



BEA WebLogic Server™

Using Web Server Plug-Ins with WebLogic Server

BETA

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

BETA

1. Overview of Using Web Server Plug-Ins With WebLogic Server

What Are Plug-Ins? 1-1

Plug-Ins Included with WebLogic Server 1-1

2. Installing and Configuring the Apache HTTP Server Plug-In

Overview of the Apache HTTP Server Plug-In 2-1

Keep-Alive Connections in Apache Version 1.3.x 2-2

Keep-Alive Connections in Apache Version 2.0 2-2

Proxying Requests 2-2

Certifications 2-3

Installing the Apache HTTP Server Plug-In 2-3

Installing the Apache HTTP Server Plug-In as a Dynamic Shared Object 2-3

Installing the Apache HTTP Server Plug-In as a Statically Linked Module 2-8

Configuring the Apache HTTP Server Plug-In 2-10

Editing the httpd.conf File 2-10

Including a weblogic.conf File in the httpd.conf File 2-13

Creating weblogic.conf Files 2-13

Sample weblogic.conf Configuration Files 2-14

Template for the Apache HTTP Server httpd.conf File 2-15

Setting Up Perimeter Authentication 2-16

Using SSL with the Apache Plug-In 2-17

Configuring SSL Between the Apache HTTP Server Plug-In and WebLogic Server 2-17

Issues with SSL-Apache Configuration 2-17

Connection Errors and Clustering Failover 2-19

Possible Causes of Connection Failures 2-19

Tuning to Reduce Connection_Refused Errors 2-19

Failover with a Single, Non-Clustered WebLogic Server 2-21

The Dynamic Server List 2-21

Failover, Cookies, and HTTP Sessions	2-21
--	------

3. Installing and Configuring the Microsoft IIS Plug-In

Overview of the Microsoft Internet Information Server Plug-In	3-1
Connection Pooling and Keep-Alive	3-2
Proxying Requests	3-2
Certifications	3-3
Installing and Configuring the Microsoft Internet Information Server Plug-In	3-3
Proxying Requests from Multiple Virtual Websites to WebLogic Server	3-8
Sample iisproxy.ini File	3-9
Creating ACLs Through IIS	3-10
Setting Up Perimeter Authentication	3-10
Using SSL with the Microsoft Internet Information Server Plug-In	3-11
Proxying Servlets from IIS to WebLogic Server	3-12
Testing the Installation	3-13
Connection Errors and Clustering Failover.	3-14
Possible Causes of Connection Failures	3-14
Failover with a Single, Non-Clustered WebLogic Server	3-14
The Dynamic Server List	3-14
Failover, Cookies, and HTTP Sessions	3-15

4. Installing and Configuring the Netscape Enterprise Server Plug-In

Overview of the Netscape Enterprise Server Plug-In	4-1
Connection Pooling and Keep-Alive	4-2
Proxying Requests	4-2
Installing and Configuring the Netscape Enterprise Server Plug-In	4-3
Guidelines for Modifying the obj.conf File	4-8

Sample obj.conf File (Not Using a WebLogic Cluster)	4-8
Sample obj.conf File (Using a WebLogic Cluster).	4-11
Setting Up Perimeter Authentication.	4-13
Using SSL with the NES Plug-In	4-14
Connection Errors and Clustering Failover.	4-15
Possible Causes of Connection Failures.	4-15
Failover with a Single, Non-Clustered WebLogic Server	4-15
The Dynamic Server List	4-15
Failover, Cookies, and HTTP Sessions	4-15
Failover Behavior When Using Firewalls and Load Directors	4-16

5. Parameters for Web Server Plug-Ins

Entering Parameters in Web Server Plug-In Configuration Files	5-1
General Parameters for Web Server Plug-Ins	5-1
SSL Parameters for Web Server Plug-Ins	5-13

BETA

Overview of Using Web Server Plug-Ins With WebLogic Server

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

- [“What Are Plug-Ins?” on page 1-1](#)
- [“Plug-Ins Included with WebLogic Server” on page 1-1](#)

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 Apache HTTP Server, Netscape Enterprise 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).

Plug-Ins Included with WebLogic Server

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

- Apache HTTP Server
- Microsoft Internet Information Server
- Netscape Enterprise Server

BETA

Installing and Configuring the Apache HTTP Server Plug-In

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

- [“Overview of the Apache HTTP Server Plug-In” on page 2-1](#)
- [“Installing the Apache HTTP Server Plug-In” on page 2-3](#)
- [“Configuring the Apache HTTP Server Plug-In” on page 2-10](#)
- [“Template for the Apache HTTP Server httpd.conf File” on page 2-15](#)
- [“Sample weblogic.conf Configuration Files” on page 2-14](#)
- [“Setting Up Perimeter Authentication” on page 2-16](#)
- [“Using SSL with the Apache Plug-In” on page 2-17](#)
- [“Issues with SSL-Apache Configuration” on page 2-17](#)
- [“Connection Errors and Clustering Failover” on page 2-19](#)

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.

Keep-Alive Connections in Apache Version 1.3.x

Version 1.3.x of the Apache HTTP Server Plug-In creates a socket for each request and closes the socket after reading the response. Because Apache HTTP Server is multiprocessed, connection pooling and keep-alive connections between WebLogic Server and the Apache HTTP Server Plug-In cannot be supported.

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 30 seconds, (or a user-defined amount of time) the connection is closed and returned to the pool. You can disable this feature if desired. For more information, see “KeepAliveEnabled”.

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 2-10](#).

Certifications

The Apache HTTP Server Plug-In is supported on Linux, Solaris, Windows, and HP-UX11 platforms. For information on support for specific versions of Apache, see the [BEA WebLogic Server Certifications Page](#)

Installing the Apache HTTP Server Plug-In

You install the Apache HTTP Server Plug-In as an Apache module in your Apache HTTP Server installation. There are two ways that this module can be compiled and linked to Apache—as a statically linked module (available only for Apache version 1.3.x), or as a Dynamic Shared Object (DSO).

Statically linking a plug-in module requires recompiling Apache, which may be inconvenient.

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

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, Windows, and HP-UX11 platforms. BEA WebLogic supplies versions of shared object files that vary according to platform, whether or not SSL is to be used between the client and Apache, and the SSL encryption strength (regular or 128 bit—128 bit versions are only installed if you install the 128 bit version of WebLogic Server).

[Table 2-1, “Locations of Plug-In Shared Object Files,” on page 2-4](#) shows the directories of your WebLogic Server installation that contain shared object files for various platforms (where WL_HOME is the top-level installation directory for the WebLogic platform).

[Table 2-2, “Apache Plug-In Shared Object File Versions,” on page 2-4](#) identifies the WebLogic Server Apache Plug-In modules for different versions of Apache HTTP Server and different encryption strengths.

Table 2-1 Locations of Plug-In Shared Object Files

Operating System	Shared Object Location
Solaris	WL_HOME/weblogic90/server/solaris/sparc
Linux	WL_HOME/weblogic90/server/linux/i686
Windows (Apache 2.0 only)	WL_HOME\weblogic90\server\32 or WL_HOME\weblogic90\server\64
HPUX11	WL_HOME/weblogic90/server/hpux11/PA_RISC

Warning: 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"
```

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

Table 2-2 Apache Plug-In Shared Object File Versions

Apache Version	Regular Strength Encryption	128-bit Encryption
Standard Apache Version 1.x	mod_wl.so	mod_wl128.so
Apache w/ SSL/EAPI Version 1.x (Stronghold, modssl etc.)	mod_wl_ssl.so	mod_wl128_ssl.so

Table 2-2 Apache Plug-In Shared Object File Versions

Apache Version	Regular Strength Encryption	128-bit Encryption
Apache + Raven Version 1.x Required because Raven applies frontpage patches that makes the plug-in incompatible with the standard shared object	<code>mod_wl_ssl_raven.so</code>	<code>mod_wl128_ssl_raven.so</code>
Standard Apache Version 2.x	<code>mod_wl_20.so</code>	<code>mod_wl128_20.so</code>

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

1. Locate the shared object directory for your platform using [Table 2-1, “Locations of Plug-In Shared Object Files,”](#) on page 2-4.
2. Identify the plug-in shared object file for your version of Apache in [Table 2-2, “Apache Plug-In Shared Object File Versions,”](#) on page 2-4.
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 an Apache Dynamic Shared Object (DSO). DSO support in Apache is based on a module `mod_so.c`, which must be enabled before `mod_wl.so` is loaded. If you installed Apache HTTP Server using the script supplied by Apache, `mod_so.c` is already enabled. Verify that `mod_so.c` is enabled by executing the following command:

```
APACHE_HOME\bin\apache -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 1.3 Shared Object (DSO) Support* at <http://httpd.apache.org/docs/dso.html>.

4. Install the Apache HTTP Server Plug-In module.

- For Apache 1.x, use a command shell to navigate to the directory in your WebLogic Server installation that contains the shared object for your platform and activate the `weblogic_module` by issuing this command (note that you must have Perl installed to run this Perl script):

```
perl APACHE_HOME\bin\apxs -i -a -n weblogic mod_wl.so
```

This command copies the `mod_wl.so` file to the `APACHE_HOME\libexec` directory. It also adds two lines of instructions for `weblogic_module` to the `httpd.conf` file and activates the module. Make sure that the following lines were added to your `APACHE_HOME/conf/httpd.conf` file in your Apache 1.x server installation:

```
LoadModule weblogic_module      libexec/mod_wl.so
AddModule mod_weblogic.c
```

For information about the Apache utility `apxs` (APache eXtenSion) see <http://httpd.apache.org/docs/programs/apxs.html>.

- For Apache 2.x, install the plug-in 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 Server Plug-Ins](#)” on page 5-1. 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 commands:

```
For Apache 1.x, APACHE_HOME\bin\apachectl configtest
```

```
For Apache 2.x, 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 + “/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/`

BETA

Installing the Apache HTTP Server Plug-In as a Statically Linked Module

To install the Apache HTTP Server Plug-In as a statically linked module:

- 1. Unpack the Apache Plug-In distribution using the following command:
`tar -xvf apache_1.3.x.tar`
- 2. Within the unpacked distribution switch to the `src/modules` directory.
- 3. Create a directory called `weblogic`.
- 4. Locate the linked library file for your platform.

Each library file is distributed as a separate version, depending on the platform and the encryption strength for SSL (regular or 128-bit—128-bit versions are only installed if you install the 128-bit version of WebLogic Server). The library files are located in the following directories of your WebLogic Server installation:

Table 2-3 Locations of Plug-In Files

Operating System	Shared Object Location
Solaris	WL_HOME/Server/lib/solaris
Linux	WL_HOME/Server/lib/linux
HPUX11	WL_HOME/Server/lib/hpux11

Warning: If you are running Apache 2.0.x server on HP-UX11, set the environment variables specified 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"
```

Choose the appropriate shared object from the following table.

Table 2-4 Statically Linked Modules for Different Levels of Encryption

Apache Version	Regular Strength Encryption	128-bit Encryption
Standard Apache Version 1.3.x	libweblogic.a	libweblogic128.a

If you are using the Gnu C Compiler (gcc), gcc 2.95.x is the recommended version.

5. Copy `Makefile.libdir`, `Makefile.tmpl` from the `lib` directory of your WebLogic Server installation to `src\modules\weblogic`.
6. Copy `libweblogic.a` (use `libweblogic128.a` instead, if you are using 128 bit security.) from the same directory containing the linked library file to `src\modules\weblogic`.
7. If you are using regular strength encryption, execute the following command from the Apache 1.3 home directory:


```
configure --activate-module=src\modules\weblogic\libweblogic.a
```
8. If you are using 128 bit encryption, execute the following command (on a single line) from the Apache 1.3 home directory:


```
configure--activate-module=src\modules\weblogic\libweblogic128.a
```
9. Execute the following command:


```
make
```
10. Execute the following command:


```
make install
```
11. 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 Server Plug-Ins” on page 5-1](#). 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*.

12. Verify the syntax of the `APACHE_HOME\conf\httpd.conf` file with the following commands:

For Apache 1.x, `APACHE_HOME\bin\apachectl configtest`

For Apache 2.x, `APACHE_HOME\bin\Apache -t`

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

Syntax OK

13. Restart Weblogic Server.
14. Start (or restart if you have changed the configuration) Apache HTTP Server.
15. Test the plug-in by opening a browser and setting the URL to the Apache Server + `/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/`

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.

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 [“Template for the Apache HTTP Server httpd.conf File” on page 2-15](#).

2. Ensure that the WebLogic Server modules are included.

- For Apache 1.x, verify that the following two lines were added to the `httpd.conf` file when you ran the `apxs` utility:

```
LoadModule weblogic_module    libexec/mod_wl.so
AddModule mod_weblogic.c
```

- For Apache 2.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 wls1.com:7001,wls2.com:7001,wls3.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 Server Plug-Ins” on page 5-1](#)).

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 /HTTPCInt>
  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 Server Plug-Ins” on page 5-1](#). 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.

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.

Creating `weblogic.conf` Files

Be aware of the following when constructing a `weblogic.conf` file.

- If you are using SSL between the Apache HTTP Server Plug-In and WebLogic Server, you cannot define parameters in a file accessed, as the `weblogic.conf` file is, via the Apache Include directive.
- 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 define the `CookieName` parameter (see http://e-docs.bea.com/wls/docs90/webapp/weblogic_xml.html#session-descriptor), you must define it in an `IfModule` block.
- 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](#)).

- BEA recommends that you use the `MatchExpression` statement instead of the `<files>` block.

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 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 wls1.com:7001,wls2.com:7001,wls3.com:7001
    ErrorPage http://myerrorpage.mydomain.com
    MatchExpression *.jsp
</IfModule>

#####
```

Example Using Multiple WebLogic Clusters

In this example, 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.

```
# 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 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>
```

Template for the Apache HTTP Server httpd.conf File

This section contains a sample httpd.conf file for Apache 1.3. 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.

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

LoadModule weblogic_module    libexec/mod_wl.so

AddModule mod_weblogic.c

<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 wls1.com:7001,wls2.com:7001,wls3.com:7001
    ErrorPage http://myerrorpage.mydomain.com
</IfModule>
```

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 *Developing Security Providers for WebLogic Server*.
2. Configure the custom Identity Assertion Provider to support the Cert token type and make Cert the active token type. See [How to Create New Token Types](#) in *Developing Security Providers for 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). For more information about the `clientCertProxy` attribute, see [context-param](#) in *Developing Web Applications, Servlets and JSPs for 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 *Programming WebLogic Security*.
See [Identity Assertion Providers](#) in *Developing Security Providers for WebLogic Server*.

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.

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 the SSL Protocol](#) at <http://e-docs.bea.com/wls/docs90/secmanage/ssl.html>.
2. Configure the WebLogic Server SSL listen port. For more information, see [Configuring the SSL Protocol](#) at <http://e-docs.bea.com/wls/docs90/secmanage/ssl.html>.
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 5-13.

Issues with SSL-Apache Configuration

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

- The PathTrim parameter (see [“General Parameters for Web Server Plug-Ins”](#) on page 5-1) 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 Include directive does not work with Apache SSL. You must configure all parameters directly in the httpd.conf file. Do not use the following configuration when using SSL:

```
<IfModule mod_weblogic.c>
  MatchExpression *.jsp
  Include weblogic.conf
</IfModule>
```

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

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 2-1 “Connection Failover” on page 2-21](#) demonstrates how the plug-in handles failover.

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

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. See [Tuning Connection Backlog Buffering](#)
- On Apache 2.x, set the `KeepAlive` directive in the `httpd.conf` file to `On`. For example:


```
# KeepAlive: Whether or not to allow persistent connections (more than
```

Installing and Configuring the Apache HTTP Server Plug-In

```
# 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/>.

Apache 1.3.x does not support this feature.

- 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):

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

- Increase the open file descriptor limit on your 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`

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 until `ConnectTimeoutSecs` is exceeded.

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.

Failover, Cookies, and HTTP Sessions

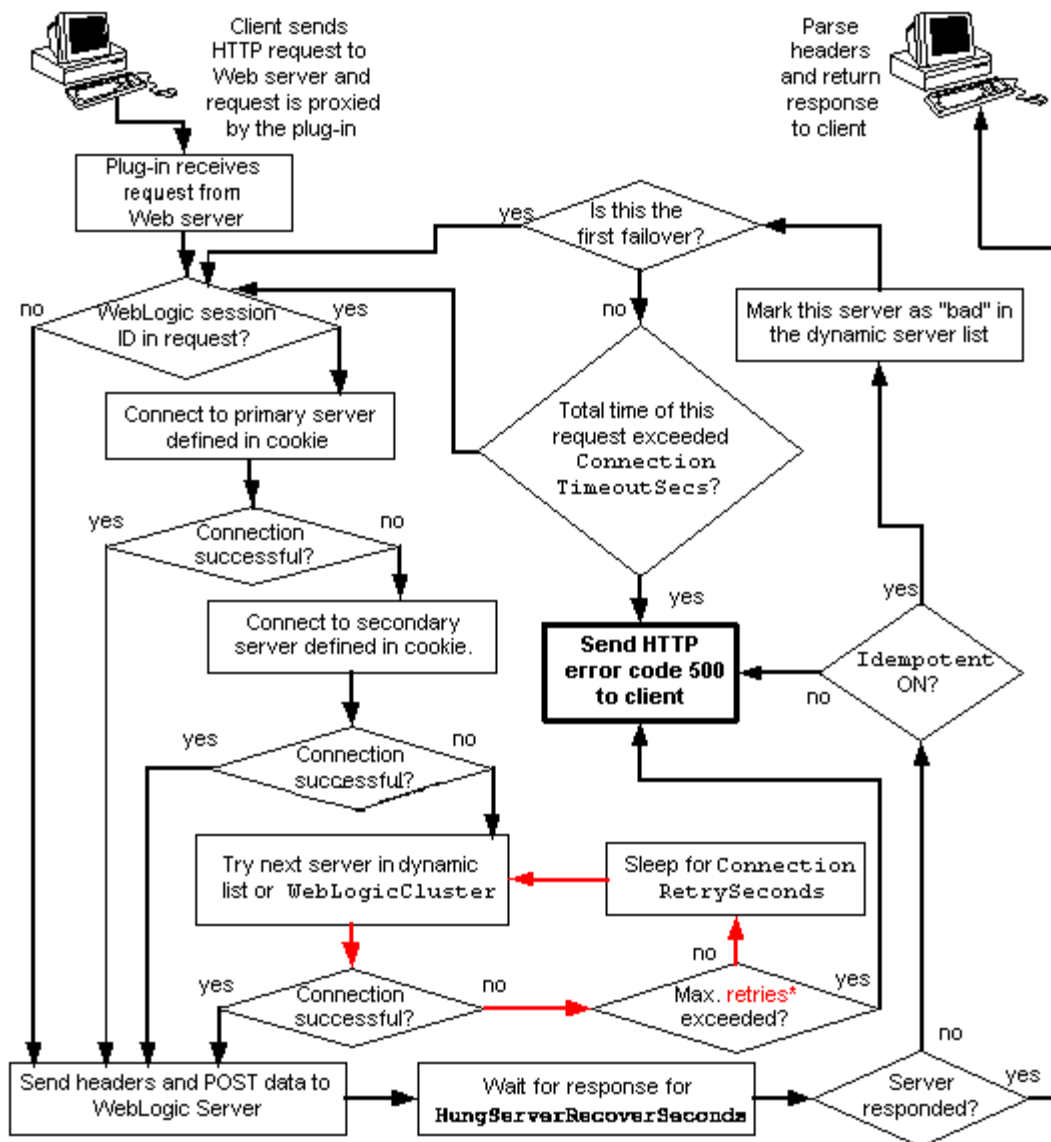
When a request contains session information stored in a cookie or in the POST data, or encoded in a URL, the session ID contains a reference to the specific server instance in which the session was originally established (called the *primary* server) and a reference to an additional server where the original session is replicated (called the *secondary* server). A request containing a cookie attempts to connect to the primary server. If that attempt fails, the request is routed to the secondary server. If both the primary and secondary servers fail, the session is lost and the plug-in attempts to make a fresh connection to another server in the dynamic cluster list. See [Figure 2-1 “Connection Failover” on page 2-21](#).

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 2-1 Connection Failover

The Maximum number of retries allowed in the red loop is equal to $\text{ConnectTimeoutSecs} \div \text{ConnectRetrySecs}$.

Installing and Configuring the Apache HTTP Server Plug-In



Installing and Configuring the Microsoft IIS Plug-In

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

- [“Overview of the Microsoft Internet Information Server Plug-In” on page 3-1](#)
- [“Certifications” on page 3-3](#)
- [“Installing and Configuring the Microsoft Internet Information Server Plug-In” on page 3-3](#)
- [“Proxying Requests from Multiple Virtual Websites to WebLogic Server” on page 3-8](#)
- [“Sample iisproxy.ini File” on page 3-9](#)
- [“Creating ACLs Through IIS” on page 3-10](#)
- [“Setting Up Perimeter Authentication” on page 3-10](#)
- [“Using SSL with the Microsoft Internet Information Server Plug-In” on page 3-11](#)
- [“Proxying Servlets from IIS to WebLogic Server” on page 3-12](#)
- [“Testing the Installation” on page 3-13](#)
- [“Connection Errors and Clustering Failover” on page 3-14](#)

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.

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” on page 5-10](#).

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 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 [“Installing and Configuring the Microsoft Internet Information Server Plug-In” on page 3-3](#).

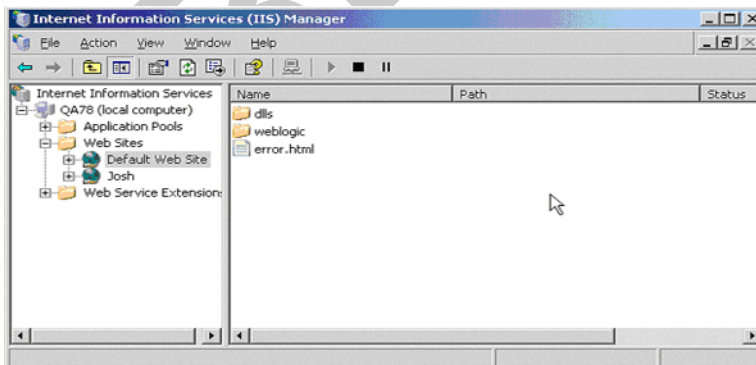
Certifications

For the latest information on operating system and IIS version compatibility with the Microsoft Internet Information Server Plug-In, see the [platform support page](http://e-docs.bea.com/wls/certifications/certifications/index) at <http://e-docs.bea.com/wls/certifications/certifications/index>.

Installing and Configuring the Microsoft Internet Information Server Plug-In

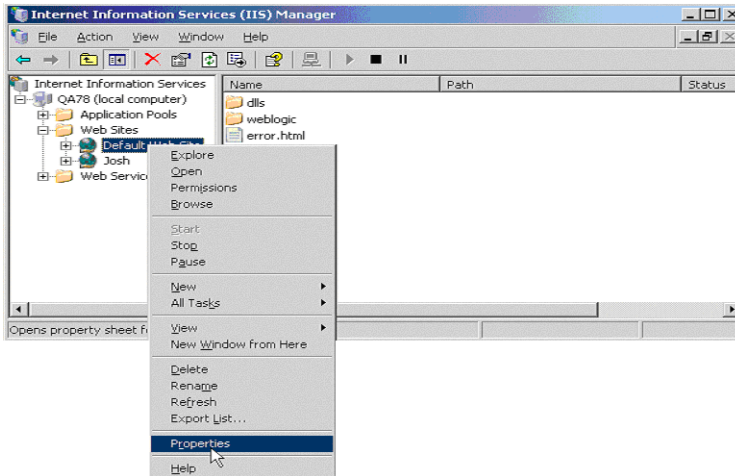
To install the Microsoft Internet Information Server Plug-In:

1. Copy the `iisproxy.dll` file from the `WL_HOME/server/bin` 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.
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”).

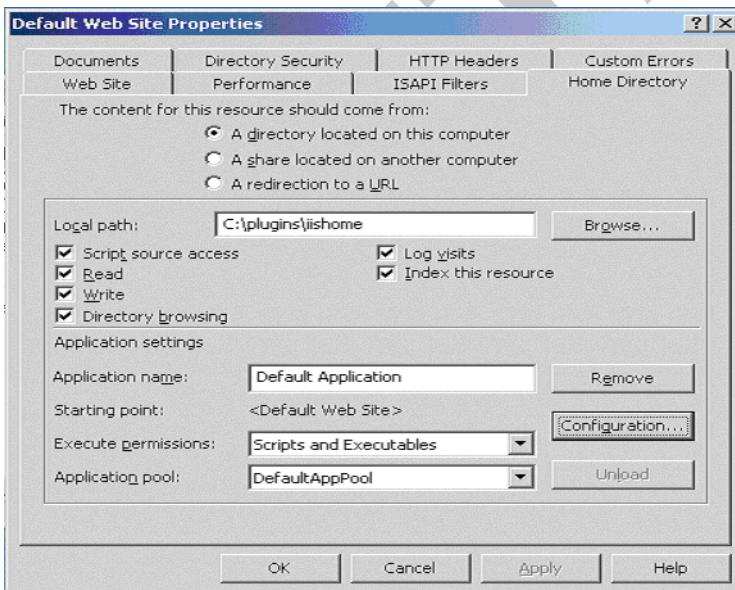


- c. Click the “Play” arrow in the toolbar to start.
- d. Open the properties for the selected website by right-clicking the website selection in the left panel and selecting Properties.

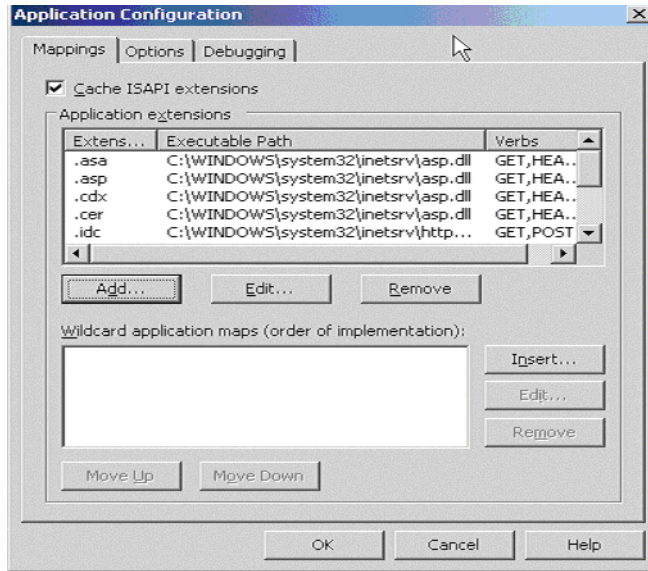
Installing and Configuring the Microsoft IIS Plug-In



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



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



- 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 3-8.](#)

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`).
 - c. Register `.wlforward` as a special file type to be handled by `iisproxy.dll`.
 - 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 Server Plug-Ins” on page 5-1](#).

Use the example `iisproxy.ini` file in this section ([“Sample iisproxy.ini File” on page 3-9](#)) 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).

BEA 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=*/HTTPCInt*
```
 - 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 Server Plug-Ins”](#) on page 5-1.
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 3-12.
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.

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 step 1.
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 websites, as described in [step 4. in “Proxying Requests”](#). Copy this `iisproxy.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 WLS instances defined in the `iisproxy.ini` of the first website.

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.

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”.

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 *Developing Security Providers for 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 *Developing Security Providers for 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 *Developing Web Applications for 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 *Programming WebLogic Security*.

See [Identity Assertion Providers](#) in *Developing Security Providers for WebLogic Server* for more information about Identity Assertion Providers.

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 the SSL Protocol](http://e-docs.bea.com/wls/docs90/secmanage/ssl.html) at <http://e-docs.bea.com/wls/docs90/secmanage/ssl.html>.
2. Configure the WebLogic Server SSL listen port. For more information, see [Configuring the SSL Protocol](http://e-docs.bea.com/wls/docs90/secmanage/ssl.html) at <http://e-docs.bea.com/wls/docs90/secmanage/ssl.html>.
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 5-13.

For example:

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

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 3-3.
2. Register your servlet in the appropriate Web Application. For more information on registering servlets, see [Configuring Servlets](http://e-docs.bea.com/wls/docs90/webapp/components.html#configuring-servlets) at <http://e-docs.bea.com/wls/docs90/webapp/components.html#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.

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 + 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.

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 3-1 “Connection Failover” on page 3-15 demonstrates how the plug-in handles failover.

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.

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 until `ConnectTimeoutSecs` is exceeded.

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.

Failover, Cookies, and HTTP Sessions

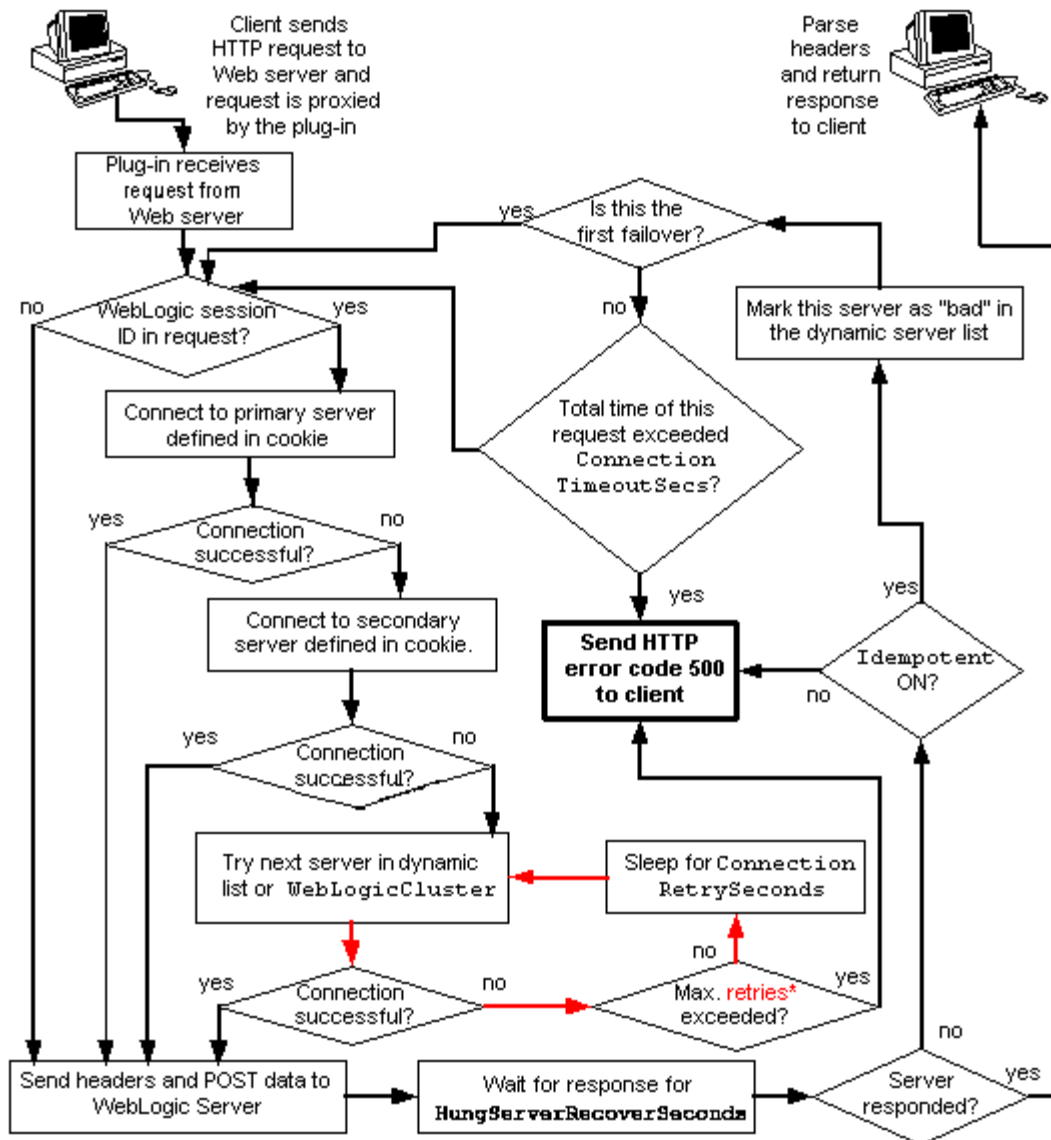
When a request contains a session information stored in a cookie, in the POST data, or by URL encoding, the session ID contains a reference to the specific server in which the session was originally established (called the *primary* server) and a reference to an additional server where the original session is replicated (called the *secondary* server). A request containing a cookie attempts to connect to the primary server. If that attempt fails, the request is routed to the secondary server. If both the primary and secondary servers fail, the session is lost and the plug-in attempts to make a fresh connection to another server in the dynamic cluster list. For more information see [Figure 3-1 “Connection Failover” on page 3-15](#).

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.

Figure 3-1 Connection Failover

*The Maximum number of retries allowed in the **red** loop is equal to $\text{ConnectTimeoutSecs} \div \text{ConnectRetrySecs}$.

Installing and Configuring the Microsoft IIS Plug-In



Installing and Configuring the Netscape Enterprise Server Plug-In

The following sections describe how to install and configure the Netscape Enterprise Server (NES) proxy plug-in:

- [“Overview of the Netscape Enterprise Server Plug-In” on page 4-1](#)
- [“Installing and Configuring the Netscape Enterprise Server Plug-In” on page 4-3](#)
- [“Setting Up Perimeter Authentication” on page 4-13](#)
- [“Using SSL with the NES Plug-In” on page 4-14](#)
- [“Connection Errors and Clustering Failover” on page 4-15](#)
- [“Failover Behavior When Using Firewalls and Load Directors” on page 4-16](#)
- [“Sample obj.conf File \(Not Using a WebLogic Cluster\)” on page 4-8](#)
- [“Sample obj.conf File \(Using a WebLogic Cluster\)” on page 4-11](#)

Overview of the Netscape Enterprise Server Plug-In

The Netscape Enterprise Server Plug-In enables requests to be proxied from Netscape Enterprise Server (NES, also called iPlanet) to WebLogic Server. The plug-in enhances an NES installation by allowing WebLogic Server to handle those requests that require the dynamic functionality of WebLogic Server.

The Netscape Enterprise Server Plug-In is designed for an environment where Netscape Enterprise 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 Netscape Enterprise 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 Netscape Enterprise Server Plug-In, providing access to all WebLogic Server services (not just dynamic pages).

The Netscape Enterprise Server Plug-In operates as an [NES module](#) (see <http://home.netscape.com/servers/index.html>) within a Netscape Enterprise Server. The NES module is loaded by NES at startup, and then certain HTTP requests are delegated to it. NES is similar to an HTTP (Java) servlet, except that an NES module is written in code native to the platform.

For more information on supported versions of Netscape Enterprise Server and iPlanet servers, see the [BEA WebLogic Server Certifications Page](#).

Connection Pooling and Keep-Alive

The WebLogic Server Netscape Enterprise Server Plug-In provides efficient performance by using a re-usable pool of connections from the plug-in to WebLogic Server. The NES 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” on page 5-10](#).

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 Netscape Enterprise Server Plug-In” on page 4-3](#).

Installing and Configuring the Netscape Enterprise Server Plug-In

To install and configure the Netscape Enterprise Server Plug-In:

1. Copy the library.

The WebLogic NES 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 respectively located in the *WL_HOME*/server/lib or *WL_HOME*/server/bin directories 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.

Choose the appropriate library file for your environment from the [Certifications table](http://e-docs.bea.com/wls/certifications/certifications/index.html) at <http://e-docs.bea.com/wls/certifications/certifications/index.html> and copy that file into the file system where NES is located.

2. Read “[Guidelines for Modifying the obj.conf File](#)” on page 4-8, then modify the NES 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 NES instance is in the following location:

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

Where *NETSCAPE_HOME* is the root directory of the NES 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 NES to load the native library (the .so or .dll file) as an NES 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 Netscape Enterprise Server Plug-In](#)” on page 4-3. The function “load-modules” tags the shared library for loading

when NES starts up. The values “wl_proxy” and “wl_init” identify the functions that the Netscape Enterprise Server Plug-In executes.

To use iPlanet 6.0, add the following lines to the beginning of the `magnus.conf` file. These lines instruct NES to load the native library (the `.so` or `.dll` file) as an NES module:

```
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 Netscape Enterprise Server Plug-In” on page 4-3. The function “load-modules” tags the shared library for loading when NES starts up. The values “wl_proxy” and “wl_init” identify the functions that the Netscape Enterprise 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 Netscape Enterprise 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 Netscape Enterprise 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 Server Plug-Ins” on page 5-1](#). 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 Netscape Enterprise Server Plug-In. For a complete list of plug-in parameters, see [“General Parameters for Web Server Plug-Ins” on page 5-1](#).

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 NES 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 Preferences→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 Netscape Enterprise Server Plug-In, the default `Object` definition will be similar to the following example, with the additions shown in **bold**:

- ```
<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/imapmap\ fn=imapmap
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.
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="*/HTTPCInt">
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 Netscape Enterprise Server Plug-In

- a. Start WebLogic Server.
- b. Start Netscape Enterprise Server. If NES 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 Netscape Enterprise Server Plug-In, open a browser and set the URL to the Netscape Enterprise Server + `/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, read [Developing Web Applications, Servlets and JSAPs for WebLogic Server](http://e-docs.bea.com/wls/docs90/webapp/overview.html) at <http://e-docs.bea.com/wls/docs90/webapp/overview.html>.

## Guidelines for Modifying the obj.conf File

To use the Netscape Enterprise 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 Netscape Enterprise Server Plug-In](#)” on page 4-3.

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`.)

## 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 Netscape Enterprise Server Plug-In 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
#
```

## Installing and Configuring the Netscape Enterprise Server Plug-In

```
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="*/HTTPCln*">
Service fn=wl_proxy WebLogicHost=192.192.1.4 WebLogicPort=7001
</Object>

#
----- END SAMPLE OBJ.CONF CONFIGURATION -----
```

## 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" func="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.
```

## Installing and Configuring the Netscape Enterprise Server Plug-In

```
<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/imapemap\ fn=imapemap
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="*/HTTPCInt*">
Service fn=wl_proxy WebLogicCluster="myweblogic.com:7001,\
 yourweblogic.com:7001,theirweblogic.com:7001"
</Object>

#
----- END SAMPLE OBJ.CONF CONFIGURATION -----

```

## Setting Up Perimeter Authentication

Use perimeter authentication to secure your WebLogic Server applications that are accessed via the Netscape Enterprise 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 Netscape Enterprise 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 *Developing Security Providers for 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 *Developing Security Providers for 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 *Developing Web Applications, Servlets and JSPs for 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 Netscape Enterprise Server Plug-In is running. See [Using Network Connection Filters](#) in *Programming WebLogic Security*.

See [Identity Assertion Providers](#) in *Developing Security Providers for WebLogic Server* for more information about Identity Assertion Providers.

## Using SSL with the NES Plug-In

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

The Netscape Enterprise 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 Netscape Enterprise Server Plug-In and WebLogic Server.

To use the SSL protocol between Netscape Enterprise Server Plug-In and WebLogic Server:

1. Configure WebLogic Server for SSL. For more information, see [Configuring the SSL Protocol](http://e-docs.bea.com/wls/docs90/secmanage/ssl.html) at <http://e-docs.bea.com/wls/docs90/secmanage/ssl.html>.
2. Configure the WebLogic Server SSL listen port. For more information, see [Configuring the SSL Protocol](http://e-docs.bea.com/wls/docs90/secmanage/ssl.html) at <http://e-docs.bea.com/wls/docs90/secmanage/ssl.html>.
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 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 5-13.

## Connection Errors and Clustering Failover

When the Netscape Enterprise 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 4-1 “Connection Failover” on page 4-16 demonstrates how the plug-in handles failover.

## 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.

## 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.

## 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.

## Failover, Cookies, and HTTP Sessions

When a request contains session information stored in a cookie, in the POST data, or by URL encoding, the session ID contains a reference to the specific server in which the session was originally established (called the *primary* server) and a reference to an additional server where

the original session is replicated (called the *secondary* server). A request containing a cookie attempts to connect to the primary server. If that attempt fails, the request is routed to the secondary server. If both the primary and secondary servers fail, the session is lost and the plug-in attempts to make a fresh connection to another server in the dynamic cluster list. For more information, see [Figure 4-1 “Connection Failover” on page 4-16](#).

**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.

## Failover Behavior When Using Firewalls and Load Directors

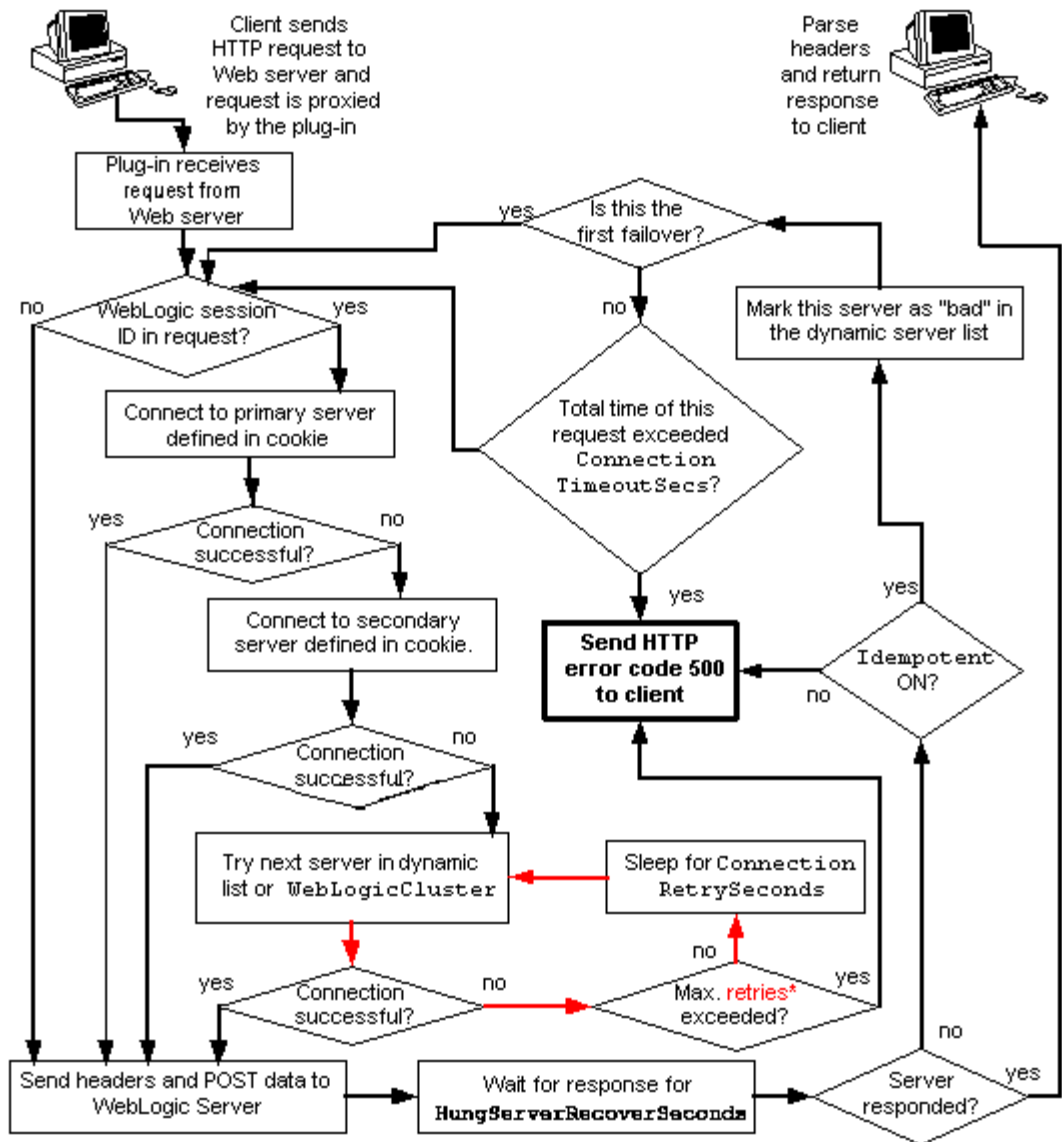
In most configurations, the Netscape Enterprise Server Plug-In sends a request to the primary instance of a cluster. When that instance is unavailable, the request fails over to the secondary instance. However, in some configurations that use combinations of firewalls and load-directors, any one of the 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.

### Figure 4-1 Connection Failover

\*The Maximum number of retries allowed in the `red` loop is equal to  $\text{ConnectTimeoutSecs} \div \text{ConnectRetrySecs}$ .





BETA

# Parameters for Web Server Plug-Ins

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

- [Entering Parameters in Web Server Plug-In Configuration Files](#)
- [General Parameters for Web Server Plug-Ins](#)
- [SSL Parameters for Web Server Plug-Ins](#)

## 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:

- [“Installing and Configuring the Apache HTTP Server Plug-In” on page 2-1](#)
- [“Installing and Configuring the Microsoft IIS Plug-In” on page 3-1](#)
- [“Installing and Configuring the Netscape Enterprise Server Plug-In” on page 4-1](#)

## General Parameters for Web Server Plug-Ins

**Note:** Parameters are case sensitive.

Table 5-1

Parameter	Default	Description
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.
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 <a href="#">Configuring the SSL Protocol</a> at <a href="http://e-docs.bea.com/wls/docs90/secmanage/ssl.html">http://e-docs.bea.com/wls/docs90/secmanage/ssl.html</a> ) and set the SecureProxy parameter to ON ).  If you are using a WebLogic Cluster, use the WebLogicCluster parameter instead of WebLogicPort.

Table 5-1

Parameter	Default	Description
WebLogicCluster (Required when proxying to a cluster of WebLogic Servers.)	none	<p>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.</p> <p>The method of specifying the parameter, and the required format vary by plug-in. See the examples in:</p> <ul style="list-style-type: none"> <li>"Installing and Configuring the Netscape Enterprise Server Plug-In (NSAPI)"</li> <li>"Installing and Configuring the Microsoft Internet Information Server (ISAPI) Plug-In"</li> <li>"Installing and Configuring the Apache HTTP Server Plug-In"</li> </ul> <p>If you are using SSL between the plug-in and WebLogic Server, set the port number to the SSL listen port (see <a href="#">Configuring the SSL Protocol</a> at <a href="http://e-docs.bea.com/wls/docs90/secmanage/ssl.html">http://e-docs.bea.com/wls/docs90/secmanage/ssl.html</a>) and set the <a href="#">SecureProxy</a> parameter to ON.</p> <p>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.</p> <p>You can disable the use of the dynamic cluster list by setting the <a href="#">DynamicServerList</a> parameter to OFF</p> <p>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.</p>

Table 5-1

Parameter	Default	Description
PathTrim	null	<p>String trimmed by the plug-in from the beginning of the original URL, before the request is forwarded to WebLogic Server. For example, if the URL</p> <pre>http://myWeb.server.com/weblogic/foo</pre> <p>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:</p> <pre>http://myWeb.server.com:7001/foo</pre> <p>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.</p>
PathPrepend	null	<p>String that the plug-in prepends to the beginning of the original URL, after PathTrim is trimmed and before the request is forwarded to WebLogic Server.</p>
ConnectTimeoutSecs	10	<p>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.</p> <p>You can customize the error response by using the ErrorPage parameter.</p>
ConnectRetrySecs	2	<p>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.</p> <p>To specify no retries, set ConnectRetrySecs equal to ConnectTimeoutSecs. However, the plug-in attempts to connect at least twice.</p> <p>You can customize the error response by using the ErrorPage parameter.</p>

Table 5-1

Parameter	Default	Description
Debug	OFF	<p>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 <a href="#">WLLogFile</a> parameter to a different directory and file. Ensure that the tmp or TEMP directory has write permission assigned to the user who is logged in to the server. 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”):</p> <p>ON The plug-in logs informational and error messages.</p> <p>OFF No debugging information is logged.</p> <p>HFC The plug-in logs headers from the client, informational, and error messages.</p> <p>HTW The plug-in logs headers sent to WebLogic Server, and informational and error messages.</p> <p>HFW The plug-in logs headers sent from WebLogic Server, and informational and error messages.</p> <p>HTC The plug-in logs headers sent to the client, informational messages, and error messages.</p> <p>ERR Prints only the Error messages in the plug-in.</p> <p>ALL The plug-in logs headers sent to and from the client, headers sent to and from WebLogic Server, information messages, and error messages.</p>

Table 5-1

Parameter	Default	Description
WLLogFile	See the <a href="#">Debug</a> parameter	Specifies path and file name for the log file that is generated when the <a href="#">Debug</a> parameter is set to ON. You must create this directory before setting this parameter.
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 the server list.
WLTempDir	See the <a href="#">Debug</a> parameter	<p>Specifies the directory where a <code>wlproxy.log</code> will be created. If the location fails, the Plug-In resorts to creating the log file under <code>C:/temp</code> in Windows and <code>/tmp</code> in all Unix platforms.</p> <p>Also specifies the location of the <code>_wl_proxy</code> directory for post data files.</p> <p>When both <code>WLTempDir</code> and <code>WLLogFile</code> are set, <code>WLLogFile</code> will override as to the location of <code>wlproxy.log</code>. <code>WLTempDir</code> will still determine the location of <code>_wl_proxy</code> directory.</p>
DebugConfigInfo	OFF	<p>Enables the special query parameter “<code>__WebLogicBridgeConfig</code>”. Use it to get details about configuration parameters from the plug-in.</p> <p>For example, if you enable “<code>__WebLogicBridgeConfig</code>” by setting <code>DebugConfigInfo</code> and then send a request that includes the query string <code>?__WebLogicBridgeConfig</code>, 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.</p> <p>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.</p>



Table 5-1

Parameter	Default	Description
StatPath  (Not available for the Microsoft Internet Information Server Plug-In)	false	<p>If set to <code>true</code>, 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.</p> <p>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.</p> <p>You can customize the error response by using the <a href="#">ErrorPage</a> parameter.</p>
ErrorPage	none	You can create your own error page that is displayed when your Web server is unable to forward requests to WebLogic Server.
WLSocketTimeoutSecs	2 (must be greater than 0)	Set the timeout for the socket while connecting, in seconds.
WLIOTimeoutSecs (new name for HungServerRecoverSecs)	300	<p>Defines the amount of time the plug-in waits for a response to a request from WebLogic Server. The plug-in waits for HungServerRecoverSecs 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.</p> <p>Minimum value: 10 Maximum value: Unlimited</p>
Idempotent	ON	<p>When set to ON and if the servers do not respond within <a href="#">WLIOTimeoutSecs (new name for HungServerRecoverSecs)</a>, the plug-ins fail over.</p> <p>If set to “OFF” the plug-ins do not fail over. If you are using the Netscape Enterprise Server Plug-In, or Apache HTTP Server you can set this parameter differently for different URLs or MIME types.</p>

Table 5-1

Parameter	Default	Description
CookieName	JSESSIONID	If you change the name of the WebLogic Server session cookie in the WebLogic Server Web application, you need to change the CookieName 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 <code>&lt;session-descriptor&gt;</code> (see <a href="http://e-docs.bea.com/wls/docs90/webapp/weblogic_xml.html#session-descriptor">http://e-docs.bea.com/wls/docs90/webapp/weblogic_xml.html#session-descriptor</a> ) element.
DefaultFileName	none	<p>If the URI is “/” then the plug-in performs the following steps:</p> <ol style="list-style-type: none"> <li>1. Trims the path specified with the <a href="#">PathTrim</a> parameter.</li> <li>2. Appends the value of DefaultFileName.</li> <li>3. Prepends the value specified with <a href="#">PathPrepend</a>.</li> </ol> <p>This procedure prevents redirects from WebLogic Server.</p> <p>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 “<a href="#">Configuring Welcome Pages</a>” at <a href="http://e-docs.bea.com/wls/docs90/webapp/configureservlet">http://e-docs.bea.com/wls/docs90/webapp/configureservlet</a>.</p> <p>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.</p>
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.

Table 5-1

Parameter	Default	Description
MatchExpression (Apache HTTP Server only)	none	<p>When proxying by MIME type, set the filename pattern inside of an <code>IfModule</code> block using the <code>MatchExpression</code> parameter.</p> <p>Example when proxying by MIME type:</p> <pre>&lt;IfModule mod_weblogic.c&gt;   MatchExpression *.jsp   WebLogicHost=myHost paramName=value &lt;/IfModule&gt;</pre> <p>Example when proxying by path:</p> <pre>&lt;IfModule mod_weblogic.c&gt;   MatchExpression /weblogic   WebLogicHost=myHost paramName=value &lt;/IfModule&gt;</pre>
FileCaching	ON	<p>When set to ON, and the size of the POST data in a request is greater than 2048 bytes, the POST data is stored on disk in a temporary file and forwarded to WebLogic Server in chunks of 8192 bytes. Setting <code>FileCaching</code> to ON, however, can cause a problem with the progress bar displayed by a browser that indicates the progress of a download. The browser shows that the download has completed even though the file is still being transferred.</p> <p>When set to OFF and size of the POST data in a request is greater than 2048 bytes, the POST data is stored in memory and sent to WebLogic Server in chunks of 8192 bytes. Setting to OFF causes problems if the server goes down while processing the request because the plug-in is not able to fail over.</p>
FilterPriorityLevel (Microsoft Internet Information Server only)	2	<p>The values for this parameter are 0 (low), 1 (medium), and 2 (high). The default value is 2. This priority should be put in <code>iisforward.ini</code> file. This property is used to set the priority level for the <code>iisforward.dll</code> 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.</p>

Table 5-1

Parameter	Default	Description
WLExcludePathOrMimeType	none	<p>This parameter allows you make exclude certain requests from proxying.</p> <p>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.</p>
WlForwardPath (Microsoft Internet Information Server only)	null	<p>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.</p> <p>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.</p>
KeepAliveSecs (Does not apply to Apache HTTP Server version 1.3.x)	20	<p>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.</p> <p>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.</p>
KeepAliveEnabled (Does not apply to Apache HTTP Server version 1.3.x)	true (Netscape and Microsoft IIS plug-ins) ON (Apache plug-in)	<p>Enables pooling of connections between the plug-in and WebLogic Server.</p> <p>Valid values for the Netscape and Microsoft IIS plug-ins are true and false.</p> <p>Valid values for the Apache plug-in are ON and OFF.</p>

Table 5-1

Parameter	Default	Description
QueryFromRequest (Apache HTTP Server only)	OFF	<p>When set to ON, specifies that the Apache plug-in use (request_rec *)r-&gt;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:</p> <ul style="list-style-type: none"> <li>• When a Netscape version 4.x browser makes requests that contain spaces in the query string</li> <li>• If you are using Raven Apache 1.5.2 on HP</li> </ul> <p>When set to OFF, the Apache plug-in uses (request_rec *)r-&gt;args to pass the query string to WebLogic Server.</p>
MaxSkipTime	10	<p>If a WebLogic Server listed in either the <a href="#">WebLogicCluster</a> 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.</p> <p>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).</p>
DynamicServerList	ON	<p>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 <a href="#">WebLogicCluster</a> parameter. Normally this parameter should remain set to ON.</p> <p>There are some implications for setting this parameter to OFF:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>

Table 5-1

Parameter	Default	Description
WLProxySSL	OFF	<p>Set this parameter to ON to maintain SSL communication between the plug-in and WebLogic Server when the following conditions exist:</p> <ul style="list-style-type: none"><li>• An HTTP client request specifies the HTTPS protocol</li><li>• The request is passed through one or more proxy servers (including the WebLogic Server proxy plug-ins)</li><li>• The connection between the plug-in and WebLogic Server uses the HTTP protocol</li></ul> <p>When WLProxySSL is set to ON, the location header returned to the client from WebLogic Server specifies the HTTPS protocol.</p>
WLLocalIP	none	<p>Defines the IP address to bind to when the plug-in connects to a WebLogic Server instance running on a multihomed machine.</p> <p>If WLLocalIP is not set, a random IP address on the multi-homed machine is used.</p>

## 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.

BETA

**Note:** Parameters are case sensitive.

BETA



Table 5-2

Parameter	Default	Description
EnforceBasicConstraints	Strong	<p>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.</p> <p>The levels of enforcement are as follows:</p> <p><b>OFF</b></p> <p>This level entirely disables enforcement and is not recommended. Most current commercial CA certificates should work under the default STRONG setting.</p> <p><code>EnforceBasicConstraints=off</code>  <code>EnforceBasicConstraints=false</code></p> <p><b>STRONG</b></p> <p>Default. The BasicConstraints for V3 CA certificates are checked and the certificates are verified to be CA certificates.</p> <p><code>EnforceBasicConstraints=strong</code>  <code>EnforceBasicConstraints=true</code></p> <p><b>STRICT</b></p> <p>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.</p> <p><code>EnforceBasicConstraints=strict</code></p>

Table 5-2

Parameter	Default	Description
SecureProxy	OFF	<p>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.</p> <p>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.</p>
TrustedCAFile	none	<p>Name of the file that contains the digital certificates for the trusted certificate authorities for the plug-in. This parameter is required if the <a href="#">SecureProxy</a> parameter is set to ON.</p> <p>The filename must include the full directory path of the file.</p>
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.
SSLHostMatchOID	22	<p>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 <code>CommonName</code> field of the Subject Distinguished Name. Common OID values are:</p> <ul style="list-style-type: none"> <li>• Sur Name—23</li> <li>• Common Name—22</li> <li>• Email—13</li> <li>• Organizational Unit—30</li> <li>• Organization—29</li> <li>• Locality—26</li> </ul>

## A

- Apache plug-in 1
  - and clusters 14
  - and SSL 17
  - httpd.conf file 10
  - installing 3
  - parameters 6, 9, 12
  - sample httpd.conf file 15

## C

### Configuration

- Apache plug-in 6, 9, 12
- Microsoft-IIS (proxy) plug-in 6

ConnectionRetrySecs 4

ConnectionTimeoutSecs 4

## D

Debug 5

DebugConfigInfo 6

DefaultFileName 8

DynamicServerList 11

## E

ErrorPage 7

## F

FileCaching 9

## H

HungServerRecoverSecs 7

## I

Idempotent 7

Introduction to Plug-Ins 1

## K

KeepAliveSecs 10

## M

MatchExpression 9

MaxPostSize 8

MaxSkips 11

Microsoft-IIS (proxy) plug-in

- Configuration 6
  - proxying requests 3
  - proxying servlets 12
  - testing 13

## N

- Netscape (proxy) Plug-in 1
  - and clustering 16
  - MIME types 5
  - obj.conf file 8
  - sample obj.conf file 8

## P

- PathPrepend 4
- PathTrim 4
- proxying requests
  - Microsoft-IIS (proxy) plug-in 3

## Q

- QueryFromRequest 11

## R

- RequireSSLHostMatch 16

## S

- SecureProxy 16
- SSLHostMatchOID 16
- StatPath 7

## T

- Table 2-1, “Locations of Plug-In Shared Object Files,” on page 2-4 5

- TrustedCAFile 16

## W

- WebLogicCluster 3
- WebLogicHost 2
- WebLogicPort 2
- What are Plug-Ins 1
- WLForwardPath 10