BEA WebLogic Server™

Using Web Server Plug-Ins WithWebLogic Server

Release 8.1
Document Revised: September 3, 2002

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using Web Server Plug-Ins With WebLogic Server

Part Number	Document Revised	Software Version
N/A	September 9, 2002	BEA WebLogic Server Version 8.1

Contents

(NES) Plug-In

5. Parameters for Web Server Plug-Ins

A. Proxying Requests to Another Web Server

About This Document

This document explains use of plug-ins provided for proxying requests to third party administration servers. It is organized as follows:

- Chapter 1, "Overview of Using Web Server Plugins With WebLogic Server," describes the plug-ins available for use with WebLogic Server.
- Chapter A, "Proxying Requests to Another Web Server," describes use of WebLogic Server to serve as a proxy forwarding HTTP requests to other Web servers.
- Chapter 2, "Installing and Configuring the Apache HTTP Server Plug-In," explains how to install and configure the WebLogic Server Apache plug-in.
- Chapter 3, "Installing and Configuring the Microsoft Internet Information Server (IIS) Plug-In," explains how to install and conifgure the WebLogic Server plug-in for the Microsoft Internet Information Server.
- Chapter 4, "Installing and Configuring the Netscape Enterprise Server (NES) Plug-In," explains how to to install and configure the Netscape Enterprise Server proxy plug-in.
- Appendix 5, "Parameters for Web Server Plug-Ins," discusses the parameters for Web server plug-ins.

Audience

This document is intended mainly for system administrators who will be managing the WebLogic Server application platform and its various subsystems.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File—Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String` *`CustomerName`*`;`

Convention	Usage
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Example*s: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*: `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`
`|`	Separates mutually exclusive choices in a syntax line. *Example*: `java weblogic.deploy [list|deploy|undeploy|update] password {application} {source}`
`...`	Indicates one of the following in a command line: ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information
. . .	Indicates the omission of items from a code example or from a syntax line.

CHAPTER

1 Overview of Using Web Server Plugins With WebLogic Server

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

What Are Plug-Ins?

Plug-ins are small software programs that developers use to extend a WebLogic Server implementation. Plug-ins enableWebLogic Server to communicate with applications which have been deployed on a different Web Server. WebLogic Server can then handle those requests that require the dynamic functionality of WebLogic Server. These are usually 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

CHAPTER

2 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

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 those 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 alsoproxy requests 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 type of request that define additional behavior of the plug-in. For more information, see "Configuring the Apache HTTP Server Plug-In" on page 2-9.

Certifications

The Apache HTTP Server Plug-In is supported on Linux, Solaris, and HPUX11 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. The module is installed either as a Dynamic Shared Object (DSO) or as a statically linked module. (Installation as a statically linked module is only available for Apache version 1.3.x). There are separate instructions in this section for DSO and statically linked module installation.

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

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

1. Locate the shared object file for your platform.

   The Apache plug-in is distributed as a shared object (`.so`) for use on Solaris, Linux, and HPUX11 platforms. Each shared object file is distributed as a separate version, depending on the platform, whether or not SSL is to be used between the client and Apache, 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 shared object files are located in the following directories of your WebLogic Server installation:

   Solaris
   `WL_HOME/Server/lib/solaris`

   Linux
   `WL_HOME/Server/lib/linux`

   HPUX11
   `WL_HOME/Server/lib/hpux11`

   Windows (Apache 2.0 only)
   `WL_HOME\Server\bin\apache`

   where WL_HOME is the top-level installation directory for the WebLogic platform and the Server directory contains WebLogic Server installation files. The Plug-In for Apache 2.0.39 is not in the distribution, but can be found at http://dev2dev.bea.com/managed_content/direct/apache2.0/apache2.0.zip.

   Choose the appropriate shared object from the following table:

Apache Version	Regular Strength Encryption	128-bit Encryption
Standard Apache Version 1.x	`mod_wl.so`	`mod_wl128.so`

Apache Version	Regular Strength Encryption	128-bit Encryption
Apache w/ SSL/EAPI Version 1.x (Stronghold, modssl etc.)	`mod_wl_ssl.so`	`mod_wl128_ssl.so`
Apache + Raven Version 1.x Required because Raven applies frontpage patches that makes the plug-in incompatible with the standard shared object	`mod_wl_ssl_raven.so`	`mod_wl128_ssl_raven.so`
Standard Apache Version 2.x	`mod_wl_20.so`	`mod_wl28_20.so`

2. Enable the shared object.

   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 named `mod_so.c` that must be enabled before `mod_wl.so` is loaded. If you installed Apache using the supplied script, `mod_so.c` should already be enabled. To verify that `mod_so.c` is enabled, execute one of the following commands:

   For Apache 1.x, *`APACHE_HOME`*`\bin\httpd -l`

   For Apache 2.x, *`APACHE_HOME`*`\bin\Apache -l`

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

   This command lists all of the enabled modules. If `mod_so.c` is not listed, build your Apache HTTP Server from the source code, making sure that the following options are configured:

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

3. Install the Apache HTTP Server Plug-In in the Apache 1.x server with a support program called `apxs` (APache eXtenSion) that builds DSO-based modules outside of the Apache source tree, and adds the following line to the `httpd.conf` file:

   ```
   AddModule mod_so.c
   ```

   For Apache 2.x, copy the `mod_wl_20.so` file to the *`APACHE_HOME`*`\modules` directory rather than running apxs. For more information, see the Apache HTTP Server Version 2.0 documentation at `http://httpd.apache.org/docs-2.0/`.

4. For Apache 1.x, in your WebLogic 1.x Server installation, use a command shell to navigate to the directory 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 Apache 2.x, add the following line to your *`APACHE_HOME`*`/conf/httpd.conf` file manually:

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

5. Configure any additional parameters in the Apache `httpd.conf` configuration file as described in the section "Configuring the Apache HTTP Server Plug-In" on page 2-9. The `httpd.conf` file allows you to customize the behavior of the Apache HTTP Server Plug-In.

6. Verify the syntax of the *`APACHE_HOME`*`\conf\httpd.conf` file with one of the following commands:

   For Apache 1.x, `APACHE_HOME\bin\apachect1 configtest`

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

The output of this command indicates any errors in your `httpd.conf` file.

7. Restart Weblogic Server.
8. Start (or restart if you have changed the configuration) Apache HTTP Server.
9. Test the Apache 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/
   ```

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. 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:

   Solaris
   `WL_HOME/Server/lib/solaris`

   Linux
   `WL_HOME/Server/lib/linux`

   HPUX11
   `WL_HOME/Server/lib/hpux11`

   Choose the appropriate shared object from the following table.

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

2. Unpack the Apache Plug-In distribution using the following command:

   ```
   tar -xvf apache_1.3.x.tar
   ```

3. Within the unpacked distribution switch to the `src/modules` directory.
4. Create a directory called `weblogic`.

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 (see step 1.) `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. Follow steps 4 through 8, in "Installing the Apache HTTP Server Plug-In as a Dynamic Shared Object".

Configuring the Apache HTTP Server Plug-In

After you install the plug-in in the Apache HTTP server, edit the `httpd.conf` file to configure the Apache plug-in. Editing the `httpd.conf` file informs the Apache Web server that it should load the native library for the plug-in as an Apache module and also describes which requests 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:

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).
2. 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
   ```

3. For Apache 2.x, add the following line to the httpd.conf file:

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

4. 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>
   ```

5. If you want to proxy requests by MIME type, also add a `MatchExpression` line to the `IfModule` block. (You can proxy requests by path in addition to or instead of proxying by MIME type. Proxying by path takes precedence over proxying by MIME type. To configure proxying requests by path, see step 6.)

   For example, the following `IfModule` block for a non-clustered WebLogic Server specifies that all files with MIME type `.jsp` are proxied:

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

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

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

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

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

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

6. If you want 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
   </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 5-2. 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*.

Alternative Procedure for Editing the httpd.conf File

- As an alternative to the procedure in "Editing the httpd.conf File" on page 2-9, you can define parameters in a separate file called `weblogic.conf` file that is *included* in the `IfModule` block. Using this included file may help modularize your configuration. For example:

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

  Note: Defining parameters in an *included* file is not supported when using SSL between Apache HTTP Server Plug-In and WebLogic Server.

- Enter each parameter on a new line. Do not put an '=' between the 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, you must define it in an `IfModule` block.
- If you use the `<files>` block in addition to the `<location>` block to match requests and you are using Stronghold SSL with virtual hosting, `MatchExpression` is ignored and the rules defined in the `<files>` and `<location>` blocks are applied to the request. If you are not using Stronghold, rules in the `<location>` block overrule those in a `<files>` block.
- If you use the `<VirtualHost>` block, you must place all the configuration parameters (`MatchExpression`, for example) for each virtual host within its `<VirtualHost>` block.
- BEA recommends that you use the `MatchExpression` statement instead of the `<files>` block.

Template for the Apache HTTP Server httpd.conf File

This section contains a sample `httpd.conf` file. You can use this sample as a template that you can modify to suit your environment and server. Lines beginning with # are comments. Note that Apache HTTP Server is not case sensitive, and that the `LoadModule` and `AddModule` lines are automatically added by the `apxs` utility.

```
#####################################################

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 w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
</IfModule>
```

Sample httpd.conf Configuration Files

Instead of defining parameters in the `location` block of your `httpd.conf` file, if you prefer, you can use a `weblogic.conf` file that is loaded by the `IfModule` in the `httpd.conf` file. The following examples 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 w1s1.com:7001,w1s2.com:7001,w1s3.com:7001
  ErrorPage http://myerrorpage.mydomain.com
  MatchExpression *.jsp
```

```
</IfModule>
#####################################################
```

Example Using Multiple WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

<IfModule mod_weblogic.c>
  MatchExpression *.jsp WebLogicHost=myHost|WebLogicPort=7001|Debug=ON
  MatchExpression *.html WebLogicCluster=myHost1:7282,myHost2:7283|ErrorPage=
    http://www.xyz.com/error.html
</IfModule>
```

Example Without WebLogic Clusters

```
# These parameters are common for all URLs which are
# directed to the current module. If you want to override
# these parameters for each URL, you can set them again in
# the <Location> or <Files> blocks (Except WebLogicHost,
# WebLogicPort, WebLogicCluster, and CookieName.)

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

Example Configuring IP-Based Virtual Hosting

```
NameVirtualHost 172.17.8.1
<VirtualHost goldengate.domain1.com>
WebLogicCluster tehama1:4736,tehama2:4736,tehama:4736
PathTrim /x1
ConnectTimeoutSecs 30
</VirtualHost>
<VirtualHost goldengate.domain2.com>
```

```
WeblogicCluster green1:4736,green2:4736,green3:4736
PathTrim /y1
ConnectTimeoutSecs 20
</VirtualHost>
```

Example Configuring Name-Based Virtual Hosting With a Single IP Address

```
<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
         <Location / >
               SetHandler weblogic-handler
               WebLogicCluster 162.99.55.71:7001,162.99.55.72:7001
               Idempotent ON
               Debug ON
               DebugConfigInfo ON
        </Location>
</VirtualHost>

<VirtualHost 162.99.55.208>
        ServerName myserver.mydomain.com
       <Location / >
                SetHandler weblogic-handler
                WebLogicHost russell
                WebLogicPort 7001
                Debug ON
                DebugConfigInfo ON
        </Location>
</VirtualHost>
```

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. In addition, the SSL protocol allows the plug-in to authenticate itself to WebLogic Server to ensure that information is passed to a trusted principal.

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.

Implementing two-way SSL between Apache and the HTTP Client:

1. Configure Apache HTTP Server to request a client certificate. The certificate is stored as one of the following request attributes:
   - `javax.net.ssl.peer_certificates`
     returns a `weblogic.security.X509Certificate` certificate
   - `java.security.cert.X509Certificate`
     returns a `java.security.cert.X509` certificate
2. Access the certificate by reading the request attribute, for example:

   ```
   request.getAttribute("javax.net.ssl.peer_certificates");
   ```

3. In WebLogic Server, authenticate the user with the `weblogic.security.acl.certAuthenticator.authenticate() method`.

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/docs81b/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/docs81b/secmanage/ssl.html.
3. In the Apache Server, set the `WebLogicPort` parameter in the `httpd.conf` file to the 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 parameters, see "SSL Parameters for Web Server Plug-Ins" on page 5-12.

Issues with SSL-Apache Configuration

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

- The PathTrim (see page A-3) parameter 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>
```

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 2-20 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 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 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.

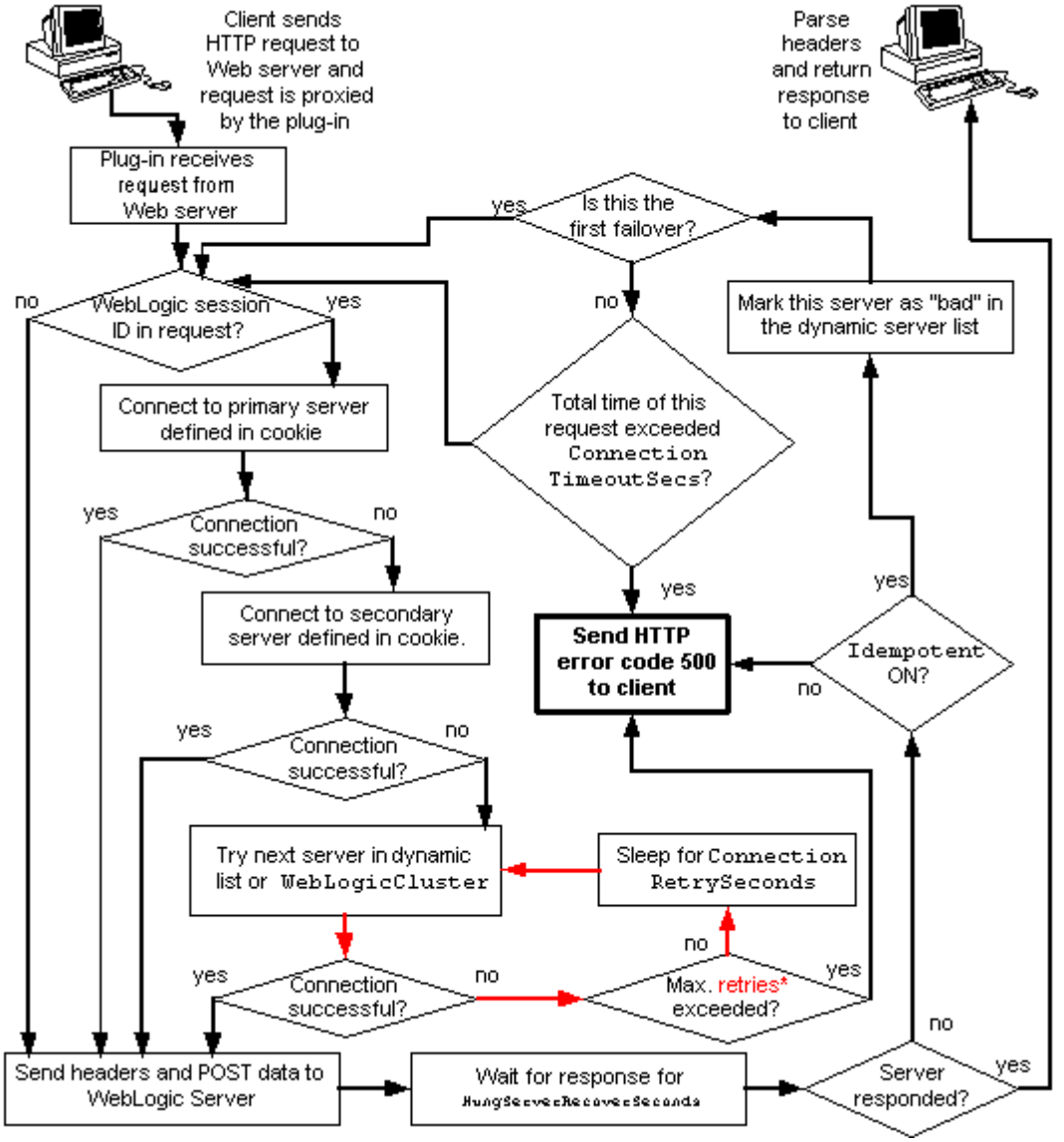
Failover, Cookies, and HTTP Sessions

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 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. For more information, see Figure 2-1 “Connection Failover” on page 2-20.

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 `ConnectTimeoutSecs` ÷ `ConnectRetrySecs`.


Client sends HTTP request to Web server and request is proxied by the plug-in
Plug-in receives request from Web server
WebLogic session ID in request?
yes
no
Connect to primary server defined in cookie
Connection successful?
yes
no
Connect to secondary server defined in cookie.
Connection successful?
yes
no
Try next server in dynamic list or WebLogicCluster
Sleep for Connection RetrySeconds
Connection successful?
yes
no
Max. retries* exceeded?
no
yes
Send headers and POST data to WebLogic Server
Wait for response for HungServerRecoverSeconds
Server responded?
no
yes
Total time of this request exceeded Connection TimeoutSecs?
yes
no
Is this the first failover?
yes
no
Send HTTP error code 500 to client
Idempotent ON?
yes
no
Mark this server as "bad" in the dynamic server list
Parse headers and return response to client

CHAPTER

3 Installing and Configuring the Microsoft Internet Information Server (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

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 by employing a re-usable 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 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`" 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 either based on the URL of the request or a portion of the URL. This is called proxying by *path*. You can also proxy a request based on the *MIME type* of the requested file, called proxying by file extension. You can also 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 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 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 by IIS. This directory must also contain the `iisproxy.ini` file that you will create in step 8.
2. Start the IIS Internet Service Manager by selecting it from the Microsoft IIS Start menu.

3. In the left panel of the Service Manager, select your website (the default is "Default Web Site").
4. Click the "Play" arrow in the toolbar to start.
5. Open the properties for the selected website by holding the right mouse button down over the website selection in the left panel.
6. In the Properties panel, select the Home Directory tab, and click the Configuration button in the Applications Settings section.
7. 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 5-2.

   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. The same directory where `iisproxy.dll` is located.

   b. 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 there, 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. The directory `c:\weblogic`, if it exists.

8. 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 9. .)

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

   b. In the dialog box, browse to find the "`iisproxy.dll`" file.

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

d. Deselect the "Check that file exists" check box.

e. Set the Method exclusions as needed to create a secure installation.

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

g. 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
```

9. 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 3-7.

   To configure proxying by path:

   a. 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`).

   b. Register `.wlforward` as a special file type to be handled by `iisproxy.dll`.

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

   d. 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`.

e. 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.

f. 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.

10. 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.

11. Optionally, enable HTTP tunneling by following the instructions for proxying by path (see step 9 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.

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

You do not need to use the PathTrim parameter.

Note: The only time you need to use HTTP-tunneling is when you connect through an applet through IIS/NES to WebLogic Server and and use http as the protocol instead of t3. (For example, http:// as the protocol in the provider URL instead of t3://.)

12. 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 5-2.
13. 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 3-10.

Proxying Requests from Multiple Virtual Websites to WebLogic Server

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

1. Create a new directory for the virtual directories. This directory will contain `dll` and `ini` files used to define the proxy.
2. Copy `iisforward.dll` to the directory you created in step1.
3. Register the `iisforward.dll` for each Website with IIS.
4. Create a file called `iisforward.ini`. Place this file in the same directory that contains `iisforward.dll`. This file should contain the following entry for each virtual website defined in IIS:

   `vhost`*`N`*`=`*`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 7. in "Proxying Requests". Copy this `iispoxy.ini` file to the directory you created in step 1.
6. Copy `iisproxy.dll` to the directory you created in step 1.
7. In IIS, set the value for the Application Protection option to high (isolated).

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”.

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. In addition, the SSL protocol allows the Microsoft Internet Information Server Plug-In to authenticate itself to the Microsoft Internet Information Server to ensure that information is passed to a trusted principal.

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* at http://e-docs.bea.com/wls/docs81b/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/docs81b/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 5-12.

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 8. 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 at `http://e-docs.bea.com/wls/docs81b/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 3-14 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.

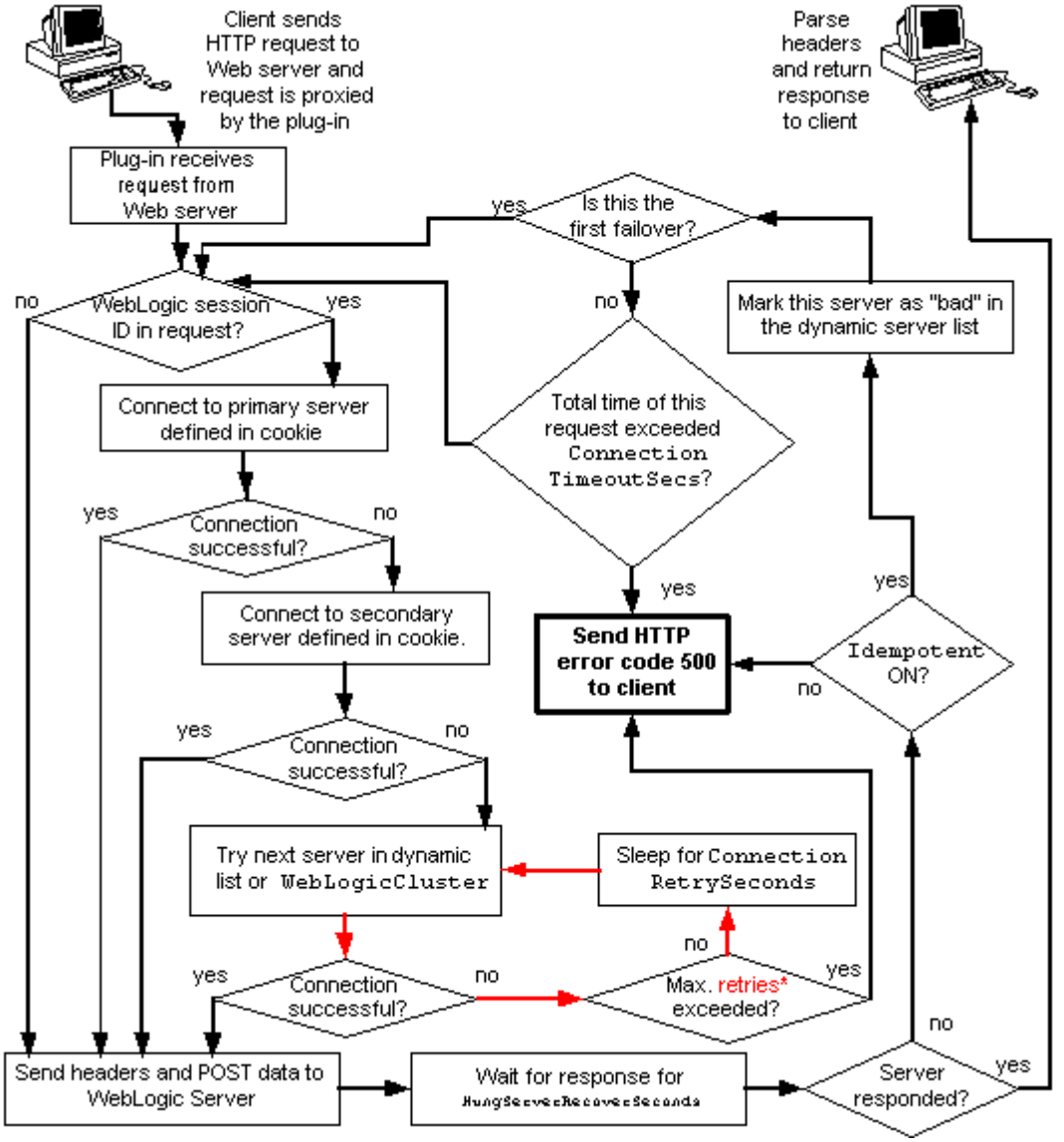
Failover, Cookies, and HTTP Sessions

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 3-14.

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 `ConnectTimeoutSecs` ÷ `ConnectRetrySecs`.


Client sends HTTP request to Web server and request is proxied by the plug-in
Plug-in receives request from Web server
WebLogic session ID in request?
yes
no
Connect to primary server defined in cookie
Connection successful?
yes
no
Connect to secondary server defined in cookie.
Connection successful?
yes
no
Try next server in dynamic list or WebLogicCluster
Sleep for Connection RetrySeconds
Connection successful?
yes
no
Max. retries* exceeded?
no
yes
Send headers and POST data to WebLogic Server
Wait for response for HungServerRecoverSeconds
Server responded?
no
yes
Total time of this request exceeded Connection TimeoutSecs?
yes
no
Is this the first failover?
yes
no
Send HTTP error code 500 to client
Idempotent ON?
yes
no
Mark this server as "bad" in the dynamic server list
Parse headers and return response to client

CHAPTER

4 Installing and Configuring the Netscape Enterprise Server (NES) 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

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`* represets 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 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 4-9, 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 5-2. 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 5-2.

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

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

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

   Add the following object definition to the `obj.conf` file, substituting the WebLogic Server host name and the WebLogic Server port number, or the name of a WebLogic Cluster that you wish to handle HTTP tunneling requests.

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

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 *Configuring Web Application Components* at http://e-docs.bea.com/wls/docs81b/webapp/components.html.

Guidelines for Modifying the obj.conf File

To use the Netscape Enterprise Server Plug-In, you must make several modifications to the NES `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
#
# This configuration means that any file with the extension
# ".jsp" are proxied to WebLogic. Then you must add the
# Service line for this extension to the Object "default",
# which should already exist in your obj.conf file:

<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicHost=localhost WebLogicPort=7001 PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

# The following directive enables HTTP-tunneling of the
# WebLogic protocol through the NES plug-in.

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

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

Sample obj.conf File (Using a WebLogic Cluster)

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

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

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

```
## ------------ BEGIN SAMPLE OBJ.CONF CONFIGURATION ---------
# (using a WebLogic Cluster)
#
# The following line locates the NES library for loading at
# startup, and identifies which functions within the library are
# NES functions.  Verify the path to the library (the value
# of the shlib=<...> parameter) and that the file is
# readable, or the server fails to start.

Init fn="load-modules" funcs="wl_proxy,wl_init"\
 shlib=/usr/local/netscape/plugins/libproxy.so
Init fn="wl_init"

# Configure which types of HTTP requests should be handled by the
# NES module (and, in turn, by WebLogic).  This is done
# with one or more "<Object>" tags as shown below.

# Here we configure the NES module to pass requests for
# "/weblogic" to a cluster of WebLogic Servers.

<Object name="weblogic" ppath="*/weblogic/*">
Service fn=wl_proxy \
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001" PathTrim="/weblogic"
</Object>

# Here we configure the plug-in so that requests that
# match "/servletimages/" are handled by the
# plug-in/WebLogic.
```

```
<Object name="si" ppath="*/servletimages/*">
Service fn=wl_proxy \
WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001"
</Object>

# This Object directive works by file extension rather than
# request path. To use this configuration, you must also add
# a line to the mime.types file:
#
# type=text/jsp             exts=jsp
#
# This configuration means that any file with the extension
# ".jsp" is proxied to WebLogic. Then you must add the
# Service line for this extension to the Object "default",
# which should already exist in your obj.conf file:

<Object name=default>
NameTrans fn=pfx2dir from=/ns-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn=pfx2dir from=/mc-icons\
 dir="c:/Netscape/SuiteSpot/ns-icons"
NameTrans fn="pfx2dir" from="/help" dir=\
 "c:/Netscape/SuiteSpot/manual/https/ug"
NameTrans fn=document-root root="c:/Netscape/SuiteSpot/docs"
Service method="(GET|HEAD|POST|PUT)" type=text/jsp fn=wl_proxy\
 WebLogicCluster="myweblogic.com:7001,yourweblogic.com:7001,\
 theirweblogic.com:7001",PathPrepend=/jspfiles
PathCheck fn=nt-uri-clean
PathCheck fn="check-acl" acl="default"
PathCheck fn=find-pathinfo
PathCheck fn=find-index index-names="index.html,home.html"
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap\
 fn=imagemap
Service method=(GET|HEAD) \
 type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

# The following directive enables HTTP-tunneling of the
# WebLogic protocol through the NES plug-in.

<Object name="tunnel" ppath="*/HTTPClnt*">
Service fn=wl_proxy WebLogicCluster="myweblogic.com:7001,\
 yourweblogic.com:7001,theirweblogic.com:7001"
</Object>
```

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

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. In addition, the SSL protocol allows the WebLogic Server proxy plug-in to authenticate itself to the Netscape Enterprise Server to ensure that information is passed to a trusted principal.

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* at http://e-docs.bea.com/wls/docs81b/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/docs81b/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 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 5-12.

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.

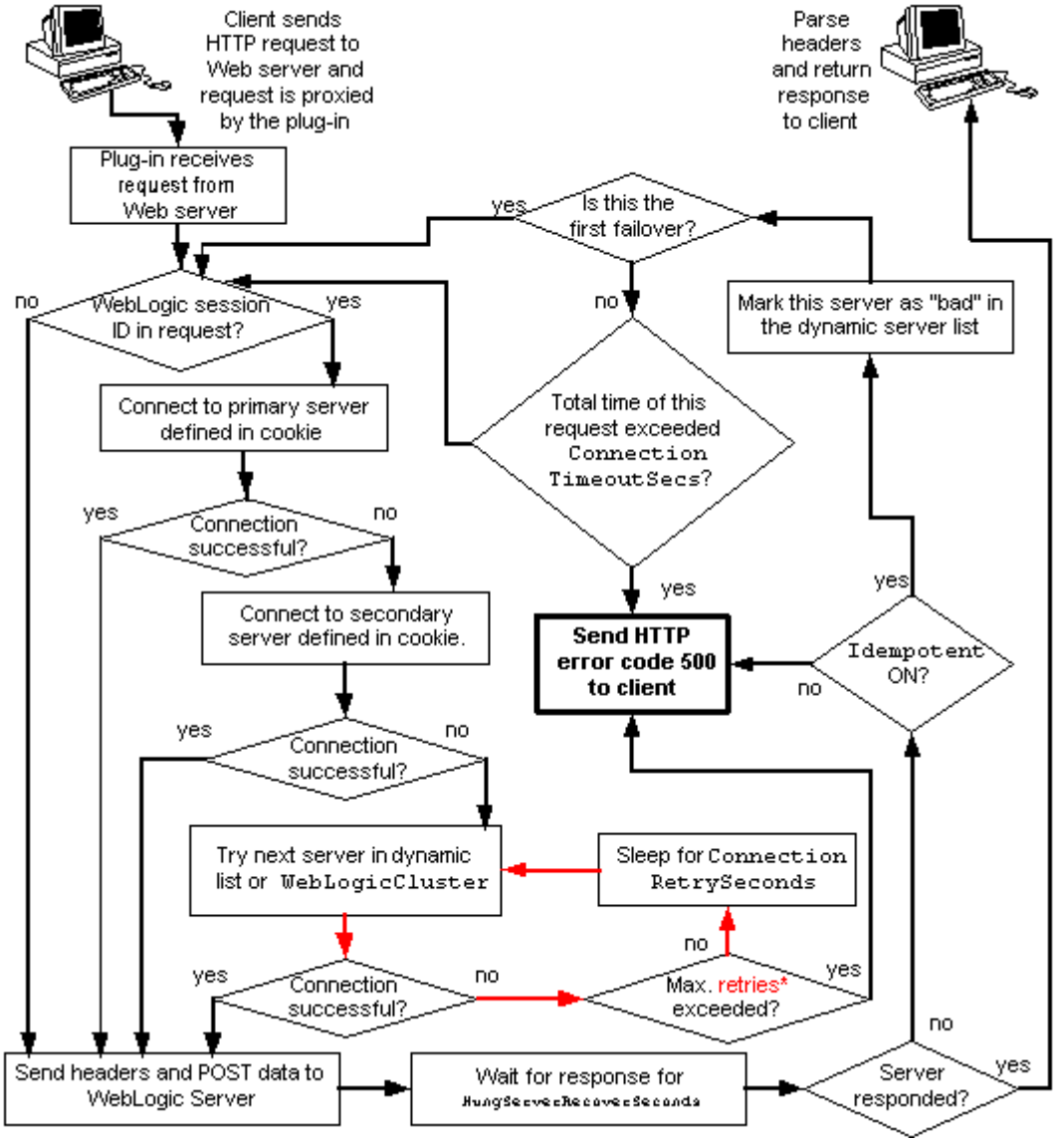
Failover, Cookies, and HTTP Sessions

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.

Figure 4-1 Connection Failover

*The Maximum number of retries allowed in the red loop is equal to `ConnectTimeoutSecs` ÷ `ConnectRetrySecs`.


Client sends HTTP request to Web server and request is proxied by the plug-in
Plug-in receives request from Web server
WebLogic session ID in request?
yes
no
Connect to primary server defined in cookie
Connection successful?
yes
no
Connect to secondary server defined in cookie.
Connection successful?
yes
no
Try next server in dynamic list or WebLogicCluster
Sleep for Connection RetrySeconds
Connection successful?
yes
no
Max. retries* exceeded?
no
yes
Send headers and POST data to WebLogic Server
Wait for response for HungServerRecoverSeconds
Server responded?
no
yes
Total time of this request exceeded Connection TimeoutSecs?
yes
no
Is this the first failover?
yes
no
Send HTTP error code 500 to client
Idempotent ON?
yes
no
Mark this server as "bad" in the dynamic server list
Parse headers and return response to client

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.

CHAPTER

5 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 Internet Information Server (IIS) Plug-In" on page 3-1
- "Installing and Configuring the Netscape Enterprise Server (NES) Plug-In" on page 4-1

General Parameters for Web Server Plug-Ins

Note: Parameters are case sensitive.

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 *Configuring the SSL Protocol* at http://e-docs.bea.com/wls/docs81b/secmanage/ssl.html) and set the `SecureProxy` parameter to `ON`). If you are using a WebLogic Cluster, use the `WebLogicCluster` parameter instead of `WebLogicPort`.

Parameter	Default	Description
`WebLogicCluster` (Required when proxying to a cluster of WebLogic Servers.)	`none`	List of WebLogic Servers that can be used in a cluster for load balancing. The cluster list is a comma-delimited list of host:port entries. For example: `WebLogicCluster myweblogic.com:7001, yourweblogic.com:7001,theirweblogic.com:7001` If you are using SSL between the plug-in and WebLogic Server, set the port number to the SSL listen port (see *Configuring the SSL Protocol* at http://e-docs.bea.com/wls/docs81b/secmanage/ssl.html) and set the `SecureProxy` parameter to `ON`. Use `WebLogicCluster` instead of the `WebLogicHost` and `WebLogicPort` parameters. WebLogic Server looks first for the `WebLogicCluster` parameter. If not found, it looks for and uses `WebLogicHost` and `WebLogicPort`. The plug-in does a simple round-robin between all available cluster members. The cluster list specified in this property is a starting point for the dynamic cluster list that the server and plug-in maintain. WebLogic Server and the plug-in work together to update the cluster list automatically with new, failed, and recovered cluster members. Microsoft Internet Information Server only: You can disable the use of the dynamic cluster list by setting the `DynamicServerList` parameter to `OFF` The plug-in directs HTTP requests containing a cookie, URL-encoded session, or a session stored in the POST data to the server in the cluster that originally created the cookie.

`PathTrim`	`null`	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
		`http://myWeb.server.com/weblogic/foo`
		is passed to the plug-in for parsing and if `PathTrim` has been set to strip off `/weblogic` before handing the URL to WebLogic Server, the URL forwarded to WebLogic Server is:
		`http://myweblogic.server.com:7001/foo`

`PathPrepend`	`null`	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.

Parameter	Default	Description
`ConnectTimeoutSecs`	`10`	Maximum time in seconds that the plug-in should attempt to connect to the WebLogic Server host. Make the value greater than `ConnectRetrySecs`. If `ConnectTimeoutSecs` expires without a successful connection, even after the appropriate retries (see `ConnectRetrySecs`), an `HTTP 503/Service Unavailable` response is sent to the client. You can customize the error response by using the `ErrorPage` parameter.
`ConnectRetrySecs`	`2`	Interval in seconds that the plug-in should sleep between attempts to connect to the WebLogic Server host (or all of the servers in a cluster). Make this number less than the `ConnectTimeoutSecs`. The number of times the plug-in tries to connect before returning an `HTTP 503/Service Unavailable` response to the client is calculated by dividing `ConnectTimeoutSecs` by `ConnectRetrySecs`. To specify no retries, set `ConnectRetrySecs` equal to `ConnectTimeoutSecs`. However, the plug-in attempts to connect at least twice. You can customize the error response by using the `ErrorPage` parameter.

Parameter	Default	Description
Debug	OFF	Sets the type of logging performed for debugging operations. It is not advisable to switch on these debugging options in production systems. The debugging information is written to the /tmp/wlproxy.log file on UNIX systems and c:\TEMP\wlproxy.log on Windows NT/2000 systems. You can override this location and filename by setting the WLLogFile parameter to a different directory and file. For Debug to work correctly, the system administrator must ensure that the tmp or TEMP directory has write permission assigned to the user who is logged in to the server. You can set any of the following logging options (the HFC, HTW, HFW, and HTC options may be set in combination by entering them separated by commas, for example "HFC, HTW"): ON — The plug-in logs only informational and error messages. OFF — No debugging information is logged. HFC — The plug-in logs headers from the client, informational, and error messages. HTW — The plug-in logs headers sent to WebLogic Server, informational messages, and error messages. HFW — The plug-in logs headers sent from WebLogic Server, informational messages, and error messages. HTC — The plug-in logs headers sent to the client, informational messages, and error messages. ALL — The plug-in logs headers sent to and from the client, headers sent to and from WebLogic Server, information messages, and error messages.

Parameter	Default	Description
`WLLogFile`	See the `Debug` parameter	Specifies path and file name for the log file that is generated when the `Debug` parameter is set to `ON`. You must create this directory before setting this parameter.
`DebugConfigInfo`	`OFF`	Enables the special query parameter "`__WebLogicBridgeConfig`". Use it to get details about configuration parameters from the plug-in. For example, if you enable "`__WebLogicBridgeConfig`" by setting `DebugConfigInfo` and then send a request that includes the query string `?__WebLogicBridgeConfig`, then the plug-in gathers the configuration information and run-time statistics and returns the information to the browser. The plug-in does not connect to WebLogic Server in this case. This parameter is strictly for debugging and the format of the output message can change with releases. For security purposes, keep this parameter turned `OFF` in production systems.
`StatPath` (Not available for the Microsoft Internet Information Server Plug-In)	`false`	If set to `true`, the plug-in checks the existence and permissions of the translated path ("Proxy-Path-Translated") of the request before forwarding the request to WebLogic Server. If the file does not exist, an `HTTP 404 File Not Found` response is returned to the client. If the file exists but is not world-readable, an `HTTP 403/Forbidden` response is returned to the client. In either case, the default mechanism for the Web server to handle these responses fulfills the body of the response. This option is useful if both the WebLogic Server Web Application and the Web Server have the same document root. You can customize the error response by using the `ErrorPage` parameter.

Parameter	Default	Description
`ErrorPage`	`none`	You can create your own error page that is displayed when your Web server is unable to forward requests to WebLogic Server. You can set this parameter in one of two ways: ■ As a relative `URI` (file name). The plug-in automatically adds the context path of the Web Application returning the error to the URI. Depending on how you configure proxying (by MIME type or path) the request for the error page might *not* be proxied to WebLogic Server. ■ As an absolute URL (recommended). Using an absolute URL to the error page will always proxy the request to the correct resource on WebLogic Server. For example: `http://host:port/myWebApp/ErrorPage.html`.
`HungServerRecoverSecs`	`300`	Defines the amount of time the plug-in waits for a response to a request from WebLogic Server. The plug-in waits for `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. Minimum value: `10` Maximum value: `600`
`Idempotent`	`ON`	When set to `ON` and if the servers do not respond within `HungServerRecoverSecs`, the plug-ins fail over. 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.
`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 `<session-descriptor>` (see `http://e-docs.bea.com/wls/docs81b/webapp/weblogic_xml.html#session-descriptor`) element.

Parameter	Default	Description
`DefaultFileName`	`none`	If the URI is "`/`" then the plug-in performs the following steps: 1. Trims the path specified with the `PathTrim` parameter. 2. Appends the value of `DefaultFileName`. 3. Prepends the value specified with `PathPrepend`. This procedure prevents redirects from WebLogic Server. Set the `DefaultFileName` to the default welcome page of the Web Application in WebLogic Server to which requests are being proxied. For example, If the `DefaultFileName` is set to `welcome.html`, an HTTP request like "`http://somehost/weblogic`" becomes "`http://somehost/weblogic/welcome.html`". For this parameter to function, the same file must be specified as a welcome file in all the Web Applications to which requests are directed. For more information, see "Configuring Welcome Pages" at `http://e-docs.bea.com/wls/docs81b/webapp/components`. 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.
`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.

`MatchExpression`
(Apache HTTP Server only)
	`none`	When proxying by MIME type, set the filename pattern inside of an `IfModule` block using the `MatchExpression` parameter.
		Example when proxying by MIME type:
		`<IfModule mod_weblogic.c>`
		`  MatchExpression *.jsp`
		`   WebLogicHost=myHost|paramName=value`
		`</IfModule>`
		Example when proxying by path:
		`<IfModule mod_weblogic.c>`
		`  MatchExpression /weblogic`
		`   WebLogicHost=myHost|paramName=value`
		`</IfModule>`

Parameter	Default	Description
`FileCaching`	`ON`	When set to `ON`, and the size of the POST data in a request is greater than 2048 bytes, the POST data is stored on disk in a temporary file and forwarded to WebLogic Server in chunks of 8192 bytes. Setting `FileCaching` 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. 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.
`FilterPriorityLevel` (Microsoft Internet Information Server only)	2	The values for this parameter are 0 (low), 1 (medium), and 2 (high). The default value is 2. This priority should be put in iisforward.ini file. If you do not use virtual host, but choose to set this property, you must use iisforward.ini file.
`WLExcludePathOrMimeTyp e` (Apache and Microsoft IIS only)	none	This parameter allows you make exclude certain requests from proxying.
`WlForwardPath` (Microsoft Internet Information Server only)	null	If `WlForwardPath` is set to "/" all requests are proxied. To forward any requests starting with a particular string, set `WlForwardPath` to the string. For example, setting `WlForwardPath` to `/weblogic` forwards all requests starting with `/weblogic` to Weblogic Server. This parameter is required if you are proxying by path. You can set multiple strings by separating the strings with commas. For example: `WlForwardPath=/weblogic,/bea`.
`KeepAliveSecs` (Does not apply to Apache HTTP Server version 1.3.x)	`30`	The length of time after which an inactive connection between the plug-in and WebLogic Server is closed. You must set `KeepAliveEnabled` to `true` for this parameter to be effective. The value of this parameter must be less than or equal to the value of the Duration field set in the Administration Console on the Server/HTTP tab, or the value set on the `server` Mbean with the `KeepAliveSecs` attribute.

Parameter	Default	Description
`KeepAliveEnabled` (Does not apply to Apache HTTP Server version 1.3.x)	`true`	Enables pooling of connections between the plug-in and WebLogic Server.
`QueryFromRequest` (Apache HTTP Server only)	`OFF`	When set to `ON`, specifies that the Apache plug-in use `(request_rec *)r->the request` to pass the query string to WebLogic Server. (For more information, see your Apache documentation.) This behavior is desirable in the following situations: ■ When a Netscape version 4.x browser makes requests that contain spaces in the query string ■ If you are using Raven Apache 1.5.2 on HP When set to `OFF`, the Apache plug-in uses `(request_rec *)r->args` to pass the query string to WebLogic Server.
`MaxSkips` (Not available for Apache 1.3.x)	10	Valid only if `DynamicServerList` is set to `OFF`. If a WebLogic Server listed in either the `WebLogicCluster` parameter or a dynamic cluster list returned from WebLogic Server fails, the failed server is marked as "bad" and the plug-in attempts to connect to the next server in the list. `MaxSkips` sets the number of attempts 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).
`DynamicServerList`	ON	When set to `OFF`, the plug-in ignores the dynamic cluster list used for load balancing requests proxied from the plug-in and only uses the static list specified with the `WebLogicCluster` parameter. Normally this parameter should remain set to `ON`. There are some implications for setting this parameter to `OFF`: ■ If one or more servers in the static list fails, the plug-in could waste time trying to connect to a dead server, resulting in decreased performance. ■ If you add a new server to the cluster, the plug-in cannot proxy requests to the new server unless you redefine this parameter. WebLogic Server automatically adds new servers to the dynamic server list when they become part of the cluster.

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

SSL Parameters for Web Server Plug-Ins

Note: Parameters are case sensitive.

Parameter	Default	Description
`SecureProxy`	`OFF`	Set this parameter to `ON` to enable the use of the SSL protocol for all communication between the plug-in and WebLogic Server. Remember to configure a port on the corresponding WebLogic Server for the SSL protocol before defining this parameter. This parameter may be set at two levels: in the configuration for the main server and—if you have defined any virtual hosts—in the configuration for the virtual host. The configuration for the virtual host inherits the SSL configuration from the configuration of the main server if the setting is not overridden in the configuration for the virtual host.
`TrustedCAFile`	`none`	Name of the file that contains the digital certificates for the trusted certificate authorities for the plug-in. This parameter is required if the `SecureProxy` parameter is set to `ON`. The filename must include the full directory path of the file.
`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`	The ASN.1 Object ID (OID) that identifies which field in the Subject Distinguished Name of the peer digital certificate is to be used to perform the host match comparison. The default for this parameter corresponds to the `CommonName` field of the Subject Distinguished Name. Common OID values are: ■ Sur Name—23 ■ Common Name—22 ■ Email—13 ■ Organizational Unit—30 ■ Organization—29 ■ Locality—26

CHAPTER

A Proxying Requests to Another Web Server

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

Overview of Proxying Requests to Another Web Server

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

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

Setting Up a Proxy to a Secondary Web Server

To set up a proxy to a secondary HTTP server:

1. Register the `proxy` servlet in your Web Application deployment descriptor (see "Sample web.xml for Use with ProxyServlet" on page A-3). The Web Application must be the default Web Application of the server instance that is responding to requests. The class name for the proxy servlet is `weblogic.t3.srvr.HttpProxyServlet`. For more information, see Assembling and Configuring Web Applications at `http://e-docs.bea.com/wls/docs81b/webapp/index.html`.

2. Define an initialization parameter for the `ProxyServlet` with a `<param-name>` of `redirectURL` and a `<param-value>` containing the URL of the server to which proxied requests should be directed.

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

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

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

Sample Deployment Descriptor for the Proxy Servlet

The following is an sample of a Web Applications deployment descriptor for using the Proxy Servlet.

Listing 5-1 Sample web.xml for Use with ProxyServlet

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
 //DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

<servlet>
  <servlet-name>ProxyServlet</servlet-name>
  <servlet-class>weblogic.t3.srvr.HttpProxyServlet</servlet-class
>

  <init-param>
    <param-name>redirectURL</param-name>
    <param-value>
       <scheme>server:port
    </param-value>
  </init-param>

</servlet>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.jsp</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
  <servlet-name>ProxyServlet</servlet-name>
  <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```

Index

P

Q

R

S

T

V

W