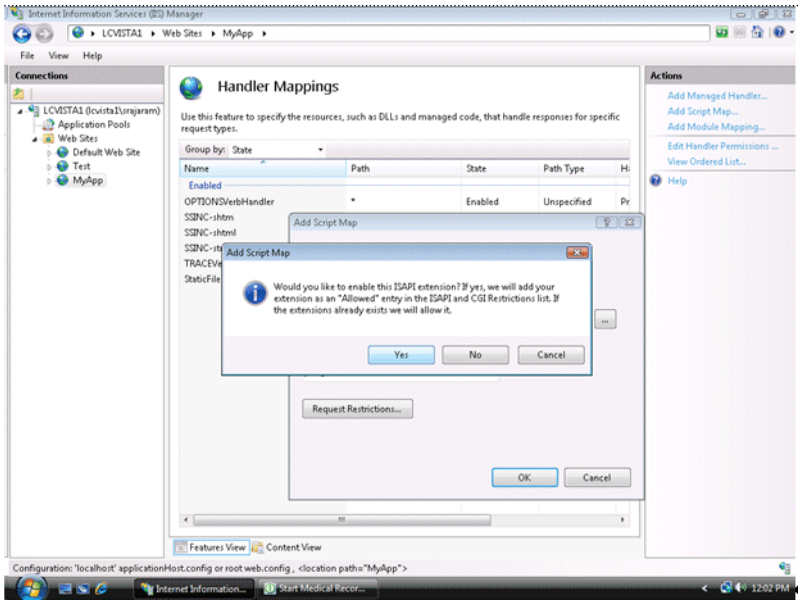
5. Click on the "Request Restrictions…" button and uncheck the box "Invoke handler only if the request is mapped to".

Figure 4–9 Editing the Request Restrictions



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

Figure 4–10 Adding the Script Map

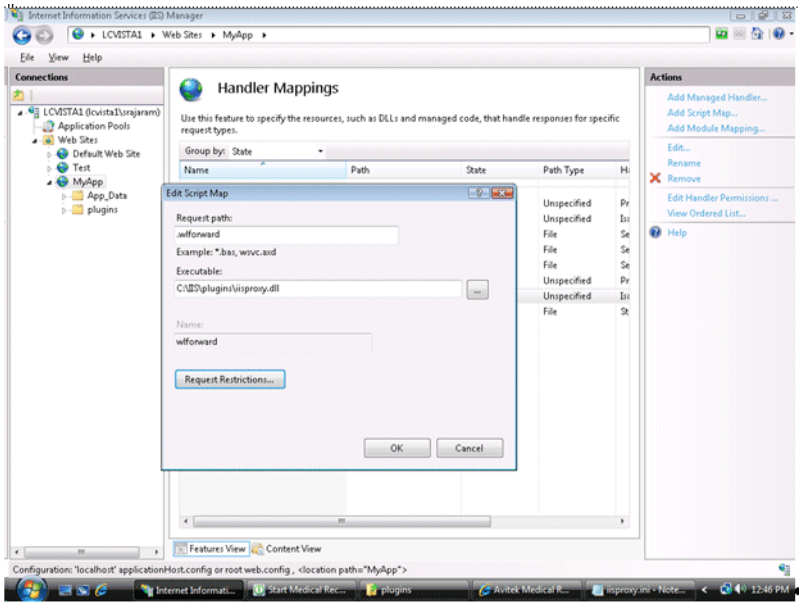


7. If you want to configure proxying by path, click on Add Script Map. Provide the Request path as **.wlforward* and select the executable as *iisproxy.dll*.

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

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

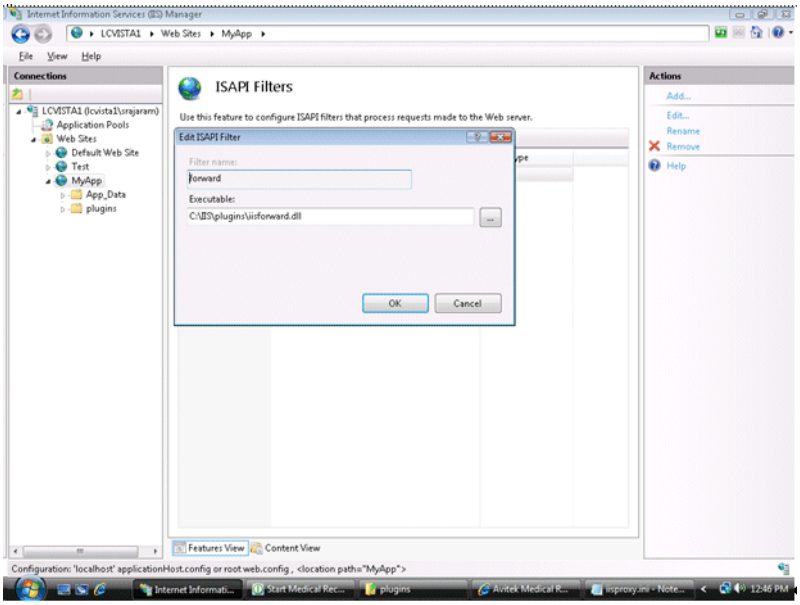
Figure 4–11 Adding the Script Map for Proxying by Path



8. Click on the *MyApp* application on the left-hand side and click on ISAPI Filters. Click Add to add the ISAPI filter.

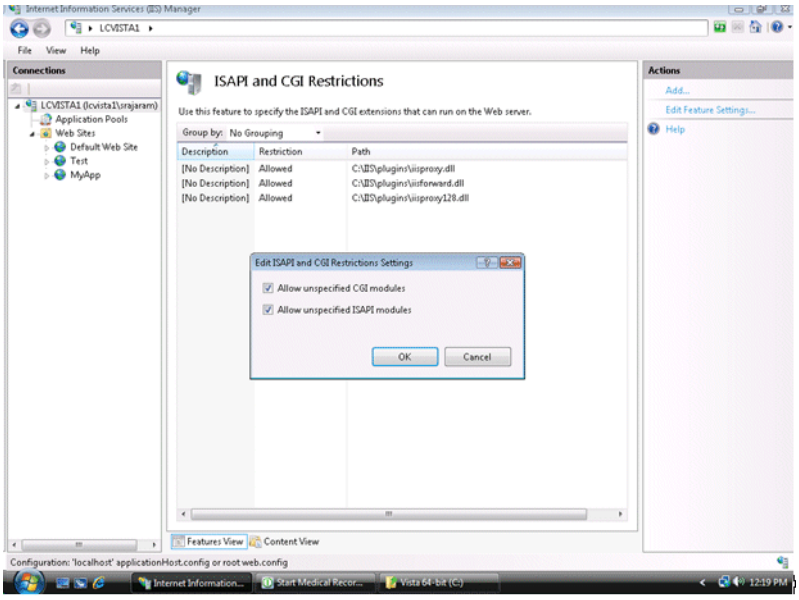
Enter the Filter name as *forward* and select the executable as *iisforward.dll*. Click OK.

Figure 4–12 Editing ISAPI Filters



9. 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 4–13 Editing ISAPI and CGI Restrictions



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

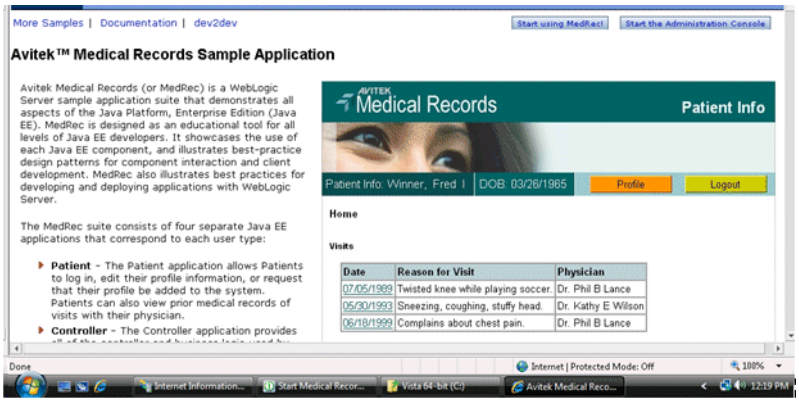
```
WebLogicHost= @hostname@
WebLogicPort= @port@
```

```
ConnectRetrySecs=5
ConnectTimeoutSecs=25
Debug=ALL
DebugConfigInfo=ON
KeepAliveEnabled=true


WlForwardPath=/weblogic
PathTrim=/weblogic

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

11. 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 4–14 Medrec Sample Application



4.6 Proxying Requests from Multiple Virtual Websites to WebLogic Server

To proxy requests from multiple websites (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. Copy *iisforward.dll* to the directory you created in step1.
3. Register the *iisforward.dll* for each website with IIS.
4. Create a file called *iisforward.ini*. Place this file in the same directory that contains *iisforward.dll*. This file should contain the following entry for each virtual website defined in IIS:

   ```
   vhostN=websiteName:port
   websiteName:port=dll_directory/iisproxy.ini
   ```

   Where:

- N is an integer representing the virtual website. The first virtual website you define should use the integer 1 and each subsequent website should increment this number by 1.
- `websiteName` is the name of the virtual website as registered with IIS.
- `port` is the port number where IIS listens for HTTP requests.
- `dll_directory` is the path to the directory you created in step 1.

For example:

```
vhost1=strawberry.com:7001
strawberry.com:7001=c:\strawberry\iisproxy.ini
vhost2=blueberry.com:7001
blueberry.com:7001=c:\blueberry\iisproxy.ini
...
```

5. Create an *iisproxy.ini* file for the virtual eebsites, as described in step 4 in "Proxying Requests" on page 4-2. Copy this *iispoxy.ini* file to the directory you created in step 1.
6. Copy *iisproxy.dll* to the directory you created in step 1.
7. In IIS, set the value for the Application Protection option to high (isolated). If the Application Protection option is set to Medium(pooled), the iisproxy.dll that registered as the first website will always be invoked. In this event, all the requests will be proxied to the same WebLogic Server instances defined in the *iisproxy.ini* of the first website.

4.6.1 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.

4.7 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”.

4.8 Setting Up Perimeter Authentication

Use perimeter authentication to secure your WebLogic Server applications that are accessed via the Microsoft Internet Information Server 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 Microsoft Internet Information Server 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 *Oracle Fusion Middleware Developing Security Providers for Oracle WebLogic Server*.
2. Configure the custom Identity Assertion Provider to support the "Cert" token type and make it the active token type. See How to Create New Token Types in *Oracle Fusion Middleware Developing Security Providers for Oracle WebLogic Server*.
3. Set the `clientCertProxy` attribute 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). See context-param in *Oracle Fusion Middleware 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 Microsoft Internet Information Server Plug-In is running. See Using Network Connection Filters in *Oracle Fusion Middleware 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. Use Sun Microsystems' keytool utility to export a trusted Certificate Authority file from the *DemoTrust.jks* keystore file that resides in *WL_HOME/server/lib*.
   a. To extract the wlsdemoca file, for example, use the command:

      ```
      keytool -export -file trustedcafile.der -keystore DemoTrust.jks -alias
      wlsdemoca
      ```

      Change the alias name to obtain a different trusted CA file from the keystore.

      To look at all of the keystore's trusted CA files, use: `keytool -list -keystore DemoTrust.jks`.

      Press enter if prompted for password.

b. To convert the Certificate Authority file to pem format: `java utils.der2pem trustedcafile.der`.

See Identity Assertion Providers in *Oracle Fusion Middleware Developing Security Providers for Oracle WebLogic Server* for more information about Identity Assertion Providers.

4.9 Using SSL with the Microsoft Internet Information Server Plug-In

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between WebLogic Server and the Microsoft Internet Information Server Plug-In. The SSL protocol provides confidentiality and integrity to the data passed between the Microsoft Internet Information Server Plug-In and WebLogic Server.

The Microsoft Internet Information Server Plug-In does not use the transport protocol (http or https) to determine whether the SSL protocol will be used to protect the connection between the proxy plug-in and the Microsoft Internet Information Server. In order to use the SSL protocol with the Microsoft Internet Information Server Plug-In, configure the WebLogic Server instance receiving the proxied requests to use the SSL protocol. The port on the WebLogic Server that is configured for secure SSL communication is used by the Microsoft Internet Information Server Plug-In to communicate with the Microsoft Internet Information Server.

To use the SSL protocol between Microsoft Internet Information Server Plug-In and WebLogic Server:

1. Configure WebLogic Server for SSL. For more information, see Configuring SSL.
2. Configure the WebLogic Server SSL listen port. For more information, see Configuring SSL.
3. Set the WebLogicPort parameter in the *iisproxy.ini* file to the listen port configured in step 2.
4. Set the SecureProxy parameter in the *iisproxy.ini* file to ON.
5. Set additional parameters in the *iisproxy.ini* file that define the SSL connection. For a complete list of parameters, see "SSL Parameters for Web Server Plug-Ins" on page 7-14.

For example:

```
WebLogicHost=myweblogic.com
WebLogicPort=7002
SecureProxy=ON
```

4.10 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 "Installing and Configuring the Microsoft Internet Information Server Plug-In" on page 4-4.

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.

4.11 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
   ```

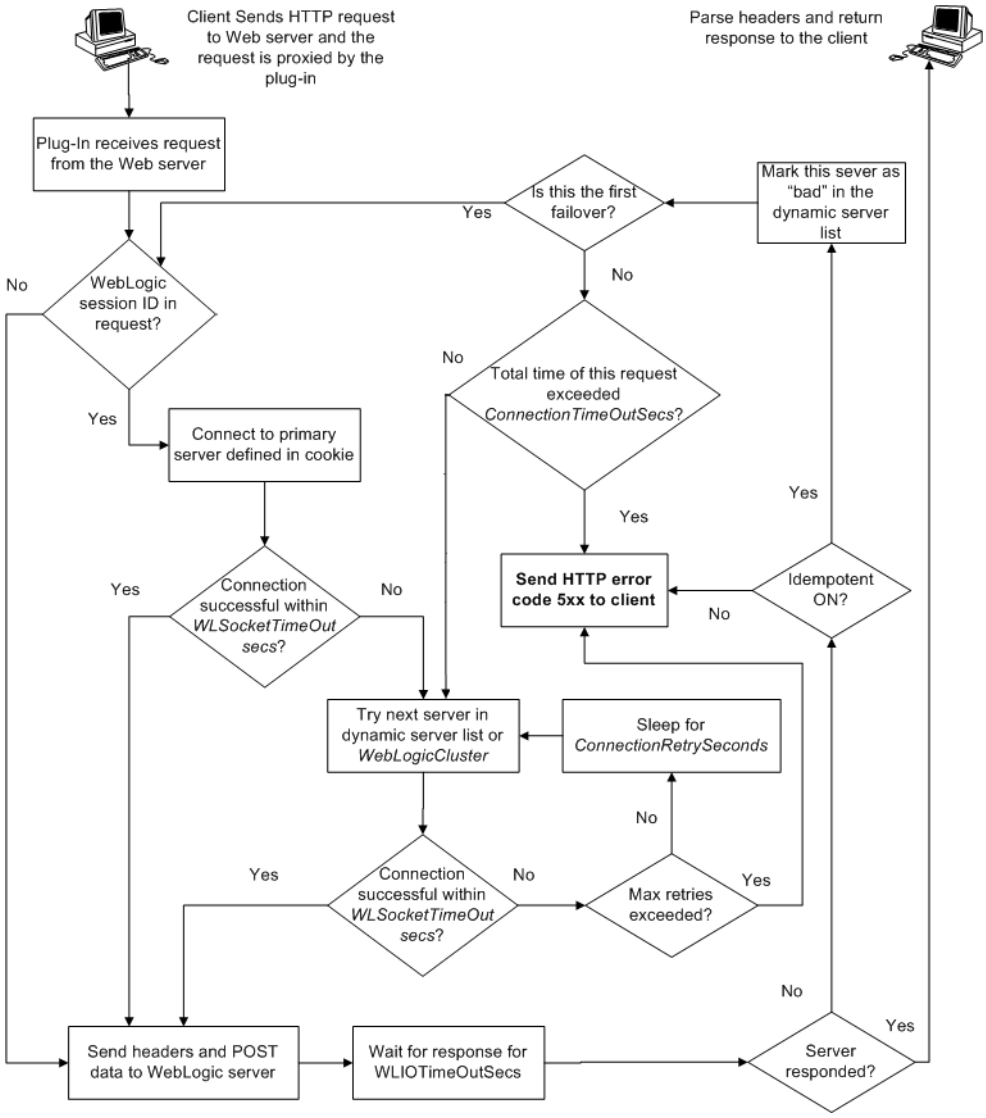
   If filename.jsp is displayed in your browser, the plug-in is functioning.

4.12 Connection Errors and Clustering Failover

When the Microsoft Internet Information Server 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 sends the request to other WebLogic Servers in the cluster. If the connection fails or there is no response from any WebLogic Server instance in the cluster, an error message is sent.

Figure 4–15 demonstrates how the plug-in handles failover.

Figure 4–15 Connection Failover



4.12.1 Possible Causes of Connection Failures

Failure of the WebLogic Server host to respond to a connection request could indicate problems with the host machine, networking problems, or other server failures.

Failure of any WebLogic Server instance in the cluster to respond, could indicate that WebLogic Server is not running or is unavailable, a hung server, a database problem, or other application failure.

4.12.2 Failover with a Single, Non-Clustered WebLogic Server

If you are running only a single WebLogic Server, 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 WebLogic Server as determined by the *ConnectTimeoutSecs* and *ConnectRetrySecs* parameters.

4.12.3 The Dynamic Server List

When you specify a list of WebLogic Servers in the WebLogicCluster parameter, 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.

4.12.4 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. For more information see Figure 4–15.

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.

5

Installing and Configuring the Sun Java System Web Server Plug-In

This release documents how to install and configure the Sun Java System Web Server plug-in.

In previous releases of WebLogic Server, this plug-in was referred to as the Netscape Enterprise Server plug-in. References to file specifications in this chapter continue to use the Netscape Enterprise Server nomenclature.

The following sections describe how to install and configure the Sun Java System Web Server proxy plug-in:

5.1 Overview of the Sun Java System Web Server Plug-In

The Sun Java System Web Server Plug-In enables requests to be proxied from Sun Java System Web Server to WebLogic Server. The plug-in enhances a Sun Java System Web Server installation by allowing WebLogic Server to handle those requests that require the dynamic functionality of WebLogic Server.

The Sun Java System Web Server Plug-In is designed for an environment where Sun Java System Web Server serves static pages, and a Weblogic Server instance (operating in a different process, possibly on a different machine) is delegated to serve dynamic pages, such as JSPs or pages generated by HTTP Servlets. The connection between WebLogic Server and the Sun Java System Web Server Plug-In is made using clear text or Secure Sockets Layer (SSL). To the end user—the browser—the HTTP requests delegated to WebLogic Server appear to come from the same source as the static pages. Additionally, the HTTP-tunneling facility of WebLogic Server can operate through the Sun Java System Web Server Plug-In, providing access to all WebLogic Server services (not just dynamic pages).

The Sun Java System Web Server Plug-In operates as a module within a Sun Java System Web Server. The module is loaded at startup, and then certain HTTP requests are delegated to it. The module is similar to an HTTP (Java) servlet, except that a module is written in code native to the platform.

For more information on supported versions of Sun Java System Web Server see http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html.

5.1.1 Connection Pooling and Keep-Alive

The WebLogic Server Sun Java System Web Server Plug-In provides efficient performance by using a re-usable pool of connections from the plug-in to WebLogic Server. The plug-in automatically implements "keep-alive" connections between the plug-in and WebLogic Server. If a connection is inactive for more than 30 seconds or a user-defined amount of time, the connection is closed. You can disable this feature if desired. For more information, see `KeepAliveEnabled` in Table 7–1.

5.1.2 Proxying Requests

The plug-in proxies requests to WebLogic Server based on a configuration that you specify. You can proxy requests based on the URL of the request (or a portion of the URL). This is called proxying by *path*. You can also proxy request based on the MIME type of the requested file. Or you can use a combination of both methods. If 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. For more information, see "Installing and Configuring the Sun Java System Web Server Plug-In" on page 5-2.

Note: The request processing behavior has changed in Sun Java System Web Server 7.0 Update 2 release. See http://wikis.sun.com/display/WebServerdocs/Release+Notes, issue 6747123.

5.2 Installing and Configuring the Sun Java System Web Server Plug-In

To install and configure the Sun Java System Web Server Plug-In:

1. Copy the library.

   The WebLogic Sun Java System Web Server plug-in module is distributed as a shared object (.so) on UNIX platforms and as a dynamic-link library (.dll) on Windows. These files are located in the *WL_HOME/server/plugin/OperatingSystem/Architecture* directory of your WebLogic Server distribution. WL_HOME represents the top level installation directory for your WebLogic platform. The server directory contains installation files for WebLogic Server. OperatingSystem refers to the operating system, such as UNIX or Windows.

   Choose the appropriate library file for your environment from based on http://www.oracle.com/technology/software/products/ias/files/fusion_certification.html and copy that file into the file system where Sun Java System Web Server is located.

   The Sun Java System Web Server 7.0 can use the plug-in named *xxx_61*. For example, *libproxy128_61.so* and *libproxy_61.so*.

2. Read "Guidelines for Modifying the obj.conf File" on page 5-7, then modify the *obj.conf* file as described in the following steps. The *obj.conf* file defines which requests are proxied to WebLogic Server and other configuration information.

3. Locate and open *obj.conf*.

The obj.conf file for your instance is in the following location:

```
NETSCAPE_HOME/https-INSTANCE_NAME/config/obj.conf
```

Where *NETSCAPE_HOME* is the root directory of the installation, and *INSTANCE_NAME* is the particular "instance" or server configuration that you are using. For example, on a UNIX machine called myunixmachine, the *obj.conf* file would be found here:

```
/usr/local/netscape/enterprise-351/
 https-myunixmachine/config/obj.conf
```

4. Instruct Sun Java System Web Server to load the native library (the .so or .dll file) as a module.

   To use iPlanet 4.x or earlier, add the following lines to the beginning of the obj.conf file.

   ```
   Init fn="load-modules" funcs="wl_proxy,wl_init"\
   shlib=/usr/local/netscape/plugins/SHARED_LIBRARY
   Init fn="wl_init"
   ```

   Where *SHARED_LIBRARY* is the shared object or dll (for example *libproxy.so*) that you installed in step 1 under "Installing and Configuring the Sun Java System Web Server Plug-In" on page 5-2. The function *load-modules* tags the shared library for loading when Sun Java System Web Server starts up. The values *wl_proxy* and *wl_init* identify the functions that the Sun Java System Web Server Plug-In executes.

   To use iPlanet 6.0, add the following lines to the beginning of the *magnus.conf* file. These lines instruct Sun Java System Web Server to load the native library (the .so or .dll file) as a module:

   > **Note:** Spacing is important. There must be a space between the " and \, or there must be a leading space before `shlib`.

   ```
   Init fn="load-modules" funcs="wl_proxy,wl_init"\
    shlib=/usr/local/netscape/plugins/SHARED_LIBRARY
   Init fn="wl_init"
   ```

   Where *SHARED_LIBRARY* is the shared object or dll (for example libproxy.so) that you installed in step 1 under "Installing and Configuring the Sun Java System Web Server Plug-In" on page 5-2. The function *load-modules* tags the shared library for loading when Sun Java System Web Server starts up. The values `wl_proxy` and `wl_init` identify the functions that the Sun Java System Web Server Plug-In executes.

5. If you want to proxy requests by URL, (also called proxying by path.) create a separate `<Object>` tag for each URL that you want to proxy and define the `PathTrim` parameter. (You can proxy requests by MIME type, in addition to or instead of proxying requests by path. See step 6 Proxying by path supersedes proxying by MIME type.) The following is an example of an `<Object>` tag that proxies a request containing the string */weblogic/**.

   ```
   <Object name="weblogic" ppath="*/weblogic/*">
   Service fn=wl_proxy WebLogicHost=myserver.com\
    WebLogicPort=7001 PathTrim="/weblogic"
   </Object>
   ```

   To create an `<Object>` tag to proxy requests by URL:

a. Specify a name for this object (optional) inside the opening `<Object>` tag using the name attribute. The name attribute is informational only and is not used by the Sun Java System Web Server Plug-In. For example:

```
<Object name=myObject ...>
```

b. Specify the URL to be proxied within the `<Object>` tag, using the `ppath` attribute. For example:

```
<Object name=myObject ppath="*/weblogic/*>
```

The value of the `ppath` attribute can be any string that identifies requests intended for Weblogic Server. When you use a `ppath`, every request that contains that path is redirected. For example, a ppath of */weblogic/** redirects every request that begins *http://enterprise.com/weblogic* to the Sun Java System Web Server Plug-In, which sends the request to the specified Weblogic host or cluster.

c. Add the Service directive within the `<Object>` and `</Object>` tags. In the Service directive you can specify any valid parameters as name=value pairs. Separate multiple name=value pairs with one and only one space. For example:

```
Service fn=wl_proxy WebLogicHost=myserver.com\
 WebLogicPort=7001 PathTrim="/weblogic"
```

For a complete list of parameters, see "General Parameters for Web Service Plug-Ins" on page 7-1. You must specify the following parameters:

For a non-clustered WebLogic Server: the WebLogicHost and WebLogicPort parameters.

For a cluster of WebLogic Server instances: the WebLogicCluster parameter.

Always begin the Service directive with Service *fn=wl_proxy*, followed by valid name=value pairs of parameters.

Here is an example of the object definitions for two separate ppaths that identify requests to be sent to different instances of WebLogic Server:

```
<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy WebLogicHost=myserver.com\
 WebLogicPort=7001 PathTrim="/weblogic"
</Object>
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy WebLogicHost=otherserver.com\
 WebLogicPort=7008
</Object>
```

Note: Parameters that are not required, such as PathTrim, can be used to further configure the way the ppath is passed through the Sun Java System Web Server Plug-In. For a complete list of plug-in parameters, see "General Parameters for Web Service Plug-Ins" on page 7-1.

---

6. If you are proxying requests by MIME type, add any new MIME types referenced in the *obj.conf* file to the MIME.types file. You can add MIME types by using the Netscape server console or by editing the MIME.types file directly.

To directly edit the *MIME.types* file, open the file for edit and type the following line:

```
type=text/jsp        exts=jsp
```

Note: For Netscape Enterprise Server 4.0 (iPlanet), instead of adding the MIME type for JSPs, change the existing MIME type from magnus-internal/jsp to text/jsp.

To use the Netscape console, select Manage PreferencesÆ Mime Types, and make the additions or edits.

7. All requests with a designated MIME type extension (for example, .jsp) can be proxied to the WebLogic Server, regardless of the URL. To proxy all requests of a certain file type to WebLogic Server:

   a. Add a Service directive to the existing default Object definition. (`<Object name=default ...>`)

      For example, to proxy all JSPs to a WebLogic Server, the following Service directive should be added after the last line that begins with:

      ```
      NameTrans fn=....
      ```

      and before the line that begins with:

      ```
      PathCheck.
      Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
       WebLogicHost=192.1.1.4 WebLogicPort=7001 PathPrepend=/jspfiles
      ```

      This Service directive proxies all files with the .jsp extension to the designated WebLogic Server, where they are served with a URL like this:

      ```
      http://WebLogic:7001/jspfiles/myfile.jsp
      ```

      The value of the `PathPrepend` parameter should correspond to the context root of a Web Application that is deployed on the WebLogic Server or cluster to which requests are proxied.

      After adding entries for the Sun Java System Web Server Plug-In, the default Object definition will be similar to the following example:

      ```
      <Object name=default>
      NameTrans fn=pfx2dir from=/ns-icons\
       dir="c:/Netscape/SuiteSpot/ns-icons"
      NameTrans fn=pfx2dir from=/mc-icons\
       dir="c:/Netscape/SuiteSpot/ns-icons"
      NameTrans fn="pfx2dir" from="/help" dir=\
       "c:/Netscape/SuiteSpot/manual/https/ug"
      NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
      Service method="(GET|HEAD|POST|PUT)" type=text/jsp\
       fn=wl_proxy WebLogicHost=localhost WebLogicPort=7001\
      PathPrepend=/jspfiles
      PathCheck fn=nt-uri-clean
      PathCheck fn="check-acl" acl="default"
      PathCheck fn=find-pathinfo
      PathCheck fn=find-index index-names="index.html,home.html"
      If a required parameter is missing from the configuration, when the object
      is invoked it issues an HTML error that notes the missing parameter from
      the configuration.
      ```

```
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\  fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) \
 type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>
```

b. Add a similar Service statement to the default object definition for all other MIME types that you want to proxy to WebLogic Server.

c. To configure proxy-by-MIME for the JSP, you must add the following entry to the mime.types file

```
type=text/jsp exts=jsp
```

For proxy-by-MIME to work properly you need to disable JAVA from the Sun One Web Server otherwise SUN One will try to serve all requests that end in *.jsp and will return a 404 error as it will fail to locate the resource under $doc_root.

To disable JAVA from the Sun One Web Server, comment out the following in the *obj.conf* file under the name="default"#NameTrans fn="ntrans-j2ee" name="j2ee" and restart the webserver.

8. Optionally, if you are proxying by path, enable HTTP-tunneling:

a. If you are using weblogic.jar and tunneling the t3 protocol, add the following object definition to the *obj.conf* file, 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.

```
<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicHost=192.192.1.4\  WebLogicPort=7001
</Object>
```

b. If you are tunneling IIOP, which is the only protocol used by the WebLogic Server thin client, *wlclient.jar*, add the following object definition to the *obj.conf* file, 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.

```
<Object name="tunnel" ppath="*/iiop*">
Service fn=wl_proxy WebLogicHost=192.192.1.4\  WebLogicPort=7001
</Object>
```

9. Deploy and test the Sun Java System Web Server Plug-In

a. Start WebLogic Server.

b. Start Sun Java System Web Server. If Sun Java System Web Server is already running, you must either restart it or apply the new settings from the console in order for the new settings to take effect.

c. To test the Sun Java System Web Server Plug-In, open a browser and set the URL to the Sun Java System Web Server plus */weblogic/*, which should bring up the default WebLogic Server HTML page, welcome file, or default servlet, as defined for the default Web Application as shown in this example:

```
http://myenterprise.server.com/weblogic/
```

For information on how to create a default Web Application, see *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.

5.2.1 Guidelines for Modifying the obj.conf File

To use the Sun Java System Web Server Plug-In, you must make several modifications to the *obj.conf* file. These modifications specify how requests are proxied to WebLogic Server. You can proxy requests by URL or by MIME type. The procedure for each is described in "Installing and Configuring the Sun Java System Web Server Plug-In" on page 5-2.

The Netscape *obj.conf* file is very strict about the placement of text. To avoid problems, note the following regarding the *obj.conf* file:

- Eliminate extraneous leading and trailing white space. Extra white space can cause your Netscape server to fail.
- If you must enter more characters than you can fit on one line, place a backslash (\) at the end of that line and continue typing on the following line. The backslash directly appends the end of the first line to the beginning of the following line. If a space is necessary between the words that end the first line and begin the second line, be certain to use one space, either at the end of the first line (before the backslash), or at the beginning of the second line.
- Do not split attributes across multiple lines. (For example, all servers in a cluster must be listed in the same line, following WebLogicCluster.)

5.2.2 Sample obj.conf File (Not Using a WebLogic Cluster)

Below is an example of lines that should be added to the *obj.conf* file if you are not using a cluster. You can use this example as a template that you can modify to suit your environment and server. Lines beginning with # are comments.

Note: Make sure that you do not include any extraneous white space in the obj.conf file. Copying and pasting from the samples below sometimes adds extra white space, which can create problems when reading the file.

You can read the full documentation on Enterprise Server configuration files in the Sun Java System Web Server documentation.

```
## ------------- BEGIN SAMPLE OBJ.CONF CONFIGURATION  ---------
# (no cluster)
# The following line locates the NES library for loading at
# startup, and identifies which functions within the library are
# NES functions.  Verify the path to the library (the value
# of the shlib=<...> parameter) and that the file is
# readable, or the server fails to start.
Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"
# Configure which types of HTTP requests should be handled by the
# NES module (and, in turn, by WebLogic).  This is done
# with one or more "<Object>" tags as shown below.
# Here we configure the NES module to pass requests for
# "/weblogic" to a WebLogic Server listening at port 7001 on
```

```
# the host myweblogic.server.com.
<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy WebLogicHost=myweblogic.server.com\
 WebLogicPort=7001 PathTrim="/weblogic"
</Object>
# Here we configure the plug-in so that requests that
# match "/servletimages/" is handled by the
# plug-in/WebLogic.
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>
# This Object directive works by file extension rather than
# request path. To use this configuration, you must also add
# a line to the mime.types file:
#
# type=text/jsp             exts=jsp
#
# This configuration means that any file with the extension
# ".jsp" are proxied to WebLogic. Then you must add the
# Service line for this extension to the Object "default",
# which should already exist in your obj.conf file:
<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicHost=localhost WebLogicPort=7001 PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\  fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>
# The following directive enables HTTP-tunneling of the
# WebLogic protocol through the NES plug-in.
<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>
#
## ------------- END SAMPLE OBJ.CONF CONFIGURATION ---------
```

5.2.3 Sample obj.conf File (Using a WebLogic Cluster)

Below is an example of lines that should be added to *obj.conf* if you are using a WebLogic Server cluster. You can use this example as a template that you can modify to suit your environment and server. Lines beginning with # are comments.

Note: Make sure that you do not include any extraneous white space in the *obj.conf* file. Copying and pasting from the samples below sometimes adds extra white space, which can create problems when reading the file.

For more information, see the full documentation on Enterprise Server configuration files from Netscape.

```
## ------------- BEGIN SAMPLE OBJ.CONF CONFIGURATION ---------
# (using a WebLogic Cluster)
#
# The following line locates the NES library for loading at
# startup, and identifies which functions within the library are
# NES functions.  Verify the path to the library (the value
# of the shlib=<...> parameter) and that the file is
# readable, or the server fails to start.
Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"
# Configure which types of HTTP requests should be handled by the
# NES module (and, in turn, by WebLogic).  This is done
# with one or more "<Object>" tags as shown below.
# Here we configure the NES module to pass requests for
# "/weblogic" to a cluster of WebLogic Servers.
<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy \
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001" PathTrim="/weblogic"
</Object>
# Here we configure the plug-in so that requests that
# match "/servletimages/" are handled by the
# plug-in/WebLogic.
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy \
WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001"
</Object>
# This Object directive works by file extension rather than
# request path. To use this configuration, you must also add
# a line to the mime.types file:
#
# type=text/jsp             exts=jsp
#
# This configuration means that any file with the extension
# ".jsp" is proxied to WebLogic. Then you must add the
# Service line for this extension to the Object "default",
# which should already exist in your obj.conf file:
<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001",PathPrepend=/jspfiles
```

```
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\  fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>
# The following directive enables HTTP-tunneling of the
# WebLogic protocol through the NES plug-in.
<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicCluster="myweblogic.com:7001,\
 yourweblogic.com:7001,theirweblogic.com:7001"
</Object>
#
## ------------- END SAMPLE OBJ.CONF CONFIGURATION ---------
```

5.3 Setting Up Perimeter Authentication

Use perimeter authentication to secure your WebLogic Server applications that are accessed via the Sun Java System Web Server 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 Sun Java System Web Server 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 *Oracle Fusion Middleware Developing Security Providers for Oracle WebLogic Server*.
2. Configure the custom Identity Assertion Provider to support the "Cert" token type and make it the active token type. See How to Create New Token Types in *Oracle Fusion Middleware Developing Security Providers for Oracle WebLogic Server*.
3. Set the clientCertProxy attribute 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). See context-param in *Oracle Fusion Middleware 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 Sun Java System Web Server Plug-In is running. See Using Network Connection Filters in *Oracle Fusion Middleware 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. Use Sun Microsystems' keytool utility to export a trusted Certificate Authority file from the DemoTrust.jks keystore file that resides in *WL_HOME/server/lib*.
   a. To extract the *wlsdemoca* file, for example, use the command:

      ```
      keytool -export -file trustedcafile.der -keystore DemoTrust.jks -alias
      wlsdemoca
      ```

      Change the alias name to obtain a different trusted CA file from the keystore.

To look at all of the keystore's trusted CA files, use: `keytool -list -keystore DemoTrust.jks`

Press enter if prompted for password.

b. To convert the Certificate Authority file to pem format: `java utils.der2pem trustedcafile.der`

See Identity Assertion Providers in *Oracle Fusion Middleware Developing Security Providers for Oracle WebLogic Server* for more information about Identity Assertion Providers.

5.4 Using SSL with the Sun Java System Web Server Plug-In

You can use the Secure Sockets Layer (SSL) protocol to protect the connection between the Sun Java System Web Server Plug-In, and WebLogic Server. The SSL protocol provides confidentiality and integrity to the data passed between the Sun Java System Web Server Plug-In and WebLogic Server.

The Sun Java System Web Server 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 the SSL protocol will be used to protect the connection between the Sun Java System Web Server Plug-In and WebLogic Server.

To use the SSL protocol between Sun Java System Web Server Plug-In and WebLogic Server:

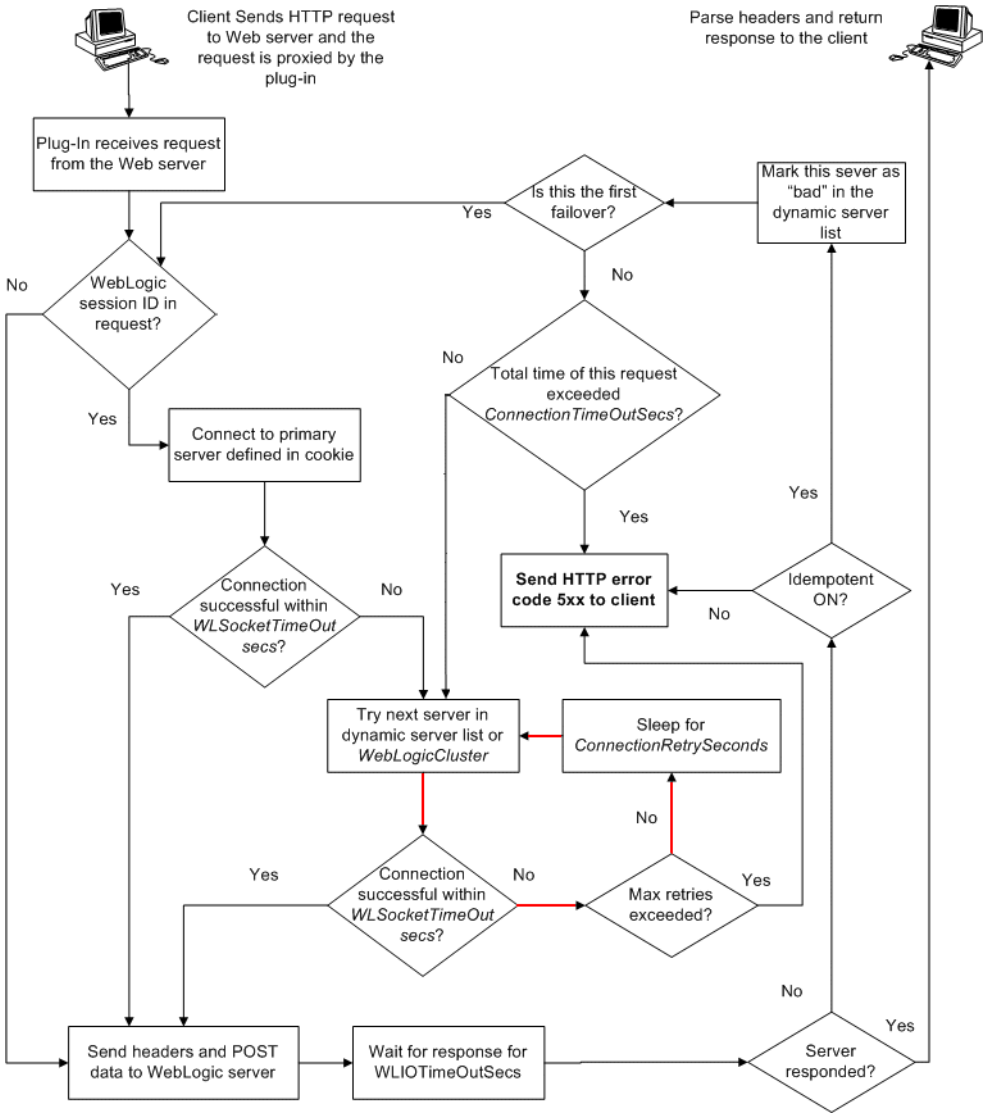
1. Configure WebLogic Server for SSL. For more information, see Configuring SSL.
2. Configure the WebLogic Server SSL listen port. For more information, see Configuring SSL.
3. Set the WebLogicPort parameter in the Service directive in the *obj.conf* file to the listen port configured in step 2.
4. Set the SecureProxy parameter in the Service directive in the *obj.conf* file file to ON.
5. Set additional parameters in the Service directive in the *obj.conf* file that define information about the SSL connection. For a complete list of parameters, see "SSL Parameters for Web Server Plug-Ins" on page 7-14.

5.5 Connection Errors and Clustering Failover

When the Sun Java System Web Server 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 Servers 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 5–1 demonstrates how the plug-in handles failover. The Maximum number of retries allowed in the red loop is equal to *ConnectTimeoutSecs* ÷ *ConnectRetrySecs*.

Figure 5–1 Connection Failover



5.5.1 Possible Causes of Connection Failures

Failure of the WebLogic Server host to respond to a connection request could indicate possible problems with the host machine, networking problems, or other server failures.

Failure of all WebLogic Server instances to respond, could indicate that WebLogic Server is not running or is unavailable, a hung server, a database problem, or other application failure.

5.5.2 Failover with a Single, Non-Clustered WebLogic Server

If you are running a single WebLogic Server instance, the plug-in attempts to connect to that server which is defined with the WebLogicHost parameter. If the attempt fails, an HTTP 503 error message is returned. The plug-in continues trying to connect to WebLogic Server until `ConnectTimeoutSecs` is exceeded.

5.5.3 The Dynamic Server List

When you specify a list of WebLogic Servers in the WebLogicCluster parameter, 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.

5.5.4 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. For more information, see Figure 5–1.

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.

5.5.5 Failover Behavior When Using Firewalls and Load Directors

In most configurations, the Sun Java System Web Server Plug-In sends a request to the primary instance of a cluster. When that instance is unavailable, the request fails over to the secondary instance. However, in some configurations that use combinations of firewalls and load-directors, any one of the servers (firewall or load-directors) can accept the request and return a successful connection while the primary instance of WebLogic Server is unavailable. After attempting to direct the request to the primary instance of WebLogic Server (which is unavailable), the request is returned to the plug-in as “connection reset.”

Requests running through combinations of firewalls (with or without load-directors) are handled by WebLogic Server. In other words, responses of connection reset fail over to a secondary instance of WebLogic Server. Because responses of connection reset fail over in these configurations, servlets must be idempotent. Otherwise duplicate processing of transactions may result.

6 Proxying Requests to Another Web Server

The following sections discuss how to proxy HTTP requests to another Web server:

6.1 Overview of Proxying Requests to Another Web Server

When you use WebLogic Server as your primary Web server, you may also want to configure WebLogic Server to pass on, or proxy, certain requests to a secondary Web server, such as Apache or Microsoft Internet Information Server. Any request that gets proxied is redirected to a specific URL.You can even proxy to another Web server on a different machine.You proxy requests based on the URL of the incoming request.

The HttpProxyServlet (provided as part of the distribution) takes an HTTP request, redirects it to the proxy URL, and sends the response to the client's browser back through WebLogic Server. To use the HttpProxyServlet, you must configure it in a Web Application and deploy that Web Application on the WebLogic Server that is redirecting requests.

6.2 Setting Up a Proxy to a Secondary Web Server

To set up a proxy to a secondary HTTP server:

1. Register the proxy servlet in your Web Application deployment descriptor (see Example 6–1, "Sample web.xml for Use with ProxyServlet"). The Web Application must be the default Web Application of the server instance that is responding to requests. The class name for the proxy servlet is `weblogic.servlet.proxy.HttpProxyServlet`. For more information, see *Oracle Fusion Middleware Developing Web Applications, Servlets, and JSPs for Oracle WebLogic Server*.
2. Define an initialization parameter for the `ProxyServlet` with a `<param-name>` of `redirectURL` and a `<param-value>` containing the URL of the server to which proxied requests should be directed.
3. Optionally, define the following `<KeyStore>` initialization parameters to use two-way SSL with your own identity certificate and key. If no `<KeyStore>` is specified in the deployment descriptor, the proxy will assume one-way SSL.
   - `<KeyStore>` – The key store location in your Web application.

- <KeyStoreType> – The key store type. If it is not defined, the default type will be used instead.
- <PrivateKeyAlias> – The private key alias.
- <KeyStorePasswordProperties> – A property file in your Web application that defines encrypted passwords to access the key store and private key alias. The file contents looks like this:

  ```
  KeyStorePassword={3DES}i4+50LCKenQO8BBvlsXTrg\=\=
  PrivateKeyPassword={3DES}a4TcG4mtVVBRKtZwH3p7yA\=\=
  ```

  You must use the weblogic.security.Encrypt command-line utility to encrypt the password. For more information on the Encrypt utility, as well as the CertGen, and der2pem utilities, see Using the Oracle WebLogic Server Java Utilities in the *Oracle Fusion Middleware Command Reference for Oracle WebLogic Server*.

4. Map the ProxyServlet to a <url-pattern>. Specifically, map the file extensions you wish to proxy, for example *.jsp, or *.html. Use the <servlet-mapping> element in the web.xml Web Application deployment descriptor.

   If you set the <url-pattern> to "/", then any request that cannot be resolved by WebLogic Server is proxied to the remote server. However, you must also specifically map the following extensions: *.jsp, *.html, and *.html if you want to proxy files ending with those extensions.

5. Deploy the Web Application on the WebLogic Server instance that redirects incoming requests.

6.3 Sample Deployment Descriptor for the Proxy Servlet

Example 6–1 is an sample of a Web Applications deployment descriptor for using the Proxy Servlet.

Example 6–1 Sample web.xml for Use with ProxyServlet

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
 //DTD Web Application 2.3//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
<servlet>
  <servlet-name>ProxyServlet</servlet-name>
  <servlet-class>weblogic.servlet.proxy.HttpProxyServlet</servlet-class>
  <init-param>
    <param-name>redirectURL</param-name>
    <param-value>server:port</param-value>
  </init-param>
  <init-param>
    <param-name>KeyStore</param-name>
    <param-value>/mykeystore</param-value>
  </init-param>
  <init-param>
    <param-name>KeyStoreType</param-name>
    <param-value>jks</param-value>
  </init-param>
  <init-param>
    <param-name>PrivateKeyAlias</param-name>
    <param-value>passalias</param-value>
  </init-param>
```

```
  <init-param>
    <param-name>KeyStorePasswordProperties</param-name>
    <param-value>mykeystore.properties</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>
</web-app>
```

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:

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 4, "Installing and Configuring the Microsoft IIS Plug-In"

7.2 General Parameters for Web Service Plug-Ins

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

Table 7–1 General Parameters for Web Service 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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet

Table 7–1 (Cont.) General Parameters for Web Service 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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet
`WebLogicCluster` (Required when proxying to a cluster of WebLogic Servers.)	none	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 4, "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 and NSAPI plug-in,and HttpClusterServlet

Table 7–1 (Cont.) General Parameters for Web Service 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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet
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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet

Table 7–1 (Cont.) General Parameters for Web Service 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.	NSAPI, ISAPI, and Apache plug-in, and HttpClusterServlet
`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.	NSAPI, ISAPI, and Apache plug-in, and HttpClusterServlet

Table 7–1 (Cont.) General Parameters for Web Service 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, informational messages, and error messages.	NSAPI, ISAPI, and Apache plug-in, HttpClusterServlet, and HttpProxyServlet

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

Parameter Name	Default	Description	Applicable to
`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.	NSAPI, ISAPI, and Apache plug-in, HttpClusterServlet, and HttpProxyServlet
`WLDNSRefreshInterval`	0 (Lookup once, during startup)	Only applies to NSAPI and 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.	NSAPI and 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.	NSAPI, 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.	NSAPI, ISAPI, and Apache plug-in, HttpClusterServlet, and HttpProxyServlet

Table 7–1 (Cont.) General Parameters for Web Service 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.	NSAPI and 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, and NSAPI plug-in
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	NSAPI, 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. 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 and NSAPI plug-in, and HttpClusterServlet

Table 7–1 (Cont.) General Parameters for Web Service 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.	NSAPI, ISAPI, and Apache plug-in, HttpClusterServlet, and HttpProxyServlet
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.	NSAPI, ISAPI, and Apache plug-in, HttpClusterServlet, and HttpProxyServlet
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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet

Table 7–1 (Cont.) General Parameters for Web Service 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 Service 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 Plugin 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 plugin will read the data into memory and use it if needed during failover to repeat to the secondary.	ISAPI, Apache and NSAPI plug-in, and HttpClusterServlet
FilterPriorityLevel	2	The values for this parameter are 0 (low), 1 (medium), and 2 (high). The default value is 2. This priority should be put in iisforward.ini file. This property is used to set the priority level for the iisforward.dll filter in IIS. Priority level is used by IIS to decide which filter will be invoked first, in case multiple filters match the incoming request.	ISAPI plug-in

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

Parameter Name	Default	Description	Applicable to
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.	NSAPI, ISAPI, and Apache plug-in
WlForwardPath	null	If WlForwardPath is set to "/" all requests are proxied. To forward any requests starting with a particular string, set WlForwardPath to the string. For example, setting WlForwardPath to /weblogic forwards all requests starting with /weblogic to Weblogic Server. This parameter is required if you are proxying by path. You can set multiple strings by separating the strings with commas. For example: WlForwardPath=/weblogic,/bea.	ISAPI plug-in
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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet
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 and NSAPI plug-in, HttpClusterServlet, and HttpProxyServlet

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

Parameter Name	Default	Description	Applicable to
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 and NSAPI plug-in, and HttpClusterServlet

Table 7–1 (Cont.) General Parameters for Web Service 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.	NSAPI, ISAPI, and Apache plug-in, and HttpClusterServlet
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.	NSAPI, ISAPI, and Apache plug-in, HttpClusterServlet, and HttpProxyServlet
WLLocalIP	none	Defines the IP address 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.	NSAPI, ISAPI, and Apache plug-in

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

Parameter Name	Default	Description	Applicable to
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 similar to the following:

```
@
@
@ <HTML>
@ <HEAD>
@ <TITLE>Weblogic Bridge Message
@ </TITLE>
@ </HEAD>
@  <BODY>
@ <H2>Failure of server APACHE bridge:</H2><P>
@ <hr>Cannot open TEMP post file '/tmp/_wl_proxy/_post_25444_36' for POST of
@ 4564 bytes
```

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
EnforceBasicConstraint	Strong	This parameter closes a security hole which existed with SSL certificate validation where certificate chains with invalid V3 CA certificates would not be properly rejected. This allowed certificate chains with invalid intermediate CA certificates, rooted with a valid CA certificate to be trusted. X509 V3 CA certificates are required to contain the `BasicConstraints` extension, marked as being a CA, and marked as a critical extension. This checking protects against non-CA certificates masquerading as intermediate CA certificates. The levels of enforcement are as follows: ■ OFF This level entirely disables enforcement and is not recommended. Most current commercial CA certificates should work under the default STRONG setting. EnforceBasicConstraints=off EnforceBasicConstraints=false ■ STRONG Default. The BasicConstraints for V3 CA certificates are checked and the certificates are verified to be CA certificates. EnforceBasicConstraints=strong EnforceBasicConstraints=true ■ STRICT This level does the same checking as the STRONG level, but in addition it also strictly enforces IETF RFC 2459 which specifies the BasicConstraints for CA certificates also must be marked as "critical". This is not the default setting because a number of current commercially available CA certificates don't conform to RFC 2459 and don't mark the BasicConstraints as critical. Set this if you want to strict conformance to RFC 2459. EnforceBasicConstraints=strict	NSAPI, ISAPI, and Apache plug-in

Table 7–2 (Cont.) 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, NSAPI, and Apache plug-ins, HttpClusterServlet, and HttpProxyServlet
TrustedCAFile	none	Name of the file that contains the digital certificates for the trusted certificate authorities for the plug-in. This parameter is required if the SecureProxy parameter is set to *ON*.	ISAPI, NSAPI, and Apache plug-ins
RequireSSLHostMatch	true	Determines whether the host name to which the plug-in is connecting must match the Subject Distinguished Name field in the digital certificate of the WebLogic Server to which the proxy plug-in is connecting. When specifying *SecureProxy=ON* and *RequireSSLHostMatch=true* in the plug-in, then the value specified in the *ListenAddress* property should exactly match the hostname value specified in the certificate. When using the *ExternalDNSName* property for WebLogic Server and setting *SecureProxy=ON* and *RequireSSLHostMatch=true* in the plug-in, then the value specified in the *ExternalDNSName* property should exactly match the hostname value specified in the certificate.	ISAPI, NSAPI, and Apache plug-ins
SSLHostMatchOID	22	The ASN.1 Object ID (OID) that identifies which field in the Subject Distinguished Name of the peer digital certificate is to be used to perform the host match comparison. The default for this parameter corresponds to the *CommonName* field of the Subject Distinguished Name. Common OID values are: ■ Sur Name—23 ■ Common Name—22 ■ Email—13 ■ Organizational Unit—30 ■ Organization—29 ■ Locality—26	ISAPI, NSAPI, and Apache plug-ins

Table 7–2 (Cont.) SSL Parameters for Web Server Plug-Ins

Parameter	Default	Description	Applicable to
KeyStore	none	For generic proxy servlets, the key store location in a Web application when using two-way SSL to create a user-defined identity certificate and key.	Applies only to the HttpClusterServlet and to the HttpProxyServlet.
KeyStoreType	none	The key store type when using two-way SSL with a generic proxy servlet. If it is not defined, the default type will be used instead.	Applies only to the HttpClusterServlet and to the HttpProxyServlet.
PrivateKeyAlias	none	The private key alias when using two-way SSL with a generic proxy servlet.	Applies only to the HttpClusterServlet and to the HttpProxyServlet.

KeyStorePasswordProperties	none	A property file in a Web application that defines encrypted passwords to access the key store and private key alias when using two-way SSL with a generic proxy servlet. The file contents looks like this:
		`KeyStorePassword={3DES}i4+50LCKenQO8BBvlsXTrg\=\=`
		`PrivateKeyPassword={3DES}a4TcG4mtVVBRKtZwH3p7yA\=\=`
		You must use the *weblogic.security.Encrypt* command-line utility to encrypt the password. For more information on the Encrypt utility, as well as the CertGen, and der2pem utilities, see Using the Oracle WebLogic Server Java Utilities in the *Oracle Fusion Middleware Command Reference for Oracle WebLogic Server*.
			Applies only to the HttpClusterServlet and to the HttpProxyServlet.