Administrator’s Guide

Sun™ ONE Web Proxy Server

Version 3.6 SP3 for UNIX

816-6826-10
May 2003

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des États-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régi par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

Preface

Welcome to Sun™ Open Net Environment (Sun ONE) Web Proxy Server (formerly, iPlanet Web Proxy Server) and the Internet. iPlanet Web Proxy Server is a high-performance server software product. It is designed for replicating and filtering access to web-based content.

What iPlanet Web Proxy Server Provides

The rapid growth of *clients* (web browsers such as Netscape Navigator) and servers for the World Wide Web and corporate *intranets* has opened new opportunities for sharing information, collaborating, and developing network-oriented applications. At the same time, for network administrators this growth has raised new issues about network congestion and security, about how to ensure fast and reliable service for mission-critical applications, and about how to control access to restricted network resources. iPlanet Web Proxy Server is designed to address these problems.

For companies that do business on the Internet or the web, a proxy server can act as a transparent intermediary between individual clients and the servers that contain the information the clients want. A proxy server allows an organization or company to provide controlled Internet access for internal users who would otherwise be blocked by a security firewall; a proxy server working in reverse can also let the organization regulate access from external clients, as a refinement to firewall security protection.

iPlanet Web Proxy Server has an added advantage—it provides *replication-on-demand* by intelligently caching frequently accessed documents, thereby conserving network bandwidth and dramatically increasing response time. This important feature makes iPlanet Web Proxy Server valuable even for companies that have full web access.

iPlanet Web Proxy Server is the first commercial proxy server to provide caching. The server’s content filtering capabilities let you fine-tune access control (for example, by the individual user or server trying to gain access), and provide the ability to track who accesses which server, and whether or not they are successful.

iPlanet Web Proxy Server enhances network security and reliability and provides advanced server management features that let you create intelligent proxy networks that are totally transparent to users.

iPlanet Web Proxy Server is fully compatible with Netscape Navigator, the other servers in Netscape SuiteSpot, and other HyperText Transfer Protocol (HTTP) clients and servers.

iPlanet Web Proxy Server offers these features:

- It provides *proxying*; that is, safe passage through the firewall for secure and unsecure protocols. The UNIX version of the proxy server proxies HTTP 1.1, FTP, Gopher, Secure Sockets Layer (SSL), HyperText Transfer Protocol-Secure (HTTPS), Web Distributed Authoring and Versioning (WebDAV), and Netscape News Transfer Protocol-Secure (NNTPS). iPlanet Web Proxy Server provides a SOCKS v5 daemon for the generic tunneling of many other protocols and applications, including streaming media. It also provides support for arbitrary methods.

- It provides *replication*; it caches documents by writing them to a local file system. If a document is requested more than once, subsequent requests are faster because the proxy doesn't have to contact the remote server repeatedly. Replication can dramatically reduce network traffic and associated costs. iPlanet Web Proxy Server also provides distributed caching so that multiple proxy servers can operate as a single logical cache for load-balancing and failover, and dynamic proxy routing so that the proxy server can query other caches to determine if a document is available.

- If used as a *reverse proxy*, it can help your host machine handle a high volume of requests while reducing their effect on the host machine's performance. A reverse proxy lets the content server reside safely inside the firewall while the reverse proxy acts as a server outside the firewall. The UNIX proxy server can also be used as a secure reverse proxy so that all information is securely transferred between clients, proxy, and content server.

- It can *filter* client transactions by controlling access to remote servers and protocols and by limiting access to specific documents or sites based on user names, URLs (Universal Resource Locators), and client host names (or IP [Internet Protocol] addresses).

- It provides *flexible logging* of client transactions, including client host names or IP addresses, access dates and times, accessed URLs, byte counts of all transferred data, routing information, and the success of transactions.

- It provides key *server management* features such as remote management, SNMP (Simple Network Management Protocol), advanced logging and reporting, cluster management of user and group information through LDAP v3, automatic proxy configuration and proxy scripting, and the server plug-in API (Application Programming Interface).
- It enables you to set up *content filtering* by URL, and it provides access control by user, IP address, host name or domain, and web content.

What’s in This Book?

This book contains information about how the proxy server works and explains how to start, configure, and use it. This book will help you to maintain the server, understand its internal workings, and customize its functions. The book is divided into two parts. Part 1 discusses the administration of the proxy server and Part 2 explains how to program the server.

For information on how to install the proxy server, see the *Sun ONE Web Proxy Server 3.6 Installation Guide* for your platform.

For information on the administration server that comes with your proxy server, see *Managing Netscape Servers*.

Conventions Used in This Book

These conventions are used in this book:

`Monospaced font.` Monospaced type is used for text that you should type. It is also used for examples of code and for directories and filenames.

Italic. Italic text is used to introduce new terms and to represent variable information.

`|.` The vertical bar is used as a separator for user interface elements. For example, “choose Server Status|Log Preferences” means you click the Server Status button in the Server Manager and then click the Log Preferences link.

Contacting iPlanet Technical Support

For product-specific Technical Support assistance, please see the Product Support Page at:

`http://www.sun.com/service/sunone/software/index.html`

Further information can be found at the following Internet locations:

- Consulting Services

  `http://www.sun.com/service/sunps/sunone/index.html`

- Developer Information

  `http://developer.iplanet.com`

- Sofware Training

  `http://www.sun.com/software/training/`

- Software

  `http://www.sun.com/software/`

- Product Data Sheet

  `http://www.sun.com/software/products/web_proxy/ds_web_proxy.html`

Part 1

Administering the Proxy Server

Chapter 1

Starting the Administration and Proxy Servers

The proxy server's installation process installs two servers, an administration server and a proxy server. This chapter explains the different methods for starting and stopping both of these servers. For information on installing the proxy server, see the *Sun ONE Web Proxy Server 3.6 Installation Guide*.

Starting and Stopping the Administration Server

To start and configure your proxy server, you need to have an administration server running on your machine. For more information about the administration server, see *Managing Netscape Servers*.

Starting the Administration Server

The administration server starts automatically when you finish installing the proxy server. However, there may be instances when you need to stop and start it.

To start the administration server:

1. At the command prompt, change to the server root directory.
2. Type `./start-admin`.

Once you have started the administration server, you need to connect to it. Using a browser that supports frames and JavaScript, such as Netscape Navigator 4.0, enter the following URL for the administration server:

`http://`*servername.sub_domain*`.`*domain*`:`*port_number*

In the above URL, use the administration server's port number (not the port number for the proxy server) that you specified during installation. You will be prompted for a user name and password. Type the administration server user name and password you specified during the installation. The Server Administration page appears. For more information on the Server Administration page, see "Using the Server Administration Page" on page 22.

Stopping the Administration Server

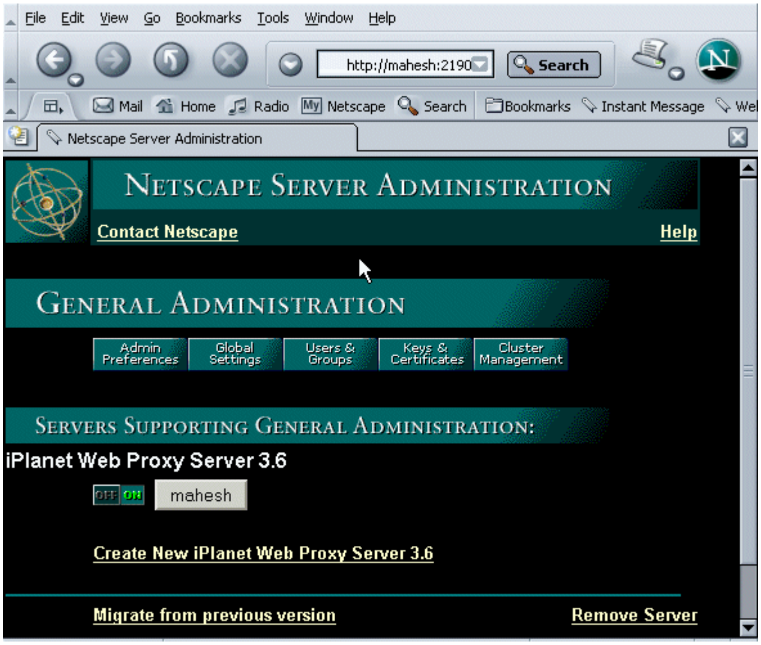
To stop the administration server:

1. At the command prompt, change to the server root directory.
2. Type `./stop-admin`.

Using the Server Administration Page

When you start the administration server, you see the Server Administration page screen, as shown in Figure 1-1.

Figure 1-1 Server Administration page


File Edit View Go Bookmarks Tools Window Help
http://mahesh:2190
Search
Mail Home Radio Netscape Search Bookmarks Instant Message We
Netscape Server Administration
NETSCAPE SERVER ADMINISTRATION
Contact Netscape
Help
GENERAL ADMINISTRATION
Admin Preferences
Global Settings
Users & Groups
Keys & Certificates
Cluster Management
SERVERS SUPPORTING GENERAL ADMINISTRATION:
iPlanet Web Proxy Server 3.6
OFF ON
mahesh
Create New iPlanet Web Proxy Server 3.6
Migrate from previous version
Remove Server

From the Server Administration page, you can perform the following tasks:

- Configure the administration server
- Choose a server to configure
- Start and stop a proxy server
- Create a new proxy server instance
- Migrate from an earlier version of the proxy server
- Remove a server

Starting and Stopping iPlanet Web Proxy Server

Once you have started the administration server, you can start your proxy server. There are several ways to start and stop the proxy server. The following sections discuss these methods.

Starting the Proxy Server

You can start the proxy server in one of the following ways:

Using the Server Administration Page

From the Server Administration page, you can start the proxy server by using one of the following options:

Option 1

Click the On / Off button next to the server you want to start.

Option 2

1. Click the name of the proxy server you want to start.
2. Choose System Settings | Start / Stop the server.
3. Click Start.

Manually

NOTE	If your proxy server is installed on a port number less than 1024, you need to be running as root to start the server manually.

To restart the proxy server from the command line:

1. If the proxy port number is less than 1024, log in as root or superuser. If the proxy port number is greater than 1024, log in with the proxy's user account.
2. At the command line prompt, go to the proxy's root directory and type the following:

   server root/`proxy-`*id*/`start`

   server root is the directory where you installed the server. The above start script has two command line arguments:

   - `-p XX` (where XX is a port number) starts the proxy on a specific port number. This overrides the setting in `magnus.conf`.
   - `-i` runs the proxy in inittab mode, so that if the proxy process is ever killed or crashes, inittab restarts the proxy server for you. It also prevents the proxy from putting itself in a background process.

3. Press Enter.

NOTE If the proxy is already running, you must stop it to use the start command. Also, if the proxy startup is unsuccessful, you should kill the process before trying to restart the proxy (see the troubleshooting section of the *Sun ONE Web Proxy Server 3.6 Installation Guide*).

Restarting the Proxy Server

Once installed, the proxy server and its child processes run constantly, waiting and handling requests. If your computer crashes or is taken offline, the server processes are lost. There are several ways you can restart the server:

- Automatically restart it from inittab.
- Automatically restart it when the machine reboots with the system RC scripts. In other words, use other daemons in `/etc/rc.local` for BSD-based systems, or `/ect/rc2.d/S99proxy` directory for SVR4 systems. Because the installation forms cannot edit the `/etc/rc.local`, `/etc/re2.d/S99proxy` or `/etc/inittab` files, you have to edit these files yourself.
- Manually soft start it.

Restarting with inittab

To restart the proxy server with inittab, put the following text on one line in the `/etc/inittab` file. The syntax is:

`proxy:2:respawn:`*server root*`/proxy-`*id*`/start -i`

server root is the directory where you installed the server. You'll need to remove this line before you stop the server (see the Troubleshooting section of the *Sun ONE Web Proxy Server 3.6 Installation Guide*).

NOTE If you are using a version of UNIX not derived from System V (such as SunOS 4.1.3), you won't be able to use the inittab option.

Restarting with the System RC Scripts

If you choose to use `/etc/rc.local`, or your system's equivalent, place the following line in that directory:

server root/`proxy-`*id*/`start`

server root is the directory where you installed the server.

Soft Starting the Proxy

If the proxy is running and you want to restart it so that it uses an updated configuration, type:

server root/`proxy-`*id*/`restart`

server root is the directory where you installed the server. This script finds the parent process ID (in the `logs/pid` file), and sends the hang-up (-HUP) signal with this process ID.

The Start-Up Process

The following events occur when you start your proxy server.

1. The administrator starts the proxy server.
2. The proxy reads the configuration files.
3. The proxy calls the init functions.
4. If caching is enabled for your proxy, the proxy calls the init-cache function. This function causes the proxy server to create an additional process, the cache monitor. This process will fork another process, the cache manager. Both processes have the name “ns-proxy.”
5. If the proxy server is running as root, the cache monitor and cache manager processes change their user id so that they don’t have root privileges.
6. The proxy server creates the listen socket to accept connections.
7. The proxy changes the uid in the parent process.
8. The proxy forks child processes. Because the server is not multithreaded inside processes, there are no worker threads.

Stopping the Proxy Server

You can stop a server in one of the following ways:

Using the Server Administration Page

From the Server Administration page screen, you can stop the proxy server by using one of the following options:

Option 1

OFF ON Click the On / Off button next to the server you want to stop.

Option 2

1. Click the name of the proxy server you want to stop.
2. Choose System Settings | Start / Stop the server.
3. Click Stop.

Manually

If you used inittab to start the server, you need to remove the line from `/etc/inittab` before you stop the server. Otherwise, the server restarts automatically after it is stopped.

To stop the server manually, log in as root or become superuser, or if you started the proxy using the proxy's user account, log in as that user. Type the following at the command-line prompt:

server root`/proxy-`*id*`/stop`

server root is the directory where you installed the server.

Creating a New Proxy Server Instance

From the Server Administration page, you can create a new instance of proxy server . To do so, complete the following steps:

1. Click Create New iPlanet Web Proxy Server 3.6 to launch the Web Proxy Server Installation page.
2. In the Web Proxy Server Installation page, type the following information for your proxy server:
   - Server Name: the host name where the proxy server is installed.
   - Bind Address: the IP address.
   - Server Port: the port that you want the proxy to listen to.
   - Server Identifier: a name used in the Server Selector to identify the specific proxy server.

NOTE The name you specify for the Server Identifier can contain only letters, digits, hyphens and underscores, and must begin with a letter.

- Server User: the current user account you specified during installation.
- Processes: the number of processes the proxy spawns. The default is 32.

In addition, specify the following information:

- Choose how you want the proxy server to resolve IP addresses. For more details, see the online help.
- Choose the log format you want the proxy to use. For more information, see Working with Log Files.
- Check the protocols you want the proxy to handle.
- Choose whether or not you want to cache documents and specify the caching-related configuration settings. For more details, see Chapter 9, “Caching”, and the online help.

Chapter 2

Managing Your Server

This chapter describes how to manage your iPlanet Web Proxy Server by using the Server Manager forms.

Once you have installed and started your administration and proxy servers, you can use the Server Manager forms to configure your proxy server. For information on installing and starting the administration server and proxy server, see the *Sun ONE Web Proxy Server 3.6 Installation Guide*.

Overview

You can configure the proxy server by using the web-based administration forms or by editing the configuration files.

The administration server runs a collection of web forms and CGI (Common Gateway Interface) scripts. The Server Administration page is the main web form that lets you configure the administration server or choose another server to configure. The Server Manager forms let you configure the server you select on the Server Administration page.

Using the Server Manager

The Server Manager is a collection of forms that lets you configure and administer your proxy server. To access the Server Manager, you choose the server you want to configure from the Server Administration page. For information on accessing the Server Administration page, see the *Installation Guide*. The Server Manager is shown in Figure 2-1. You can use the Server Manager from any remote computer as long as it has permission to access the administration server.

To access the Server Administration page and use the Server Manager:

1. Using a browser that supports frames and JavaScript, such as Netscape Navigator 4.0 or later, enter the URL for the administration server. The URL has the following format:

   http://*servername.domain.domain:port_number*/

   For example, http://atomic.acmecorp.com:1357

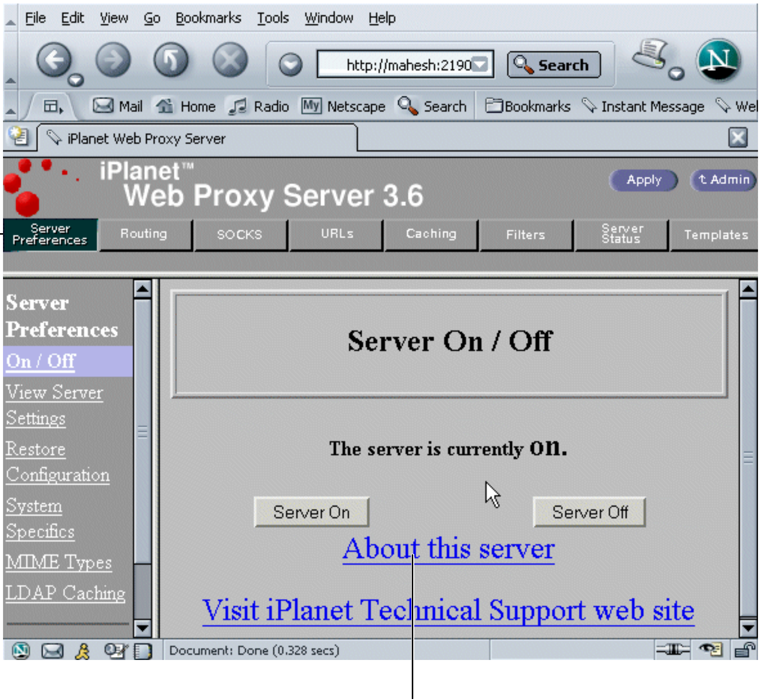
   Use the port number for the administration server that you specified during installation; this is not the port number for the proxy server.

NOTE If you are already on the Server Administration page, skip directly to step 3.

2. You'll be prompted for a user name and password. Type the administration server user name and password that you specified during installation. The Server Administration page appears.

3. Click the button containing the name of the proxy server you want to configure. The Server Manager appears, as shown in Figure 2-1.

Figure 2-1 The Proxy's Server Manager main forms



Server On / Off

The server is currently on.

Click a button category to view its list of links.

Click a link to view the form that contains its options.

Forms appear here with buttons and options you use to configure the proxy server. After applying the form, you'll get a confirmation message.

4. To configure specific aspects of your iPlanet Web Proxy Server, click a button at the top of the form, and then choose a link in the left frame. The form appears in the frame on the right.

NOTE You must save and apply your changes in order for the proxy server to begin using them. After you submit certain forms, you'll see a form that allows you to save and apply your changes. Choosing the Save option does not restart your proxy server, however, choosing Save and Apply does restart the server.

You can return to the Server Administration page by clicking the Admin button in the upper-right corner of the Server Manager.

Chapter 3

Managing Templates and Resources

Templates allow you to group URLs together so that you can configure how the proxy handles them. You can make the proxy behave differently depending on the URL the client tries to retrieve. For example, you might require the client to authenticate (type in a user name and password) when accessing URLs from a specific domain. Or, you might deny access to URLs that point to image files. You can configure different cache refresh settings based on the file type (keep some files in the cache longer than others).

What is a Template?

A *template* is a collection of URLs, called *resources*. A resource might be a single URL, a group of URLs that have something in common, or an entire protocol. You name and create a template and then you assign URLs to that template by using regular expressions. This means that you can configure the proxy server to handle requests for various URLs differently. Any URL pattern you can create with regular expressions can be included in a template. Table 3-1 lists the default resources and provides some ideas for other templates.

Table 3-1 Resource regular expression wildcard patterns

Regular expression pattern	What it configures
ftp://.*	All FTP requests
http://.*	All HTTP requests
https://.*	All secure HTTP requests
gopher://.*	All Gopher requests
connect://.*:443	All SSL (secure) transactions to HTTPS port.
http://home\.iplanet\.com.*	All documents on the home.iplanet.com web site.

Table 3-1 Resource regular expression wildcard patterns

Regular expression pattern	What it configures
.*\.gif.*	Any URL that includes the string `.gif`
.*\.edu.*	Any URL that includes the string `.edu`
http://.*\.edu.*	Any URL going to a computer in the .edu domain

Understanding Regular Expressions

iPlanet Web Proxy Server allows you to use regular expressions to identify resources. Regular expressions specify a pattern of character strings. In the proxy server, regular expressions are used to find matching patterns in URLs.

Here is an example of a regular expression:

```
[a-z]*://[^:/]*\.abc\.com.*>
```

This regular expression would match any documents from the .abc.com domain. The documents could be of any protocol and could have any file extension.

Table 3-2 contains regular expressions and their corresponding meanings.

Table 3-2 Regular expressions and their meanings

Expression	Meaning
.	Matches any single character except a newline.
_x_?	Matches zero or one occurrences of regular expression _x_.
_x_*	Matches zero or more occurrences of regular expression _x_.
_x_+	Matches one or more occurrences of regular expression _x_.
_x_{_n_,_m_}	Matches the character _x_ where _x_ occurs at least _n_ times but no more than _m_ times.
_x_{_n_,}	Matches the character _x_ where _x_ occurs at least _n_ times.
_x_{_n_}	Matches the character _x_ where _x_ occurs exactly _n_ times.
[_abc_]	Matches any of the characters enclosed in the brackets.
[^_abc_]	Matches any character not enclosed in the brackets.
[_a_-_z_]	Matches any characters within the range in the brackets.
_x_	Matches the character _x_ where _x_ is not a special character.
_x_	Removes the meaning of special character _x_.

Table 3-2 Regular expressions and their meanings

Expression	Meaning
"*x*"	Removes the meaning of special character *x*.
xy	Matches the occurrence of regular expression *x* followed by the occurrence of regular expression *y*.
x|*y*	Matches either the regular expression *x* or the regular expression *y*.
^	Matches the beginning of a string.
$	Matches the end of a string.
(*x*)	Groups regular expressions.

This example illustrates how you can use some of the regular expressions in Table 3-2.

```
[a-z]*://([^.:/]*[:/]|.*\.local\.com).*"
```

- [a-z]* matches a document of any protocol.
- :// matches a (:) followed by (//).
- [^.:/]*[:/] matches any character string that does not include a (.),(:) or (/), and is followed by either a (:) or a (/). It therefore matches host names that are not fully qualified and hosts with port numbers.
- |.*\.local\.com does not match fully qualified domain name host names such as local.com but does match documents in the .local.com domain.
- .*" matches documents with any file extension.

NOTE As noted in Table 3-2, the backslash can be used to escape or remove the meaning of special characters. Characters such as the period and question mark have special meanings, and therefore, must be escaped if they are used to represent themselves. The period, in particular, is found in many URLs. So, to remove the special meaning of the period in your regular expression, you need to precede it with a backslash.

Understanding Wildcard Patterns

You can create lists of *wildcard* patterns that enable you to specify which URLs can be accessed from your site. Wildcards can be in the form of regular expressions or shell expressions, depending on usage. As a general rule:

- Use regular expressions for any pattern that matches destination URLs. This includes <Object ppath=...>, URL filters, and the NameTrans, PathCheck, and ObjectType functions.
- Use shell expressions for any pattern that matches incoming client or user IDs, including user names and groups for access control and the IP addresses or DNS names of incoming users (for example, <Client dns=...>).

You can specify several URLs by using regular expression wildcard patterns. Wildcards let you filter by domain name or by any URL with a given word in the URL. For example, you might want to block access to URLs that contain the string “sex.” To do this, you could specify http://.*sex.* as the regular expression for the template.

Creating Templates

You can create a template using a regular expression wildcard pattern. You can then configure aspects that affect only the URLs specified in that template. For example, you might use one type of caching configuration for `.GIF` images and another for plain `.HTML` files.

To create a template:

1. From the Server Manager, choose Templates | New Templates.

   The Create a New Template form appears.

2. In the Template Name field, type a name for the template you’re creating, and click OK.

   The name should be something you can easily remember. The Server Manager prompts you to save and apply your changes. You can save the changes after you create a regular expression for the template, as described in the remaining steps.

3. Click Templates | Apply Template.

   The Apply a Configuration Template form appears.

4. Type a regular expression wildcard pattern that includes all of the URLs you want to include in your template.
5. From the list, select the name of the new template you just added.
6. Click OK.

Viewing and Removing Templates

You can view the templates created in the Server Manager. To do this, choose Templates|View Template. The templates are shown in a table that lists the regular expression for the template and the template name. To edit an existing template, click the Edit link, which takes you to the Apply form.

You can also remove existing templates. Removing a template deletes all of the associated configurations for the template. For example, if you have access control set up for all URLs in the template TEST, removing the TEST template also removes the access control to the URLs contained in then template.

To remove a template,

1. From the Server Manager, choose Templates|Remove Templates.
2. Choose the template from the Remove list.
3. Click OK.

Removing Resources

You can delete an entire regular expression object and its corresponding configurations with the Remove an Existing Resource form. For instance, you can remove the gopher resource so that all settings associated with that resource will be removed from the proxy server’s configuration files.

To remove a resource,

1. From the Server Manager, choose Templates|Remove Resource. The Remove an Existing Resource form appears.
2. Select the resource you want to remove by either choosing it from the Remove pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. Click OK.

Online Forms for Controlling Resources

This section briefly lists the features that use templates. The features are listed along with information on how to access the Server Manager forms and where to find descriptions of the features:

- Accessing a resource (Server Preferences | Restrict Access). See “Restricting Access” on page 57.
- Accessing specific URLs (Filters | URL Filters). See “Restricting Access” on page 57.
- Caching (Caching | Configuration). See “Configuring the Cache” on page 115.
- Proxying (Routing | Enable, Disable). See “Enabling Proxying for a Resource” on page 63.
- Routing (Routing | Routing). See “Configuring Routing for a Resource” on page 64.
- Setting logging preferences (Status | Log Preferences). See “Setting Access Log Preferences” on page 190.
- Mapping URLs to mirror sites (URLs | Create Mappings). See “Mapping URLs to Other URLs” on page 71.

Chapter 4

Configuring Server Preferences

This chapter describes the proxy server’s system settings and tells you how to configure them. System settings affect the entire proxy server. They include options such as the user account the proxy server uses and the port to which it listens.

For directions on starting and stopping the server, see “Starting and Stopping iPlanet Web Proxy Server” on page 21.

Starting and Stopping the Proxy Server

There are several methods by which you can start and stop your proxy server. One of these methods is to use the Server On / Off form in the Server Manager. Other methods for starting and stopping your proxy server are discussed in Chapter 1, “Starting the Administration and Proxy Servers.”

To use the Server On / Off form to start or stop the proxy server,

1. From the Server Manager, choose Server Preferences | On / Off.
2. Click the Server On or Server Off button.

Viewing Server Settings

During installation, you configure some settings for your proxy server. You can view these and other system settings from the Server Manager. The View Server Settings form lists all of the settings for your proxy server. This form also tells you if you have unsaved and unapplied changes, in which case you should save the changes and restart the proxy server so it can begin using the new configurations.

There are two types of settings, technical and content. The proxy server's technical settings come from the `magnus.conf` file, and the content settings come from the `obj.conf` file. These files are located in the server root directory in the subdirectory called `admin-serv/proxy-`*id*. For more information about the `magnus.conf` file and `obj.conf` files, see Appendix C, "Proxy Configuration Files."

To view the settings for your server, in the Server Manager, choose Server Preferences | View Server Settings. This list explains the server's technical settings:

- Server Root is the directory where the server binaries are kept. You first specified this directory during installation.
- Hostname is the URL clients will use to access your server.
- Port is the port on your system to which the server listens for HTTP requests.
- Error log is the name and path of the server's error log file.
- User is the user the server runs as.
- Processes is the number of processes your server uses when it starts.
- DNS shows whether DNS is enabled or disabled.

The server's content settings depend on how you've configured your server. Typically, the proxy lists all templates, URL mappings, and access control. For individual templates, this form lists the template name, its regular expression, and the settings for the template (such as cache settings).

Restoring and Viewing Backup Configuration Files

You can view or restore a backup copy of your configuration files (`magnus.conf`, `obj.conf`, `bu.conf`, `mime.types`, and `genwork.proxy-`*id*`.acl`). This feature lets you go to a previous configuration if you're having trouble with your current configuration. For example, if you make lots of changes to the proxy's configuration and then the proxy doesn't work the way you thought it should (for example, you denied access to a URL but the proxy will service the request), you can revert to a previous configuration and then redo your configuration changes.

To view a previous configuration:

1. From the Server Manager, choose Server Preferences | Restore Configuration. The Restore Configuration form appears. The form lists all of the previous configurations ordered by date and time.
2. Click the View button for the version you want to display. A listing of the technical and content settings in that configuration appears.

To restore a backup copy of your configuration files:

1. From the Server Manager, choose Server Preferences | Restore Configuration.
2. Click Restore for the version you want to restore.

   If you want to restore all files to their state at a particular time, click the Restore to *time* button on the left-most column of the table (*time* being the date and time to which you want to restore).

You can also set the number of backups displayed on the Restore Configuration form. To set the number of backups displayed:

1. In the Server Manager, choose Server Preferences | Restore Configuration.
2. In the “Set number of sets of backups” field, enter the number of backups you want to display.
3. Click the Change button.

Changing System Specifics

The System Specifics form lets you set up or change the basic aspects of your server. The form allows you to change the server port, server user, authentication password, and proxy timeout for your proxy server. It also allows you to enable DNS, ICP and proxy arrays. And for the UNIX server, it shows the number of processes or process life. You can also enable or disable DNS from the System Specifics form.

To change the system specifics options:

1. In the Server Manager, choose System Settings | System Specifics.

   The System Specifics form appears.
2. Change the options as needed, and then click OK.

   The options are described in the following sections.

Make sure you save and apply the changes.

Bind Address

Bind address is the IP address to which this instance of iPlanet Web Proxy Server should listen. You only need to specify a bind address if your machine is answering multiple IP addresses.

Server Port

The *server port* specifies the number of the TCP port to which the proxy listens. The number you choose is used by proxy users when configuring their web browsers to use the proxy server. Users must specify this server name and port number to get access through the proxy server.

Port numbers for all network-accessible services are maintained in the `/etc/services` file and yp services on UNIX machines. The standard Telnet port number is 23, and the standard HTTP port number is 80. Because the proxy is not a regular HTTP server, you shouldn't use port 80. Proxies haven't been assigned an official, industry-standard port number.

A recommended proxy port number is 8080. When configuring client programs to use this proxy server, you have to tell them both the host name and the port number. For example, you would use this line in the proxy preferences dialog box in Netscape Navigator:

```
proxy.netscape.com 8080
```

NOTE If you use proxy's SOCKS daemon feature, the proxy should listen to the standard SOCKS port (1080).

If you aren't sure if the port number you plan to use is available, check in the `/etc/services` file on the server machine. Technically, the proxy port number can be any port from 1 to 65535. On a UNIX machine, if you aren't running as root or superuser when you install or start the proxy, you'll have to use a number greater than 1024.

Server User

The *server user* is the user account that the proxy uses. The user name you enter as the proxy server user should already exist as a normal user account. When the server starts, it runs as if it were started by this user.

If you want to avoid creating a new user account, you can choose an account used by another HTTP server running on the same host, or if you are running a UNIX proxy, you can choose the user nobody. However, on some systems the user *nobody* can own files but can't run programs, which would make it unsuitable as the proxy user name.

On a UNIX machine, all the processes that the proxy spawns are assigned to the server user account.

Instructions for creating a new user on your UNIX system can be found in your system manual or a UNIX administrator's handbook.

Processes

The *processes* field shows how many background processes are available to service requests. When individual users send requests to the proxy server, the proxy uses background processes to service their requests. You can specify the number of processes dedicated to the proxy. These processes are spawned when the server starts and they remain idle until needed. Base your choice on achieving a balance between system load and server requests:

The process table of the proxy's system limits the number of processes the proxy can use.

- On a high-demand system, with more than a dozen users, the server requires many of these processes, for example, 80 processes, to handle many simultaneous requests.
- On a low-demand system with less than a dozen users, where only a few simultaneous connections are active at a time, 20 to 40 processes should be sufficient.

NOTE Depending upon the platform, each process uses the following amount of RAM when idle:

- AIX: 2.5 Mb
- HP-UX: 3.3 Mb
- Solaris: 5 Mb

The amount of RAM used by each process can increase by 10% when the process is active. If you specify more processes than can fit simultaneously in main memory, the system starts swapping in virtual memory, which slows down proxy service. All proxy processes must fit in main memory simultaneously to make the proxy efficient.

Table 4-1 on page 44 lists suggested numbers of processes. Use this table to determine the number of processes for your proxy server. You will have to use the extended or extended-2 access log file format to capture the data you'll need to use this table. Before you can use Table 4-1 you must know how long requests take and how many requests the proxy receives per second.

- To find the average service time per request, look at the access log file.
- To estimate the average number of new requests per second, view the access log during peak hours. Use tail -f to continuously view the access log file as the proxy adds entries to it. As entries are added, base your estimate on the number of users and how active they are.

NOTE The operating system on which you are running your proxy server may limit the number of processes per user. If you need more processes for your proxy server, change the process settings for your operating system.

You can change the number of processes at any time using the online form (System Settings | System Specifics), or you can change the number in the `magnus.conf` file manually (see "MaxProcs" on page 407).

If the server seems slow or is not responding, especially during peak hours, you should increase the number of processes available to the proxy. You might have to increase the RAM or the size of the operating system’s process table before you increase the number or processes. For details on changing the operating system’s RAM or process table, see the system administration documentation provided with your system.

Table 4-1 Suggested number of processes based on average request service time and number of requests

Average number of new requests per second	Average number of seconds of service time per request															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	10	10	10	15	15	20	20	20	25	25	30	30	30	35	35	40
2	10	15	15	20	25	25	30	30	35	40	40	45	45	50	55	55
3	15	20	20	25	30	35	40	40	45	50	55	60	60	65	70	75
4	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
5	20	25	30	40	45	50	55	60	70	75	80	85	90	100	105	110
6	25	30	35	45	50	60	65	70	80	85	95	100	105	115	120	130
7	25	35	40	50	60	65	75	80	90	100	105	115	120	130	140	145
8	30	40	45	55	65	75	85	90	100	110	120	130	135	145	155	165
9	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180
10	35	45	55	70	80	90	100	110	125	135	145	155	165	180	190	200
12	40	55	65	80	95	105	120	130	145	160	170	185	195	210	225	235
14	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270
16	55	70	85	105	120	140	155	170	190	205	225	240	255	275	290	310
18	60	80	95	115	135	155	175	190	210	230	250	270	285	305	325	345
20	65	85	105	130	150	170	190	210	235	255	275	295	315	340	360	380
22	70	95	115	140	165	185	210	230	255	280	300	325	345	370	395	415
24	75	100	125	150	175	200	225	250	275	300	325	350	375	400	425	450
26	85	110	135	165	190	220	245	270	300	325	355	380	405	435	460	490
28	90	120	145	175	205	235	265	290	320	350	380	410	435	465	495	525
30	95	125	155	190	220	250	280	310	345	375	405	435	465	500	530	560
35	110	145	180	220	255	290	325	360	400	435	470	505	540	580		
40	125	165	205	250	290	330	370	410	455	495	535	575				
45	140	185	230	280	325	370	415	460	510	555	600					
50	155	205	255	310	360	410	460	510	565							
55	170	225	280	340	395	450	505	560								
60	185	245	305	370	430	490	550									
65	200	265	330	400	465	530	595									
70	215	285	355	430	500	570										

Process Life

The *process life* is the number of requests that each server child process services before it exits and gets respawned by the master process. The process life allows memory fragmentation to be cleaned.

DNS

A *Domain Name Service* (DNS) restores IP addresses into host names. When a web browser connects to your server, the server gets only the client's IP address, for example, 198.95.251.30. The server does not have the host name information, such as www1.netscape.com. For access logging and access control, the server can resolve the IP address into a host name. On the System Specifics form, you can tell the server whether or not to resolve IP addresses into host names.

ICP

The *Internet Cache Protocol* (ICP) is a message-passing protocol that enables caches to communicate with one another. Caches can use ICP to send queries and replies about the existence of cached URLs and about the best locations from which to retrieve those URLs. You can enable ICP on the System Specifics form. For more information on ICP, see "Routing Through ICP Neighborhoods" on page 141.

Proxy Array

A *proxy array* is an array of proxies serving as one cache for the purposes of distributed caching. If you enable the proxy array option on the System Specifics form, that means that the proxy server you are configuring is a member of a proxy array, and that all other members in the array are its siblings. For more information on using proxy arrays, see "Routing through Proxy Arrays" on page 130.

Parent Array

A *parent array* is a proxy array that a proxy or proxy array routes through. So, if a proxy routes through an upstream proxy array before accessing a remote server, the upstream proxy array is considered the parent array. For more information on using parent arrays with your proxy server, see "Routing Through a Parent Array" on page 140.

Proxy Timeout

The *proxy timeout* is the maximum time between successive network data packets from the remote server before the proxy server times out the request. The default value for proxy timeout is 5 minutes.

NOTE When the remote server uses server-push and the delay between pages is longer than the proxy timeout, the connection could be terminated before the transmission is done. Instead, use client-pull, which sends multiple requests to the proxy.

Creating MIME Types

A MIME (Multi-Purpose Internet Mail Extension) type is a standard for multimedia e-mail and messaging. So that you can filter files depending on their MIME type, the proxy server provides a form that lets you create new MIME types for use with your server. The proxy adds the new types to the `mime.types` file (described on page 261). See "Filtering by MIME Type" on page 156 for more information on blocking files based on MIME types.

To add a MIME type:

1. In the Server Manager, choose System Settings | MIME Types.
2. The form that appears shows all the MIME types listed in the proxy's mime.types file.
   - You can edit any MIME type by clicking the link for any part of the MIME type.
   - To create a new MIME type, click the New Type button at the bottom of the form.
3. The form that appears is blank if you're creating a new type, or it displays the MIME type you want to edit. The fields on this form are:
   - Type is the category of MIME type. This can be type, enc, or lang, where type is the file or application type, enc is the encoding used for compression, and lang is the language encoding.

- MIME Type defines the content type that appears in the HTTP header. The receiving client (such as Netscape Navigator) uses the header string to determine how to handle the file (for example, by starting a separate application or using a plug-in application). The standard strings are listed in RFC 1521.
- File Suffix refers to the file extensions that map to the MIME type. To specify more than one extension, separate the entries with a comma. The file extensions should be unique. That is, you shouldn't map one file extension to two MIME types.

4. Click OK to submit the form. Save and apply your changes.

Understanding DNS Caching

iPlanet Web Proxy Server supports DNS caching to reduce the number of DNS lookups performed by the proxy while it resolves DNS host names into IP addresses.

How DNS Caching Works

The DNS caching feature uses a memory-mapped, shared file to store cached DNS data for all proxy server child processes. By default, this file is an invisible file called `/tmp/dnscache.8080`. An invisible file is one that remains open but does not appear in the file system's directory structure. You can make the DNS cache file visible by choosing System Settings|Tuning from the Server Manager and selecting the On radio button next to the words, "DNS cache file visible".

This shared memory area is protected by a number of semaphores, named `/tmp/dnssema.8080.`*n*, where the value of *n* can be 1 through the total number of semaphores set in the DNS Cache Configuration page. Each semaphore protects a portion of the shared memory file. By having several semaphores, you avoid potential semaphore congestion, and multiple processes can simultaneously access the shared memory DNS cache (although they access different parts).

Configuring the DNS Cache

From the DNS Cache Configuration page you can specify:

- DNS cache directory
- size of the DNS cache
- number of semaphores to protect the shared memory file
- expiration of DNS cache entries

DNS Cache Directory Location

The DNS cache directory is `/tmp` by default. You may set the DNS cache directory to any directory that is writable by the proxy process and has enough disk space to hold the DNS cache file.

DNS Cache Size Setting

The size of the DNS cache is expressed in kilobytes. By default, the size is set to 512 kilobytes (.5 MB).

DNS Cache Semaphores

The number of semaphores you need depends on how many processes there are in the server child process pool (the MaxProcs setting). If the proxy handles only a light load, a single or a few semaphores is sufficient. However, if the load is substantial or heavy, and MaxProcs is high (say over a hundred processes), there should be more semaphores to allow more processes to access the DNS cache simultaneously.

The default value is 4 semaphores, which means that at most four processes can simultaneously look up or store data to or from the DNS cache. Unless performance seems to improve by increasing this number, four is a good default value. Having too many semaphores can also hurt the performance.

DNS Cache Entry Expiration

The proxy server purges DNS cache entries from the cache when it reaches a pre-set expiration time. Because the standard gethostbyname() interface to the system resolves the host names, the explicit expiration information provided by the DNS is not available to the proxy's DNS cache.

By default, the DNS expiration time is 1 hour (3600 seconds).

Setting Levels of DNS Subdomains

Some URLs contain host names with many levels of subdomains. It can take the proxy server a long time to do DNS checks if the first DNS server can't resolve the host name. You can set the number of levels that the proxy server will check before returning a "host not found" message to the client.

For example, if the client requests http://www.sj.ca.netscape.com/index.html, it could take a long time for the proxy to resolve that host into an IP address because it might have to go through 4 DNS servers to get the IP address for the host computer. Because these lookups can take a lot of time, you can configure the proxy server to quit looking up an IP address if the proxy has to use more than a certain number of DNS servers.

To set the levels of subdomains the proxy traverses,

1. In the Server Manager, choose System Settings | DNS Subdomains.
2. Choose the template you want to use or choose the entire server.
3. Select the number of levels from the drop-down list.
4. Click OK.Be sure to save and apply your changes.

Disabling HTTP Keep-Alive

The proxy supports HTTP keep-alive packets. The keep-alive sub-system is enabled by default in the version 3.6 Service Pack 3 release of the Sun ONE Web Proxy Server. Keep-alives are a TCP/IP feature that keeps a connection open after the request is complete, so that the client can quickly reuse the open connection.

In normal client-server transactions on the web, the client can make several connections to the server that requests multiple documents. For example, if the client requests a web page that has several graphic images, the client needs to make separate requests for each graphic file. Reestablishing connections is time consuming.

By default, keep-alives are enabled on your proxy.

To disable keep-alives, complete the following steps:

1. In the Server Manager, choose System Settings | HTTP Keep-Alive.
2. Choose the template you want to use or choose the entire server.
3. Check Off, and then click OK. Be sure to save and apply your changes.

Allowing or Blocking Arbitrary Methods

The proxy can be configured to allow or block arbitrary methods by editing the `obj.conf` file found in `<`*`server-root`*`>/config` directory. See “proxy-retrieve (retrieving documents with the proxy)” on page 467 for more information.

WebDAV Support

The proxy provides support for the Web Distributed Authoring and Versioning protocol. The methods supported are:

- PROPFIND
- PROPPATCH
- MKCOL
- COPY
- LOCK
- UNLOCK
- DELETE

You can however configure the proxy server to support additional arbitrary WebDAV methods. See “proxy-retrieve (retrieving documents with the proxy)” on page 467 for more information.

The response returned by WebDAV requests is the 207 Multi-Status response.

Chapter 5

Controlling Access to Your Server

You can restrict access to all of the data served by the proxy server or to the specific URLs it serves. You can specify that only certain people access specific URLs or that everyone except those people can see the files. This access restriction applies only to URLs that your proxy server can send to a client and does not have anything to do with allowing people to administer or configure your server.

For example, you might allow all clients to access URLs for HTTP but then allow only restricted access to FTP. You could also restrict URLs based on host names or domain names, such as if you have a proxy serving many internal web servers but want only specific people to access a confidential research project stored on one of the web servers.

If your server has SSL (Secure Sockets Layer) enabled, the user's name and password are sent encrypted. Otherwise, names and passwords are sent in clear text, and can be read if intercepted.

If you want to control who can configure the proxy server itself and who can access the server configuration files, see *Managing Netscape Servers*.

When configuring access control for your server, you usually follow this process:

1. Choose an LDAP directory server or a local database.
2. Enter one or more users into the directory or database.
3. Create a resource by choosing the URLs you want to restrict (discussed on page 57).
4. Specify the default access (everyone allowed or everyone denied) for that resource (discussed on page 59).
5. Specify which users are exceptions to the default access (discussed on page 59).

For more information on databases, users, and groups, see *Managing Netscape Servers*.

How Does Access Control Work?

You can control access to the entire server or to parts of the server (that is, directories, files, file types). When the server evaluates an incoming request, it determines access based on a hierarchy of rules called access-control entries (ACEs), and then it uses the matching entries to determine if the request is allowed or denied. Each ACE specifies whether or not the server should continue to the next ACE in the hierarchy. The collection of ACEs is called an access-control list (ACL). When a request comes in to the server, the server looks in `obj.conf` for a reference to an ACL, which is then used to determine access. By default, the server has one ACL file that contains multiple ACLs.

Access Control Files

When you use access control on your proxy server, the settings are stored in a file with the extension `.acl`. These files are known as access control files, or ACL files. An ACL file is a text file containing access control lists which can be used to control access to server resources. Each access control list controls a set of access rights and specifies the clients that have these rights. Clients can be specified by their IP address, DNS name, user name, group name, or combinations of these attributes.

ACL files also contain information about how to authenticate users, such as what user database to use and what authentication method to use. ACL files do not contain any information about the server resources to which they are applied. ACLs are bound to server resources by directives in the server's `obj.conf` file, which refer to ACLs defined in the ACL file.

Access control files for the proxy server are stored in the directory *server_root*/`httpacl`. The main ACL file name is `generated.proxy-`*id*`.acl`; the temporary working file is called `genwork.proxy-`*id*`.acl`**.** If you use the Server Manager forms to restrict access, you'll have these two files. However, if you want to do more complex restrictions, you can create multiple files and reference them from the `magnus.conf` file.

You also need to know the syntax and function of ACL files if you plan to customize access control using the access-control API. See"ACL File Syntax" for more information on ACL file syntax.

ACL File Syntax

All ACL files must follow a specific format and syntax. Some general rules of ACLs are:

- Spaces, tabs, and newline characters generally are not significant except to create whitespace.
- Comments begin with a # and end with a newline, and can be placed anywhere.
- Identifiers, including ACL names, access right names, and user and group names, can contain letters, digits, hyphens, and underscores, but must begin with a letter.
- With the exception of user, group, and database names, case is generally not significant in identifiers or keywords.

An ACL file contains a sequence of ACL definitions. Each of these specifies an ACL name, a set of access rights to be controlled by the ACL, and a list of ACL directives. The following is the syntax of an ACL.

```
ACL acl-name acl-rights  {
    acl-directives
    }
```

Where:

acl-name is a unique name for the ACL. Typically, this name is generated by the Server Manager forms.

acl-rights are a list of access right names separated by commas and enclosed in parentheses. Access right names are specific to a particular type of server. Proxy servers use HTTP and FTP method names as access right names, including GET, HEAD, POST, and PUT.

acl-directives are a list of ACL directives separated by semicolons. There are two basic kinds of directives, a realm-directive and an access-directive. All directives begin with a force-keyword.

- realm-directive

The syntax of a realm-directive is:

force-keyword `Authenticate In` *realm-definition*

force-keyword is a keyword that has one of the following values:

- *Default* means that the effect of the directives is not immediate, and that the effect may be modified or even nullified by subsequent directives.
- *Always* means that the directive should take action immediately; therefore, terminating any further ACL evaluation.

realm definition is a string that gets displayed to the user. It has the form:

```
{
    Database db-name;
    Method auth-method-name;
}
```

db-name is the name of an authentication database associated with a realm. The default for db-name uses the SuiteSpot LDAP settings.

auth-method-name is the name of an authentication method supported by the server (currently basic or SSL).

- access-directive

An access-directive begins with a force-keyword, followed by either Allow or Deny. The syntax of an access-directive is:

force-keyword `Allow|Deny` *authorization-list*

force keyword is a keyword that has one of the following values:

- *Default* means that the effect of the directives is not immediate, and that the effect may be modified or even nullified by subsequent directives.
- *Always* means that the directive should take action immediately; therefore, terminating any further ACL evaluation.

authorization-list is the list of users and hosts to which the access-directive applies. It is actually a list of authorization-spec constructs, separated by commas.

Each authorization-spec must contain a user-list, and may contain a host-list. The form of an authorization-spec is:

user-list

or

user-list `At` *host-list*

user-list can be a user name, a group name, or a list of user and/or group names separated by commas and enclosed in parentheses. It can also be one of the special keywords, *anyone* or *all*. The keyword *anyone* indicates that the user's identity is not relevant to applying this directive. The keyword *all* indicates that any authenticated user in the current realm is matched by the directive.

host-list can be an ip-spec, a dns-spec, or a list of these separated by commas and enclosed in parentheses. An ip-spec specifies an IP host or network address, and consists of an IP address in dotted numeric notation, optionally followed by an IP netmask in dotted numeric notation. A dns-spec can be a fully-qualified domain name, or a partially-qualified domain name that begins with *. An dns-spec could be: "doon.mcom.com", "*.mcom.com", or "*".

Example 1

```
ACL readers (GET, HEAD) {
   Default deny anyone at *;
   Default allow anyone at *.mcom.com;
   }
```

In the above example, the name of the ACL is readers and the access rights it controls are the HTTP methods, GET and HEAD. Within the ACL are ACL directives which define the users who are denied and allowed GET and HEAD access. Each of the directives in this example begin with the word Default, which indicates that, by default, the directive applies to any client matching the criteria established by the directive. However, if the client also matches the criteria of a subsequent directive, then the that directive will override the previous directive.

The first directive in the previous example denies GET and HEAD access to any client matching its criteria. The criteria are the user name, anyone, at any host with a DNS name matching the pattern, "*". The user name, anyone, is a special name which places no requirements at all on the user identity of the client. It means that the server does not need to know the identity of the client user, so it will match unauthenticated clients. The DNS name pattern, "*", matches any client DNS name, so the net effect of this directive is that it will match any client, with or without authentication, from any host name.

By itself, the first directive denies access to anyone and everyone. However, the second directive allows access to a more selective set of clients, that is, clients with host names matching the pattern, *.mcom.com. The second directive also identifies anyone as the allowed user, indicating that the user name of the client is not relevant.

In summary, what this example does is restrict client access based only on the client host DNS name, denying access to all client hosts except those with DNS names ending with .mcom.com. We could have also identified the client hosts using IP addresses, specifying a netmask to indicate which bits of the IP address are required to match:

Example 2

```
ACL readers (GET, HEAD) {
   Default deny anyone at 0.0.0.0 0.0.0.0;
   Default allow anyone at
   (
      198.93.92.0 255.255.255.0, 198.93.93.0 255.255.255.0,
      198.93.94.0 255.255.255.0, 198.93.95.0 255.255.255.0,
      198.95.249.0 255.255.255.0, 198.95.250.0 255.255.255.0,
      205.217.226.0 255.255.255.0, 205.217.228.0 255.255.255.0,
      205.217.229.0 255.255.255.0, 205.217.230.0 255.255.255.0,
      205.217.231.0 255.255.255.0, 205.217.232.0 255.255.255.0,
        205.217.233.0 255.255.255.0, 205.217.234.0 255.255.255.0,
        205.217.235.0 255.255.255.0, 205.217.236.0 255.255.255.0,
        205.217.237.0 255.255.255.0, 205.217.238.0 255.255.255.0,
        205.217.239.0 255.255.255.0, 205.217.240.0 255.255.255.0,
        205.217.241.0 255.255.255.0, 205.217.242.0 255.255.255.0,
        205.217.243.0 255.255.255.0, 205.217.244.0 255.255.255.0,
        205.217.252.0 255.255.255.0, 205.217.254.0 255.255.255.0,
        205.217.255.0 255.255.255.0
        );
    }
```

Here 0.0.0.0 0.0.0.0 specifies an IP address and a netmask. With no bits set in the netmask, this specification will match any IP address, and therefore the first directive will have the effect of denying access to all clients.

The second directive in the above example allows access to any of the client hosts in the specified ranges. Notice that the list of IP address and netmask pairs is specified in parentheses. Because many different ranges of IP addresses are in use in the domain mcom.com, a list of IP address and netmask pairs must be given in order to identify the client hosts in this domain.

Example 3

```
ACL readers (GET, HEAD) {
    Always allow anyone at webmaster.enterprise.com;
    Default authenticate in {
    Database enterprise.com;
    Method SSL;
    };
    Default deny anyone at *;
    Default allow all at *.enterprise.com;
    Default deny contractors at *.enterprise.com;
    }
```

ACL directives are evaluated in the order in which they appear in an ACL definition. The word “Default” at the beginning of an ACL directive indicates that the effect of the directives is not immediate, and that the effect may be modified or even nullified by subsequent directives. In some cases, however, it may be desirable to have a directive which takes effect immediately. As shown in the previous example, replacing the word “Default” with the word “Always” makes the directive immediately effective.

This example immediately allows access to any user connecting from the host, webmaster.enterprise.com, without requiring authentication. When the client host is webmaster.enterprise.com, the directives following the first one are not evaluated.

Controlling Access with Client Certificates

If you have enabled SSL on your server, you can use client certificates in conjunction with access control. To do this, you must specify that a resource requires a client certificate to access it. When this feature is enabled on your server, users with certificate types their login name and password only the first time they attempt to access a restricted resource. Once their identity is established, the server maps their login name and password to that specific certificate. From then on, users no longer need to type their login name or password when accessing resources where client authentication is required. When users attempt to access a restricted resource, their client sends the server the client certificate, which the server checks against its list of mappings. If the certificate belongs to a user to whom you've granted access to the resource, the resource is served.

NOTE Requiring client authentication for controlling access to specific resources is different than requiring client authentication for all connections to the server, as described in "Setting Encryption Preferences" on page 220. Also be aware that requiring client certificates for all SSL connections does not automatically map the certificates to users in your databases. To do this, you must specify that a client certificate is required in order to access a specified resource, as described in "Allowing Access to a Resource" on page 59.

Restricting Access

After you have created the users you want to use in access control (see *Managing Netscape Servers*), you use the Restrict Access form to restrict user access to specified URLs.

To change the access control for part of your server,

1. In the Server Manager, choose Server Preferences | Restrict Access. The Restrict Access form appears.
2. Use the drop-down list to choose a regular expression that matches the URLs you want to configure.

   If an expression doesn't exist, click the Regular Expressions button and create an expression. For example, to change access to all URLs in the Netscape domain, type `.*://.*\.netscape\.com/.*` in the field. For more information on regular expressions, see "Understanding Regular Expressions" on page 32.

3. Turn access control off or on for the selected URLs by clicking either the Turn off access control or Turn on access control button.

   Turning on access control causes more access control settings to appear on your screen.

4. For both read and write access, set the default accessibility—allow or deny.

   Read access allows a user only to view the file. *Write* access allows the user to change or delete the file, assuming the user also has access to the file through your server computer's operating system. (Technically, read includes these HTTP methods: GET, HEAD, POST, and INDEX. Write includes PUT, DELETE, MKDIR, RMDIR, and MOVE.)

   When you set these access defaults, they will apply to everyone attempting to read or write to files or directories in the URLs you specify. For example, you can allow users read access to the Netscape domain so they can download software through your proxy server.

5. Specify which users are the exceptions to the default accessibility for each access type by clicking the appropriate Permissions button.

   If the default access is allow, the Deny Access to a Resource form appears (see "Denying Access to a Resource" on page 59). If the default access is deny, the Allow Access to a Resource form appears (see "Allowing Access to a Resource" on page 59). After using those forms, the Server Manager returns you to the Restrict Access form.

6. Choose the response a client will see when access is denied. Under the Access Denied Response heading, click the Respond "Forbidden" button to send a message to the client saying that access to the requested file is forbidden.

   Alternatively, you can click the "Respond with this html file" button and specify an absolute path and filename of an HTML file to send instead of sending the generic "Forbidden" message. Whether or not you specify a file, the server also sends the HTTP error code 404 Not Found.

7. Click the OK button and confirm your changes.

NOTE If you have enabled access control for your server and you want to password-protect local files such as the PAC file, add the following line to the `obj.conf` file:

```
Init fn=init-proxy-auth pac-auth=on
```

Denying Access to a Resource

In the Restrict Access form, you set the default read and write access of a resource (a regular expression of matching URLs). If you set read or write access to allow all access by default, you can specify exceptions by clicking the Permissions button. The Deny Access to a Resource form appears.

When determining who is denied access, you can specify users from specified host names or IP addresses.

First you must specify how host names are processed. If you want to deny users from only the exact host names you'll specify, click Include specified names only. However, if you also want to deny users from alias domains of your specified host names, click Include aliases of specified names.

To deny users from specific host names or IP addresses, type a comma-separated list of host names or IP addresses in the text fields. Restricting by host name is more flexible than restricting by IP address—if a user's IP address changes, you won't have to update this list. However, restricting by IP address is more reliable—if a DNS lookup fails for a connected client, host name restriction cannot be used.

The host name and IP addresses should be specified with a wildcard pattern or a comma-separated list. The wildcard notations you can use are specialized; you can only use the * character. Also, for the IP address, the * must replace an entire byte in the address. That is, 198.95.251.* is acceptable, but 198.95.251.3* is not. When the * character appears in an IP address, it must be the rightmost character. For example, 198.* is acceptable, but 198.*.251.30 is not.

For host names, the * must also replace an entire component of the name. That is, *.netscape.com is acceptable, but *sers.netscape.com is not. When the * appears in a host name, it must be the leftmost character. For example, *.netscape.com is acceptable, but users.*.com is not.

Allowing Access to a Resource

In the Restrict Access form described on page 57, you set the default read and write access of a resource. If you set read or write access to deny all access by default, you can specify exceptions by clicking the Permissions button. The Allow Access to a Resource form appears.

When determining who is allowed access, you can specify two types of users:

- Users from specified host names or IP addresses
- Users (and groups) from your database

You specify both types of users in the Allow Access to a Resource form.

If all types of user authentication are used, the server checks the user's information in the following order (if the criteria in either step 1 or step 2 are met, the client skips the other steps and is allowed access).

1. Is the client's IP address automatically allowed?
2. Is the client's host name automatically allowed?
3. Is the client identified (through password) as one of the allowed users from your database?
4. Is the client's IP address allowed if the user is one of the allowed users from your database?
5. Is the client's host name allowed if the user is one of the allowed users from your database?

When a request for a URL comes in, the server knows the IP address from which the request is coming. Once the server has this address, it uses DNS to look up the host name that corresponds to that IP address.

If you specify from which host names to allow users, decide how you want the host names processed. If you want to allow only users from the exact host names you specify, click Include specified names only. However, if you also want to accept users from alias domains of your specified host names, click Include aliases of specified names.

To allow users from specific host names or IP addresses, enter a wildcard pattern of host names or IP addresses in text fields. Restricting by host name is more flexible than restricting by IP address—if a user's IP address changes, you won't have to update this list. In contrast, restricting by IP address is more reliable—if a DNS lookup fails for a connected client, host name restriction cannot be used.

Users who are allowed access by virtue of their host name or IP address (as in steps 1 and 2 on page 60) are not prompted for a login name or password. All other users are asked for that information.

To allow access to the users listed in your database (LDAP directory), choose the user database containing the appropriate users.

NOTE	You can select whether your proxy server will use a directory server or a local database on the Global Settings page in the administration server.

1. Choose whether to allow everyone from that database or to allow only certain groups and users.
2. Using a comma-separated list, specify the groups in the Groups field or the users in the Users field.

   For example, if your database contains Bob, Juan, Margaret, and Joe but you want only Bob and Margaret to have access to this section, type **`Bob,Margaret`**. If you leave this entry blank, all users from the database are allowed access.
3. To further restrict access, specify any additional host names or IP addresses from which the users in the database must connect. These host names and IP Addresses fields can be left blank if your database users can be from any host names or IP addresses.
4. Specify the message that a user sees when asked for a login name and password by typing it in the Login Prompt field.
5. Click Done.
6. Be sure to click OK in the Restrict Access form when you have finished modifying access control for part of your server.

Chapter 6

Proxying and Routing URLs

This chapter describes how requests are handled by the proxy server. It also explains how to enable proxying for specific resources and to configure the proxy server to route URLs to different URLs or servers.

Enabling Proxying for a Resource

You can turn proxying on or off for resources. Resources can be individual URLs, groups of URLs with something in common, or an entire protocol. You can control whether proxying is on for the entire server, for various resources, or for resources as specified in a template file. This means you can deny access to one or more URLs by turning off proxying for that resource. This can be a global way to deny or allow all access to a resource. (You can also allow or deny access to resources by using URL filters. For more information on URL filters, see “Filtering URLs” on page 151.)

To enable proxying for a resource:

1. In the Server Manager, choose Routing | Enable, Disable.
2. Select the resource you want to configure by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. You can choose a default setting for the resource you specified. You can choose not to proxy that resource (disable proxying), or you can enable proxying of that resource.
    - ❍ Use default setting derived from a more general resource means that the settings for a more general resource that includes this one will be used for this resource.

- Enable proxying of this resource means the proxy lets clients access this resource (provided they pass the other security and authorization checks). When you enable proxying for a resource, *all methods are enabled*. The read methods, including GET, HEAD, PUT, INDEX, POST, and CONNECT for SSL tunneling, and the write methods, including PUT, MKDIR, RMDIR, MOVE, and DELETE, are all enabled for that resource. Barring any other security checks, clients all have read and write access.
- Do not proxy this resource means this resource cannot be reached through the proxy.

4. Click OK.

Configuring Routing for a Resource

You can configure your proxy server to route certain resources using the derived default configuration or direct connections; or you can configure it to route through proxy arrays, an ICP neighborhood, another proxy server, or a SOCKS server. To configure routing for a resource,

1. From the Server Manager, choose Routing | Routing.

   The Routing Configuration form appears.

2. Select the resource you want to configure by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. Select the radio button for the type of routing you would like for the resource you are configuring. You can choose one of the following:
   - Derived default configuration means the proxy server uses a more general template (that is, one with a shorter, matching regular expression) to determine if it should use the remote server or another proxy. For example, if the proxy routes all http://.* requests to another proxy server and all http://www.* requests to the remote server, you could create a derived default configuration routing for http://www.netscape.* requests, which would then go directly to the remote server because of the setting for the http://www.* template.
   - Direct connections means the request will always go directly to the remote server instead of through the proxy.

- Route through a SOCKS server means that requests for the specified resource will be routed through a SOCKS server. If you choose this option, you need to specify the name (or IP address) and the port number of the SOCKS server that the proxy server will route through.
- Route through lets you specify whether you would like to route through a proxy array, ICP neighborhood, parent array, and/or proxy server. If you choose multiple routing methods here, the proxy will follow the hierarchy shown on the form (i.e. proxy array, parent array, ICP, another proxy). For more information on routing through a proxy server, see “Chaining Proxy Servers” on page 65.

  For information on routing through a SOCKS server, see “Routing Through a SOCKS Server” on page 67. For information on routing through proxy arrays, parent arrays, or ICP neighborhoods, see Chapter 9, “Caching.”

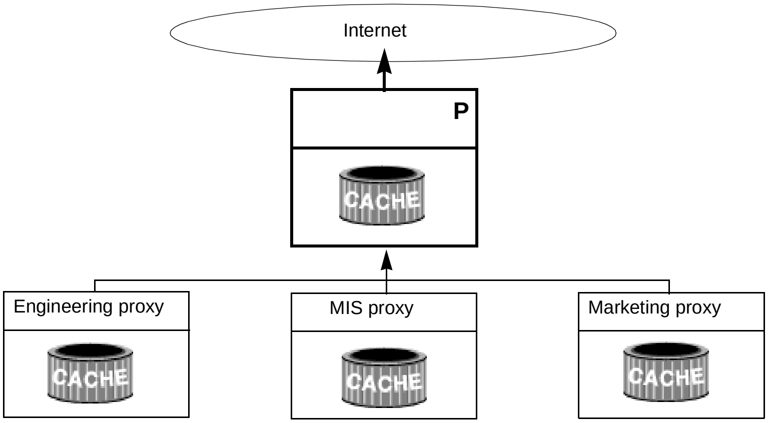
4. Click OK.

Chaining Proxy Servers

You can have the proxy access another proxy for some resources instead of accessing the remote server. This means you can chain proxies together. Chaining is a good way to organize several proxies behind a firewall. Chaining also lets you build hierarchical caching.

For example, you can chain departmental proxies within an organization to a main proxy server, as shown in Figure 6-1. In this figure, each proxy server has a small cache to which a specific group of users has access. Each proxy also has access to the proxy with the large cache. You can also set up several proxies in your organization so that each proxy server accesses and caches only specific files, such as one proxy that services HTTP requests and another that services FTP. Or, you might have one server that caches all files from the .com domain and another that caches all other files.

Figure 6-1 Chaining proxies together



To route through another proxy server,

1. From the Server Manager, choose Routing | Routing. The Routing Configuration form appears.
2. Select the resource you want to route by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. In the "Routing through another proxy" section of the form, select the radio button next to the text "Route through."
4. Select the checkbox next to "another proxy."
5. In the "another proxy" field, enter the name or IP address of the proxy sever that you want to route through.
6. In the port field, enter the port number for the proxy server you will be routing though
7. Click OK.

Routing Through a SOCKS Server

If you already have a remote SOCKS server running on your network, you can configure the proxy to connect to it for specific resources.

To route through a SOCKS server,

1. From the Server Manager, choose Routing | Routing.

   The Routing Configuration form appears.

2. Select the resource you want to route by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. Under the heading, “Routing through another proxy,” select the radio button for next to “Route through SOCKS server.”
4. Specify the name (or IP address) and the port number of the SOCKS server that the proxy server will route through.
5. Click OK.

NOTE Once you have enabling routing through a SOCKS server, you should create proxy routes using the SOCKS v5 Routing form. Proxy routes identify the IP addresses that are accessible through the SOCKS server your proxy routes through. They also specify whether that SOCKS server connects directly to the host. For more information on creating proxy routes, see “Creating SOCKS v5 Routing Entries” on page 100.

Sending the Client’s IP Address to the Server

Normally, the proxy server doesn’t send the client’s IP address to remote servers when making requests for documents. Instead, the proxy acts as the client and sends its IP address to the remote server. This is good protection if you don’t want remote servers to know your internal IP addresses.

However, there are times when you might want to pass on the client’s IP address:

- If your proxy is one in a chain of internal proxies.
- If your clients need to access servers that depend on knowing the client’s IP address. You can use templates to send the client’s IP address only to particular servers.

To configure the proxy to send client IP addresses:

1. In the Server Manager, choose Routing | Client IP Address Forwarding.
2. Choose the template you want to use, or choose the entire proxy server to always send the client's IP address.
3. Choose an option to turn on IP address forwarding.

   By default, the proxy server doesn't send IP addresses, but if you have several proxies in a chain and one proxy forwards the IP address to another, the subsequent proxy will also forward the IP address if its option is set to either default or enabled. Choose enabled to have the proxy server forward the client's IP addresses. Choose blocked to never forward the IP address.
4. You can specify an HTTP header for the proxy to use when forwarding IP addresses.

   The normal HTTP header is named Client-ip, but you can send the IP address in any header you choose.
5. Click OK. Be sure to save and apply your changes.

Allowing Clients to Check IP Addresses

To maintain your network's security, your client may have a feature that restricts access to only certain IP addresses. So that your clients can use this feature, the proxy server provides support for Java IP Address Checking. This support enables your clients to query the proxy server for the IP address used to retrieve a resource. When this feature is enabled, a client can request that the proxy server send the IP address of the origin server, and the proxy server will attach the IP address in a header. Once the client knows the IP address of the origin server, it can explicitly specify that the same IP address be used for future connections.

NOTE Versions of Netscape Navigator prior to 5.0 do not support this feature.

To use Java IP address checking:

1. From the Server Manager, choose Routing | Java IP Address Check. The Java IP Address Check form appears.

2. Select the resource you want to apply IP address checking to by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. Select the radio button to either enable, disable or use the default configuration for Java IP address checking.

NOTE The default option uses a derived default configuration from a more general template (that is, one with a shorter, matching regular expression) to determine whether Java IP address checking should be enabled or disabled.

4. Click OK.

Disconnecting the Proxy from the Network

You can connect or disconnect the proxy server machine from the network. This feature makes it convenient to install the proxy on a portable machine that you can use for demonstrations.

When the proxy is disconnected from the network, documents are returned directly from the cache—the proxy can't do up-to-date checks, so the documents are retrieved very quickly (the documents might not be up to date; see Chapter 9, "Caching for more information on caching).

Also, if you are not connected to a network, connections never hang because the proxy server is aware that there is no network and never tries to connect to a remote server. You can use this no-network setting when the network is down but the proxy server machine is running.

NOTE Keep in mind that running the proxy disconnected from the network means that you will eventually be accessing stale data from the cache. Also, running without the network makes the proxy security features unnecessary.

iPlanet Web Proxy Server offers four network connectivity modes:

Default mode is derived from the configuration of the most general matching object.

Normal mode is the normal operating mode for the proxy. The proxy retrieves documents from the content server if they are not already in the cache. If they are in the cache, they may be checked against the content server to determine if they are up to date. If a cached file has changed, it is replaced with the current copy.

Fast-demo mode is intended for giving smooth demonstrations when the network is available. If a document is found in the cache, the content server is not contacted, not even to find out if the document has changed. This mode gets rid of any latency created by waiting for the content server to respond. If a document is not in the cache, it is retrieved from the content server and cached. The fast-demo mode has less latency than the normal mode, but can occasionally return stale data, because once it has a copy of a document, it doesn't do up-to-date checks on it.

No-network mode is designed for portable machines during the time they are not connected to the network. The proxy returns the document if it is in the cache or returns an error if it isn't. The proxy never tries to contact the content server, which prevents the proxy from hanging and timing out while trying to get a connection that doesn't exist.

To change the running mode for the proxy server:

1. In the Server Manager, choose Routing | Connectivity Mode.
2. Choose the template you want to use or choose to change the mode for the entire proxy server.
3. Select the mode you want
4. Click OK.

Be sure to save and apply your changes.

Changing the Default FTP Transfer Mode

FTP has two different ways to establish a data connection between the FTP server and the client (the proxy acts as a client). The two modes are referred to as PASV (Passive) and PORT (Active) mode FTP.

- PASV Mode (the default) means the data connection is initiated from the proxy server, and the FTP server accepts the connection. This is safer for the site running the proxy server because it doesn't have to accept inbound connections.

- PORT Mode means the data connection is initiated by the remote FTP server, and the proxy accepts the incoming connection. If the proxy server is within a firewall, the firewall might block the incoming FTP data connection from the FTP server, which means the PORT mode might not work.

Some FTP sites run a firewall, which makes PASV mode non-functional for proxy servers. Because of this, the proxy server can be configured to use the PORT mode FTP. You can turn on PORT mode for the entire server, or you can turn it on only for specific FTP servers.

NOTE Even when PASV mode is on, the proxy server will use PORT mode if the remote FTP server doesn't support PASV mode.

If the proxy server is behind a firewall that makes the PORT mode FTP non-functional, you can't enable PORT mode. If default is selected for the resource, the proxy server uses the mode from a more general resource. If none is specified, PASV mode will be used.

Mapping URLs to Other URLs

The Server Manager lets you map URLs to another server, sometimes called a "mirror" server. When a client accesses the proxy with a mirrored URL, the proxy retrieves the requested document from the mirrored server and not from the server specified in the URL. The client is never aware that the request is going to a different server. You can also redirect URLs; in this case, the proxy returns only the redirected URL to the client (and not the document), so the client can then request the new document. Mapping also allows you to map URLs to a file, as in PAC and PAT mappings.

To map a URL, you specify a URL prefix and where to map it. The following sections describe the various types of URL mappings.

Creating a URL Mapping

You can create four types of URL mappings:

- *Regular mappings* map a URL prefix to another URL prefix. For example, you can configure the proxy to go to a specific URL anytime it gets a request that begins http://www.netscape.com.

- *Reverse mappings* map a redirected URL prefix to another URL prefix. These are used with reverse proxies when the internal server sends a redirected response instead of the document to the proxy. See Chapter 7, “Reverse Proxy for more information.
- *Regular expressions* map all URLs matching the expression to a single URL. For example, you can map all URLs matching .*sex.* to a specific URL (perhaps one that explains why the proxy server won’t let a user go to a particular URL). For more information on regular expressions, see “Understanding Regular Expressions” on page 32.
- *Client autoconfiguration* maps URLs to a specific `.pac` file stored on the proxy server. For more information on autoconfiguration files, see Chapter 11, “Using the Client Autoconfiguration File
- *Proxy array table* (PAT) maps URLs to a specific `.pat` file stored on the proxy server. You should only create this type of mapping from a master proxy. For more information on PAT files and proxy arrays, see “Routing through Proxy Arrays” on page 130.

Clients accessing a URL are sent to a different location on the same server or on a different server. This is useful when a resource has moved or when you need to maintain the integrity of relative links when directories are accessed without a trailing slash.

For example, suppose you have a heavily loaded web server called hi.load.com that you want mirrored to another server called mirror.load.com. For URLs that go to the hi.load.com computer, you can configure the proxy server to use the mirror.load.com computer.

The source URL prefix must be unescaped, but in the destination (mirror) URL, only characters that are illegal in HTTP requests need to be escaped.

CAUTION Do not use trailing slashes in the prefixes!

To create a URL mapping:

1. In the Server Manager, choose URLs | Create Mappings.
2. Choose the type of mapping you want to create.

3. Type the URL prefix. For regular and reverse mappings, this should be the part of the URL you want to substitute.

   For regular expression mappings, the URL prefix should be a regular expression that for all the URLs you want to match. If you also choose a template for the mapping, the regular expression will work only for the URLs within the template's regular expression. For more information on regular expressions, see "Understanding Regular Expressions" on page 32.

   For client autoconfiguration mappings and proxy array table mappings, the URL prefix should be the full URL the client accesses.

4. Type a map destination.

   For all mapping types except client autoconfiguration and proxy array table, this should be the full URL to which to map. For client autoconfiguration mappings, this value should be the absolute path to the `.pac` file on the proxy server's hard disk. For proxy array table mappings, this value should be the absolute path to the `.pat` file on the master proxy's local disk.

5. Click OK to create the mapping.

Editing Existing Mappings

To change your existing mappings,

1. In the Server Manager, choose URLs | View / Edit Mappings.

   The View, Edit, or Remove URL Mappings form appears. You can edit the prefix, the mapped URL, and template that are affected by the mapping.

2. To remove a mapping, click the mapping you want to change, then click the Remove link at the top of the form.

3. Click OK to confirm your changes, or click Reset to undo them.

Redirecting URLs

You can configure the proxy server to return a redirected URL to the client instead of getting and returning the document. With redirection, the client is aware that the URL originally requested has been redirected to a different URL. The client usually requests the redirected URL immediately. Netscape Navigator automatically requests the redirected URL—the user doesn't have to explicitly request the document a second time.

URL redirection is useful when you want to deny access to an area because you can redirect the user to a URL that explains why access was denied.

To redirect one or more URLs,

1. In the Server Manager, choose URLs | Redirections.
2. Enter a source URL. Your source URL can be either a URL prefix or a regular expression.

   If you choose to use a URL prefix as the source, select the radio button next to the URL prefix field and enter a URL prefix. If you choose to use a regular expression as the source, you should select the radio button next to the Reg. expr. field and then enter a regular expression.

NOTE If you use a regular expression as the source URL, you must use a fixed URL as the URL to which requests will be redirected.

3. Enter a URL to redirect to. This URL can either be a URL prefix or a fixed URL. However, if your source URL is a regular expression, you must use a fixed URL as the URL to which to redirect.

   If you choose to use a URL prefix as the URL to redirect to, select the radio button next to the URL prefix field and enter a URL prefix. If you choose to use a fixed URL, select the radio button next to the Fixed URL field and enter a fixed URL.
4. Click OK to create the mapping.

Specifying the SOCKS Name Server IP Address

If your proxy is configured to make its outbound connections through a SOCKS server, you may need to explicitly specify the IP address for the name server to be used with SOCKS.

You should specify the name server IP address if you are resolving outside host names with a DNS server other than an internal DNS service that is inside the firewall.

To specify the SOCKS name server IP address,

1. In the Server Manager, choose Routing | SOCKS Name Server. The SOCKS Name Server Setting form appears.

2. Enter the IP address of the DNS name server in text field.
3. Click OK.

NOTE The feature that allows you to specify the SOCKS name server IP address used to only be accessible via the SOCKS_NS environment variable. If you set the environment variable and use the SOCKS Name Server Setting form to specify the name server IP address, the proxy will use the IP address specified on the form instead of the environment variable.

Client Autoconfiguration

If your proxy server supports many clients, you can use a client autoconfiguration file to configure all of your Netscape Navigator clients. The autoconfiguration file contains a JavaScript function that determines which proxy, if any, Navigator uses when accessing various URLs. For more information on this feature, see Chapter 11, “Using the Client Autoconfiguration File.”

Chapter 7

Reverse Proxy

This chapter describes how to use iPlanet Web Proxy Server as a reverse proxy. *Reverse proxy* is the name for certain alternate uses of a proxy server. It can be used outside the firewall to represent a secure content server to outside clients, preventing direct, unmonitored access to your server's data from outside your company. It can also be used for replication; that is, multiple proxies can be attached in front of a heavily used server for load balancing. This chapter describes the alternate ways that iPlanet Web Proxy Server can be used inside or outside a firewall.

How Reverse Proxying Works

There are two models for reverse proxying. One model takes advantage of iPlanet Web Proxy Server's security features to handle transactions, and the other makes use of its caching features to provide load balancing on a heavily used server. Both of these models differ from the conventional proxy usage in that they don't operate strictly on a firewall.

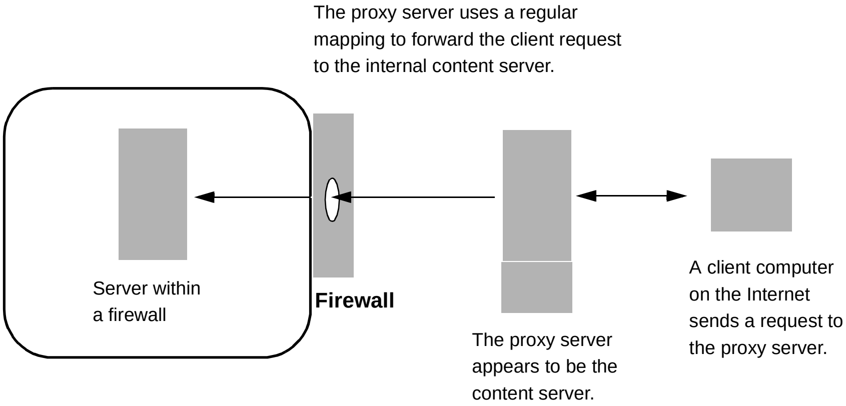
Proxy as a Stand-in for a Server

If you have a content server that has sensitive information that must remain secure, such as a database of credit card numbers, you can set up a proxy outside the firewall as a stand–in for your content server. When outside clients try to access the content server, they are sent to the proxy server instead. The real content resides on your content server, safely inside the firewall. The proxy server resides outside the firewall, and appears to the client to be the content server.

When a client makes a request to your site, the request goes to the proxy server. The proxy server then sends the client's request through a specific passage in the firewall to the content server. The content server passes the result through the passage back to the proxy. The proxy sends the retrieved information to the client, as if the proxy were the actual content server (see Figure 7-1). If the content server returns an error message, the proxy server can intercept the message and change any URLs listed in the headers before sending the message to the client. This prevents external clients from getting redirection URLs to the internal content server.

In this way, the proxy provides an additional barrier between the secure database and the possibility of malicious attack. In the unlikely event of a successful attack, the perpetrator is more likely to be restricted to only the information involved in a single transaction, as opposed to having access to the entire database. The unauthorized user can't get to the real content server because the firewall passage allows only the proxy server to have access.

Figure 7-1 A reverse proxy appears to be the real content server.


The proxy server uses a regular mapping to forward the client request to the internal content server.
Server within a firewall
Firewall
The proxy server appears to be the content server.
A client computer on the Internet sends a request to the proxy server.

You can configure the firewall router to allow a specific server on a specific port (in this case, the proxy on its assigned port) to have access through the firewall without allowing any other machines in or out.

Secure Reverse Proxying

Secure reverse proxying occurs when one or more of the connections between the proxy server and another machine uses the Secure Sockets Layer (SSL) protocol to encrypt data. For more information about SSL and encryption, see Chapter 14, "Understanding Encryption and SSL

Secure reverse proxying has many uses:

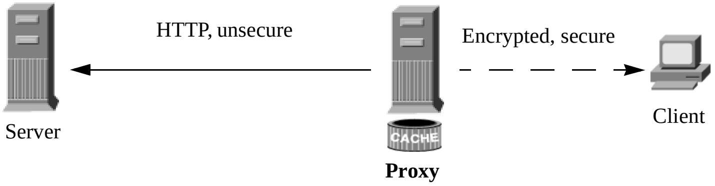
- It can provide an encrypted connection from a proxy server outside a firewall to a secure content server inside the firewall.
- It can allow clients to connect securely to the proxy server, facilitating the secure transmission of information (such as credit card numbers).

Secure reverse proxying causes each secure connection to be slower due to the overhead involved in encrypting your data. However, because SSL provides a caching mechanism, two connecting parties can reuse previously negotiated security parameters, dramatically reducing the overhead on subsequent connections.

There are three ways to configure a secure reverse proxy:

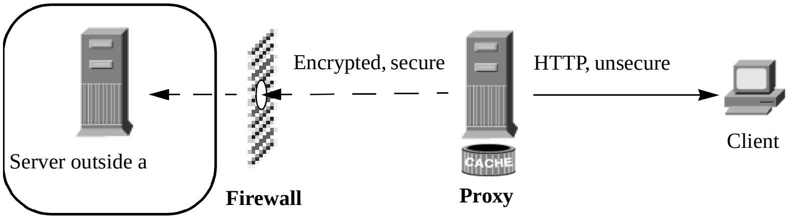
- **Secure client to proxy**. This scenario is effective if there is little or no chance that the information being exchanged between your proxy and content server can be accessed by unauthorized users (see Figure 7-2).

Figure 7-2 Secure client connection to proxy



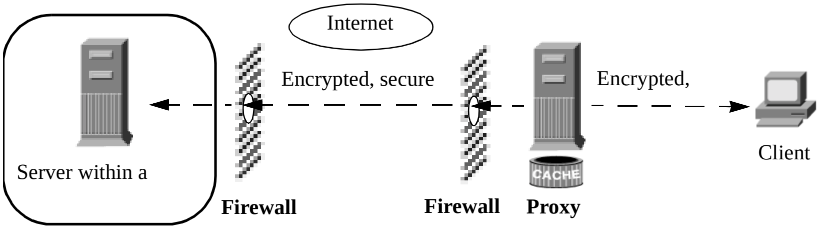
- **Secure proxy to content server**. This scenario is effective if you have clients inside the firewall and a content server that is outside the firewall. In this scenario, your proxy server can act as a secure channel between sites (see Figure 7-3).

Figure 7-3 Secure proxy connection to content server



- **Secure client to proxy and secure proxy to content server**. This scenario is effective if the information exchanged between the server, proxy and client needs to be secure. In this scenario, your proxy server can act like a secure channel between sites with the additional security of client authentication (see Figure 7-4).

Figure 7-4 Secure client connection to proxy and secure proxy connection to content server



For information on how to set up each of these configurations, see “Setting up a Reverse Proxy” on page 82.

In addition to SSL, the proxy can use client authentication, which requires that a computer making a request to the proxy provides a certificate (or form of identification) to verify its identity. For more information on client authentication, see the section entitled “What is Client Authentication?” on page 225.

Proxying for Load Balancing

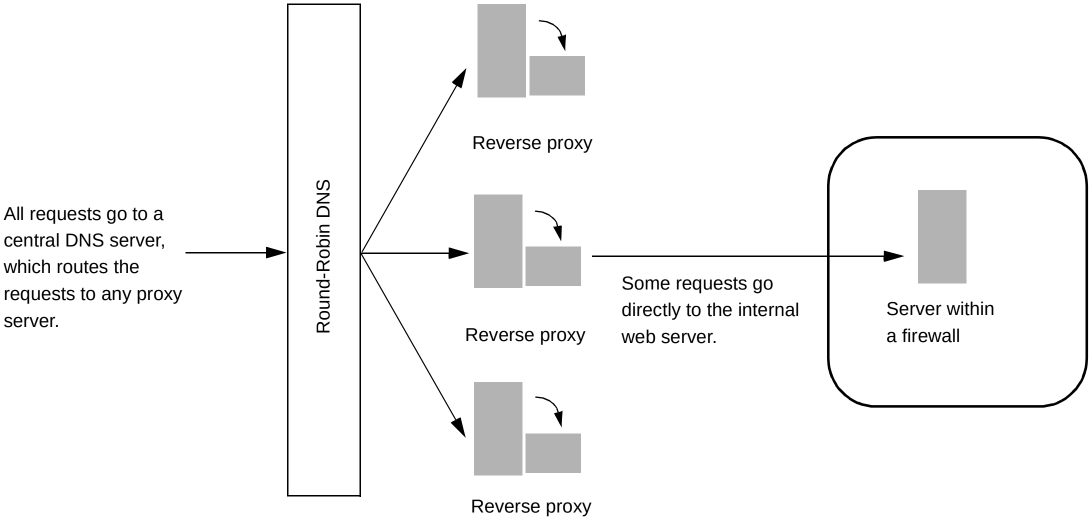
You can use multiple proxy servers within an organization to balance the network load among web servers. This model lets you take advantage of the caching features of the proxy server to create a server pool for load balancing. In this case, the proxy servers can be on either side of the firewall. If you have a web server that receives a high number of requests per day, you could use proxy servers to take the load off the web server and make the network access more efficient.

The proxy servers act as go-betweens for client requests to the real server. The proxy servers cache the requested documents. If there is more than one proxy server, DNS can route the requests randomly using a “round-robin” selection of their IP addresses. The client uses the same URL each time, but the route the request takes might go through a different proxy each time.

The advantage of using multiple proxies to handle requests to one heavily used content server is that the server can handle a heavier load, and more efficiently than it could alone. After an initial start-up period in which the proxies retrieve documents from the content server for the first time, the number of requests to the content server can drop dramatically.

Only CGI requests and occasional new requests must go all the way to the content server. The rest can be handled by a proxy. Here’s an example. Suppose that 90% of the requests to your server are not CGI requests (which means they can be cached), and that your content server receives 2 million hits per day. In this situation, if you connect three reverse proxies, and each of them handles 2 million hits per day, about 6 million hits per day would then be possible. The 10% of requests that reach the content server could add up to about 200,000 hits from each proxy per day, or only 600,000 total, which is far more efficient. The number of hits could increase from around 2 million to 6 million, and the load on the content server could decrease correspondingly from 2 million to 600,000. Your actual results would depend upon your situation.

Figure 7-5 Proxy used for load balancing


All requests go to a central DNS server, which routes the requests to any proxy server.
Round-Robin DNS
Reverse proxy
Reverse proxy
Reverse proxy
Some requests go directly to the internal web server.
Server within a firewall

Setting up a Reverse Proxy

To set up a reverse proxy, you need two mappings: a regular and a reverse mapping.

- The regular mapping redirects requests to the content server. When a client requests a document from the proxy server, the proxy server needs a regular mapping to tell it where to get the actual document.

CAUTION You shouldn't use a reverse proxy with a proxy that serves autoconfiguration files. This is because the proxy could return the wrong result. See Chapter 11, "Using the Client Autoconfiguration File for more information on using autoconfiguration files with a reverse proxy.

- The reverse mapping makes the proxy server trap for redirects from the content server. The proxy intercepts the redirect and then changes the redirected URL to map to the proxy server. For example, if the client requests a document that was moved or not found, the content server will return a message to the client explaining that it can't find the document at the requested

URL. In that returned message, the content server adds an HTTP header that lists a URL to use to get the moved file. In order to maintain the privacy of the internal content server, the proxy can redirect the URL using a reverse mapping.

Suppose you have a web server called http://http.site.com/ and you want to set up a reverse proxy server for it. You could call the reverse proxy http://proxy.site.com/.

You would create a *regular* mapping and a *reverse* mapping as follows:

1. In the Server Manager, choose URLs|Create Mappings.

   In the form that appears, enter information for a single mapping. For example:

   Regular mapping:

   Source prefix: `http://proxy.site.com`

   Source destination: `http://http.site.com/`

2. Click OK. Return to the form and create the second mapping:

   Reverse mapping:

   Source prefix: `http://http.site.com/`

   Source destination: `http://proxy.site.com/`

3. To make the change, click the OK button.

   Once you click the OK button, the proxy server adds one or more additional mappings. To see the mappings, click the link called View/Edit Mappings. Additional mappings would be in the following format:

   from: /

   to: http://http.site.com/

   These additional automatic mappings are for users who connect to the reverse proxy as a normal server. The first mapping is to catch users connecting to the reverse proxy as a regular proxy. Depending on the setup, usually the second is the only one required, but it doesn't cause problems in the proxy to have them both.

NOTE If the web server has several DNS aliases, each alias should have a corresponding regular mapping. If the web server generates redirects with several DNS aliases to itself, each of those aliases should have a corresponding reverse mapping.

CGI applications still run on the origin server; the proxy server never runs CGI applications on its own. However, if the CGI script indicates that the result can be cached (by implying a non-zero time-to-live by issuing a Last-modified or Expires header), the proxy will cache the result.

CAUTION When authoring content for the web server, keep in mind that the content will be served by the reverse proxy, too, so all links to files on the web server should be relative links. There must be *no* reference to the host name in the HTML files; that is, all links must be of the form:

/abc/def

as opposed to a fully qualified host name, such as:

http://http.site.com/abc/def

Setting up a Secure Reverse Proxy

Before setting up secure reverse proxying, you should be familiar with digital certificates, Certificate Authorities, and authentication. For more information on these subjects, see Chapter 14, “Understanding Encryption and SSL.”

Setting up a secure reverse proxy is almost the same as setting up an unsecure reverse proxy. The only difference is that you need to specify HTTPS as the protocol for the files to be encrypted. For more information on proxying HTTPS, see “Enabling HTTPS Proxying” on page 218. For more information on setting up a reverse proxy, see “Setting up a Reverse Proxy,” on page 82.

The following instructions explain how to set up your secure reverse proxy according to the configuration scenario you choose. To demonstrate how to set up mappings, the instructions suppose that you have a web server called http.site.com and that you want to set up a secure reverse proxy server called proxy.site.com. When following the steps, substitute the name of your web server and proxy for the example names used in the directions.

Secure Client to Proxy

1. In the Server Manager, choose URLs | Create Mappings. In the form that appears, set up regular and reverse mappings in the following manner:

   Regular mapping:

   Source prefix: `https://proxy.mysite.com`

   Source destination: `http://http.mysite.com/`

   Reverse mapping:

   Source prefix: `http://http.mysite.com/`

   Source destination: `https://proxy.mysite.com/`

2. Save and Apply your changes.

   To see the mappings you just created, click the link called View / Edit Mappings.

NOTE This configuration will only work if your proxy server is running in secure mode. In other words, encyrption must be enabled and the proxy must be restarted from the command line. To restart the proxy from the command line, go to the proxy directory and type `./start`.

Secure Proxy to Content Server

1. In the Server Manager, choose URLs | Create Mappings.

   In the form that appears, set up regular and reverse mappings in the following manner:

   Regular mapping:

   Source prefix: `http://proxy.mysite.com`

   Source destination: `https://http.mysite.com/`

   Reverse mapping:

   Source prefix: `https://http.mysite.com/`

   Source destination: `http://proxy.mysite.com/`

2. Save and Apply your changes. To see the mappings you just created, click the link called View / Edit Mappings.

NOTE This configuration will only work if your content server is running in secure mode.

Secure Client to Proxy and Secure Proxy to Content Server

1. In the Server Manager, choose URLs | Create Mappings.

   In the form that appears, set up regular and reverse mappings in the following manner:

   Regular mapping:

   Source prefix: `https://proxy.mysite.com`

   Source destination: `https://http.mysite.com/`

   Reverse mapping:

   Source prefix: `https://http.mysite.com/`

   Source destination: `https://proxy.mysite.com/`

2. Save and Apply your changes. To see the mappings you just created, click the link called View / Edit Mappings.

NOTE This configuration will only work if your proxy server and content server are running in secure mode. In other words, for the proxy, encryption must be enabled and the proxy must be restarted from the command line. To restart the proxy from the command line, go to the proxy directory and type `./restart`.

Virtual Multihosting in Reverse Proxy

Virtual multihosting is a feature which allows an origin server, or in our case, a reverse proxy server, to respond to multiple DNS aliases as if there was a different server installed in each of those addresses. As an example, you could have the DNS hostnames:

- www
- specs
- phones

Each of them could be mapped to the same IP address (the IP address of the reverse proxy). You could then have the reverse proxy act differently based on which DNS name was used to access it.

Virtual Multihosting allows you to host multiple different *domains* in a single reverse proxy server as well. For example:

- `www.domain-1.com`
- `www.domain-2.com`
- `www.domain-3.com`

Note that you can have a combination of multiple local hostnames as well as multiple domains, all in a single proxy server:

- `www`
- `specs`
- `phones`
- `www.domain-1.com`
- `www.domain-2.com`
- `www.domain-3.com`

Functional Details of Virtual Multihosting

The virtual multihosting feature works by specifying the DNS host and domain names (or aliases), and then giving a target URL prefix where requests sent to that hostname should be directed. As an example, you can have two mappings:

- engr.domain.com -> `http://int-engr.domain.com`
- mktg.domain.com -> `http://int-mktg.domain.com`

Mappings do not have to go root-to-root; you may specify an additional URL path prefix in the target URL:

- engr.domain.com -> `http://internal.domain.com/engr`
- mktg.domain.com -> `http://internal.domain.com/mktg`

Same applies to virtual domain mappings. For example, you could use:

- www.domain-1.com -> `http://int-engr.domain.com`
- www.domain-2.com -> `http://int-mktg.domain.com`

The system will look at the HTTP "Host:" header, and based on that header, it will choose the matching Virtual Multihosting mapping. If none of the multihosting mappings match, the server will continue looking at other mappings in the order that they appear in the configuration file, or perform no mappings if no matches are found. If there are no matches, the proxy will typically respond with the "Proxy denies fulfilling the request" response.

Configuring Virtual Multihosting

To configure Virtual Multihosting:

1. From the Server Manager, choose URLs | Virtual Multihosting.
2. In the "Source hostname (alias)" field, specify the local hostname (or DNS alias) that this mapping should apply to.
3. In the "Source domain name" field, enter the local domain name that this mapping should apply to. Typically, this is your own network's domain name, unless you want to multi-host multiple different DNS domains.
4. In the "Destination URL prefix" field, enter the target URL prefix where the request will be directed if the host and domain names match the above specifications.
5. If you are using templates, choose the template name from the "Use this template" dropdown, or leave the value at "NONE" if you don't want to apply a template.
6. Click OK, and save and apply your changes.
7. Repeat the above steps for each virtual multihosting mapping you want to establish.

All virtual multihosting mappings appear on the bottom of the Virtual Multihosting configuration page. Note that the "Source hostname (alias)" and "Source domain name" fields are merged, together with the proxy's port number, into a single regular expression that is used to match the "Host:" header.

For example, if you have hostname "www", domain "netscape.com", and port number "8080", it will display the regular expression:

```
www(|.netscape.com)(|:8080)
```

This will guarantee a match with all of the following possible combinations that the user may have typed, or the client may have sent (the port number may be omitted by some client software even when it's non-80, as it is obvious to the server which port number it was listening on):

- `www`

- `www:8080`
- `www.netscape.com`
- `www.netscape.com:8080`

Important Notes on Virtual Multihosting

You will need to disable the "Client autoconfiguration" feature before you can configure reverse proxy mappings. Doing so will not cause any problems because the "Client autoconfiguration" feature is for the forward proxy operation, not reverse proxy.

The Virtual Multihosting feature establishes “automatic reverse mappings.” In other words, do not create reverse mappings (Create Mappings | Mapping type: reverse) for mappings that you enter using the Virtual Multihosting page.

Virtual mappings are specified with "virt-map" function in `obj.conf`.

Virtual mappings are matched in the order specified in the `obj.conf` configuration file. If there are regular, reverse, regular expression, or client autoconfiguration mappings before the virtual mappings, they will be applied first. Similarly, if no matches are found in virtual mappings, translation will continue to the next mapping after the virtual mapping section in `obj.conf`.

If the port number of the proxy server is changed, you will need to recreate the Virtual Multihosting mappings, as they now have the wrong port number.

Chapter 8

Using SOCKS v5

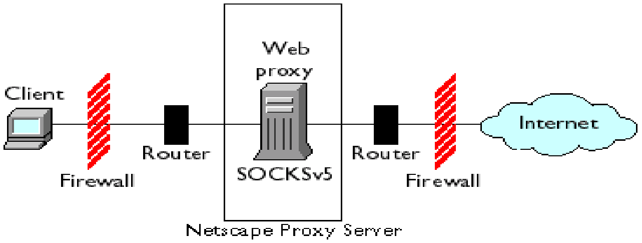
This chapter explains how to configure and use the SOCKS v5 server that comes with iPlanet Web Proxy Server.

Using a SOCKS Server

The SOCKS server is a generic firewall daemon that controls access through the firewall on a point-to-point basis. The SOCKS server works at the network level instead of the application level, and therefore has no knowledge of protocols or methods used for transferring requests. Because the SOCKS server has no knowledge of protocols, it can be used to pass those protocols which are not supported by the proxy server, such as telnet. iPlanet Web Proxy Server supports SOCKS versions 4 and 5.

iPlanet Web Proxy Server comes with a separate SOCKS daemon that understands the usual `socks5.conf` file format used by other SOCKS daemons. See "The socks5.conf File" on page 468 for information on this file format. By default, the SOCKS daemon features are disabled, but you can enable them through the SOCKS On / Off form.

Figure 8-1 SOCKS v5



You can also use the Routing Configuration form to configure your proxy to route requests through a SOCKS server. For more information on routing requests through a SOCKS server, see “Routing Through a SOCKS Server” on page 67.

To use the SOCKS server:

1. Configure SOCKS v5.
2. If SOCKS v5 will be running on a machine with multiple interfaces, create SOCKS routing entries.
3. Create authentication entries.
4. Create connection entries.
5. Enable the SOCKS server.

Configuring SOCKS v5

To configure your SOCKS server:

1. From the Server Manager, choose SOCKS | Configuration. The SOCKS v5 Configuration form appears.
2. In the SOCKS Port field, enter the port number on which the SOCKS server will listen.

3. Choose the checkbox for the SOCKS options you want to use. The options are:

   disable reverse DNS lookup - disables reverse DNS lookup for your SOCKS server. Reverse DNS translates IP addresses into host names. Disabling reverse DNS lookup can conserve network resources.

   use client-specific bind port - allows the client to specify the port in a BIND request. With this option disabled, SOCKS ignores the client's requested port and assigns a random port.

   allow wildcard as bind IP address - allows the client to specify an IP address of all zeros (0.0.0.0) in a BIND request. An IP address of all zeros means that any IP address can connect. With this option disabled, the client must specify the IP address that will be connecting to the bind port and the SOCKS server rejects requests to bind to 0.0.0.0.

4. In the Log File field, enter the full pathname of the SOCKS log file.

5. From the Log Level pull-down, choose whether you want the log file to contain warnings and errors only, all requests, or debugging messages.

6. If you want to disable the automatic logging general SOCKS statistics once an hour, select the "quench updates" checkbox.

7. Select the radio button to choose an RFC 1413 Ident Policy. Ident allows the SOCKS server to determine the user name for a client. Generally, this feature only works when the client is running UNIX. The available policies are:

   don't ask - never use Ident to determine the user name for a client. This is the recommended setting.

   ask but don't require - ask for the user name of all clients, but do not require it. This option uses Ident for logging purposes only.

   require - ask for the user name of all clients and only permit access to those with valid responses.

8. Click OK.

Creating SOCKS v5 Authentication Entries

SOCKS authentication entries identify the hosts from which the SOCKS deamon should accept connections and which types of authentication the SOCKS daemon should use to authenticate these hosts.

To create a SOCKS authentication entry:

1. From the Server Manager, choose SOCKS | Authentication.

   The SOCKS v5 Authentication Entry form appears.

2. Click the Add button.

   The SOCKS v5 Authentication Entry form appears.

3. In the Host mask field, enter the IP addresses or host names of the hosts that the SOCKS server will authenticate. If you enter an IP address, follow the address with a forward slash and the mask to be applied to the incoming IP address. The SOCKS server will apply this mask to the IP address to determine if it is a valid host. There cannot be any spaces in the Host mask entry. If you do not enter a host mask, the authentication entry will apply to all hosts.

   For example, you can enter "155.25.0.0/255.255.0.0" into the Host mask field. If the host's IP address is 155.25.3.5, the SOCKS server will apply the mask to the IP address and determine that the host's IP address matches the IP address for which the authentication record applies (155.25.0.0).

4. In the Port range field, enter the ports on the host machines that the SOCKS server will authenticate. There should not be any spaces in your port range. If you do not enter a port range, the authentication entry will apply to all ports.

   You can use brackets [ ] to include the ports at each end of the range or parentheses ( ) to exclude them. For example [1000-1010] means all port numbers between and including 1000 and 1010. (1000-1010) means all port numbers between, but not including 1000 and 1010. You can also mix brackets and parentheses. For instance, (1000-1010] means all numbers between 1000 and 1010, excluding 1000, but including 1010.

5. From the Authentication type pull-down, choose one of the following:

   - require user password - user name and password are required to access the SOCKS server
   - user-password if available - if a user name and password are available, they should be used to access the SOCKS server; but they are not required for access
   - ban - banned from the SOCKS server
   - none - no authentication is required to access the SOCKS server

6. From the "Insert" pull-down, select the position in the socks5.conf file that you want the authentication entry to be in.

   Because you can have multiple authentication methods, you need to specify the order in which they are evaluated. Therefore, if the client does not support the first authentication method listed, the second method will be used instead. If the client does not support any of the authentication methods listed, the SOCKS server will disconnect without accepting a request.

7. Click OK.

Editing SOCKS v5 Authentication Entries

To edit a SOCKS v5 authentication entry,

1. From the Server Manager, choose SOCKS | Authentication.

   The SOCKS v5 Authentication Entry form appears.

2. Select the radio button next to the authentication entry that you want to edit.
3. Click the Edit button.

   The SOCKS v5 Authentication Entry form appears.

4. Edit the appropriate information.
5. Click OK.

Deleting SOCKS v5 Authentication Entries

To delete a SOCKS v5 authentication entry,

1. From the Server Manager, choose SOCKS | Authentication. The SOCKS v5 Authentication Entry form appears.
2. Select the radio button next to the authentication entry that you want to delete.
3. Click the Delete button.
4. Click OK.

Moving SOCKS v5 Authentication Entries

Because you can have multiple authentication methods, the entries are evaluated in the order in which they appear in the `socks5.conf` file. You may want to change the order in which they are evaluated by moving them.

To move an authentication entry:

1. From the Server Manager, choose SOCKS | Authentication.

   The SOCKS v5 Authentication form appears.

2. Select the radio button next to the authentication entry that you want to move.
3. Click the Move button.

   The SOCKS v5 Move Entry form appears.

4. From the Move pull-down, choose the position in the `socks5.conf` file that you want the authentication entry to be in.

   Because you can have multiple authentication methods, you need to specify the order in which they are evaluated.

5. Click OK.

Creating SOCKS v5 Connection Entries

SOCKS connection entries specify whether the SOCKS daemon should permit or deny a request.

1. From the Server Manager, choose SOCKS | Connections. The SOCKS v5 Connections form appears.
2. Click the Add button. The SOCKS v5 Connection Entry form appears.
3. From the Authentication Type pull-down, choose the authentication method for which this access control line applies.
4. From the Connection Type pull-down, choose the type of command the line matches. Possible command types are:
   - connect
   - bind (open a listen socket)
   - UDP relay
   - all

5. In the Source host mask field, enter the IP address or host names of the hosts for which the connection control entry applies. If you enter an IP address, follow it with a forward slash and the mask to be applied to the source's IP address. The SOCKS server will apply this mask to the source's IP address to determine if it is a valid host. There cannot be any spaces in the host mask entry. If you do not enter a host mask, the connection entry will apply to all hosts.

   For example, you can enter "155.25.0.0/255.255.0.0" into the host mask field. If the host's IP address is 155.25.3.5, the SOCKS server will apply the mask to the IP address and determine that the host's IP address matches the IP address for which the connection control entry applies (155.25.0.0).

6. In the Port range field, enter the ports on the source machines for which the connection control entry applies. There should not be any spaces in your port range. If you do not specify a port range, the connection entry will apply to all ports.

   You can use brackets [ ] to include the ports at each end of the range or parentheses ( ) to exclude them. For example [1000-1010] means all port numbers between and including 1000 and 1010. (1000-1010) means all port numbers between, but not including 1000 and 1010. You can also mix brackets and parentheses. For instance, (1000-1010] means all numbers between 1000 and 1010, excluding 1000, but including 1010.

7. In the Destination host mask field, enter the IP address or host name for which the connection entry applies. If you enter an IP address, follow it with a forward slash and the mask to be applied to the incoming IP address. The SOCKS server will apply this mask to the IP address of the destination machine to determine if it is a valid destination host. There cannot be any spaces in the host mask entry. If you do not enter a destination host mask, the connection entry applies to all hosts.

   For example, you can enter "155.25.0.0/255.255.0.0" into the Destination host mask field. If the destination host's IP address is 155.25.3.5, the SOCKS server will apply the mask to the IP address and determine that the destination host's IP address matches the IP address for which the proxy entry applies (155.25.0.0).

8. In the second Port range field, enter the ports on the destination host machines for which the connection control entry applies. There should not be any spaces in your port range. If you do not enter a port range, the connection entry applies to all ports.

NOTE Most SOCKS applications will request port 0 for bind requests, meaning they have no port preference. Therefore, the destination port range for bind should always include port 0.

You can use brackets [ ] to include the ports at each enge of the range or parentheses ( ) to exclude them. For example [1000-1010] means all port numbers between and including 1000 and 1010. (1000-1010) means all port numbers between, but not including 1000 and 1010. You can also mix brackets and parentheses. For instance, (1000-1010] means all numbers between 1000 and 1010, excluding 1000, but including 1010.

9. In the User group field, enter the group to deny or permit access to. If you do not specify a group, the connection entry will apply to all users.
10. From the Action pull-down, choose to permit or deny access for the connection you are creating.
11. From the Insert pull-down, choose the position in the `socks5.conf` file that you want the connection entry to be in. Because you can have multiple connection directives, you need to specify the order in which they are evaluated.

Editing SOCKS v5 Connection Entries

To edit a SOCKS v5 connection entry,

1. From the Server Manager, choose SOCKS | Connections. The SOCKS v5 Connections form appears.
2. Select the radio button next to the connection entry that you want to edit.
3. Click the Edit button. The SOCKS v5 Connections Entry form appears.
4. Edit the appropriate information.
5. Click OK.

Deleting SOCKS v5 Connection Entries

To delete a SOCKS v5 connection entry:

1. From the Server Manager, choose SOCKS | Connections. The SOCKS v5 Connections form appears.
2. Select the radio button next to the connection entry that you want to delete.
3. Click the Delete button.
4. Click OK.

Moving SOCKS v5 Connection Entries

You may want to change the order of the connection entries in your `socks5.conf` file. You can do so by moving the connection entries.

To move a connection entry:

1. From the Server Manager, choose SOCKS | Connections. The SOCKS v5 Connections form appears.
2. Select the radio button next to the connection entry that you want to edit.
3. Click the Move button. The SOCKS v5 Move Entry form appears.
4. From the Move pull-down, choose the position in the `socks5.conf` file that you want the connection entry to be in.
5. Click OK.

Creating Routing Entries

There are two types of routing entries, the proxy routes and the SOCKS v5 routes. The proxy routes identify the IP addresses that are accessible through another SOCKS server and whether that SOCKS server connects directly to the host. Proxy routes are important when you are routing through a SOCKS server. The SOCKS v5 routes identify which interface the SOCKS deamon should use for particular IP addresses.

Creating SOCKS v5 Routing Entries

To create a SOCKS v5 route:

1. From the Server Manager, choose SOCKS | Routing. The SOCKS v5 Routing form appears.

2. Under the Routing section, click the Add button. The SOCKS v5 Routing Entry form appears.

3. In the Host mask field, enter the IP address or host name for which incoming and outgoing connections must go through the specified interface. If you enter an IP address, follow it with a forward slash and the mask to be applied to the incoming IP address. The SOCKS server will apply this mask to the IP address to determine if it is a valid host. There cannot be any spaces in the host mask entry. If you do not enter a host mask, the SOCKS v5 entry applies to all hosts.

   For example, you can enter "155.25.0.0/255.255.0.0" into the Host/Mask field. If the host's IP address is 155.25.3.5, the SOCKS server will apply the mask to the IP address and determine that the host's IP address matches the IP address for which the routing entry applies (155.25.0.0).

4. In the Port range field, enter the ports for which incoming and outgoing connections must go through the specified interface. Your port range should not have any spaces. If you do not specify a port range, the SOCKS v5 entry applies to all ports.

   You can use brackets [ ] to include the ports at each enge of the range or parentheses ( ) to exclude them. For example [1000-1010] means all port numbers between and including 1000 and 1010. (1000-1010) means all port numbers between, but not including 1000 and 1010. You can also mix brackets and parentheses. For instance, (1000-1010] means all numbers between 1000 and 1010, excluding 1000, but including 1010.

5. In the Interface/Address field, enter IP address or name of the interface through which incoming and outgoing connections must pass.

6. From the Insert pull-down, choose the position of this SOCKS v5 routing entry in your `socks5.conf` file. Because you can have multiple routing methods, you need to specify the order in which they are evaluated.

Creating Proxy Routing Entries

To create a proxy route:

1. From the Server Manager, choose SOCKS | Routing. The SOCKS v5 Routing form appears.

2. Under the Routing section, click the Add button. The SOCKS v5 Proxy Routing Entry form appears.
3. From the Proxy Type pull-down, choose the type of proxy server you will be routing through. The choices are:
    - SOCKS v5
    - SOCKS v4
    - direct connection
4. In the Destination host mask field, enter the IP address or host name for which the connection entry applies. If you enter an IP address, follow it with a forward slash and the mask to be applied to the incoming IP address. The SOCKS server will apply this mask to the IP address of the destination machine to determine if it is a valid destination host. There cannot be any spaces in the host mask entry. If you do not enter a destination host mask, the connection entry applies to all hosts.

    For example, you can enter “155.25.0.0/255.255.0.0” into the Destination host mask field. If the destination host’s IP address is 155.25.3.5, the SOCKS server will apply the mask to the IP address and determine that the destination host’s IP address matches the IP address for which the proxy entry applies (155.25.0.0).
5. In the Port range filed, enter the ports on the destination host for which the proxy entry applies. Your port range should not have any spaces. If you do not specify a port range, the proxy entry applies to all ports.

    You can use brackets [ ] to include the ports at each end of the range or parentheses ( ) to exclude them. For example [1000-1010] means all port numbers between and including 1000 and 1010. (1000-1010) means all port numbers between, but not including 1000 and 1010. You can also mix brackets and parentheses. For instance, (1000-1010] means all numbers between 1000 and 1010, excluding 1000, but including 1010.
6. In the Proxy Address field, enter the host name or IP address of the proxy server to use.
7. In the Port field, enter the port number on which the proxy server will listen for SOCKS requests.
8. From the Insert pull-down, choose the position of this routing entry in your `socks5.conf` file. Because you can have multiple routing methods, you need to specify the order in which they are evaluated.
9. Click OK.

Editing Routing Entries

To edit a proxy routing entry or a SOCKS v5 routing entry:

1. From the Server Manager, choose SOCKS | Routing. The SOCKS v5 Routing form appears.
2. Select the radio button next to the routing entry that you want to edit.
3. Click the Edit button.
4. On the form that appears, edit the appropriate information.
5. Click OK.

Deleting Routing Entries

To delete a proxy routing entry or a SOCKS v5 routing entry:

1. From the Server Manager, choose SOCKS | Routing. The SOCKS v5 Routing form appears.
2. Select the radio button next to the routing entry that you want to edit.
3. Click the Delete button.

Moving Routing Entries

You may want to change the order of the routing entries in your `socks5.conf` file. You can do so by moving the routing entries.

To move a routing entry:

1. From the Server Manager, choose SOCKS | Routing. The SOCKS v5 Routing form appears.
2. Select the radio button next to the routing entry that you want to move.
3. Click the Move button. The SOCKS v5 Move Entry form appears.
4. From the Move pull-down, choose the position in the `socks5.conf` file that you want the routing entry to be in. Because you can have multiple routing methods, you need to specify the order in which they are evaluated.
5. Click OK.

Enabling SOCKS

To enable your SOCKS server:

1. From the Server Manager, choose SOCKS | On / Off. The SOCKS On / Off form appears.
2. Click the Server On button.

Authenticating Through a SOCKS Server Chain

You can chain SOCKS servers together in the same manner that you chain proxy servers together. In other words, you can have your SOCKS server route through another SOCKS server.

To set up SOCKS server chaining:

1. From the Server Manager, choose SOCKS | Routing. The SOCKS v5 Routing form appears.
2. In the Server Chaining Section, enter your user name and password for authenticating to chained proxy servers.
3. Click OK.

Chapter 9

Caching

This chapter describes how iPlanet Web Proxy Server caches documents. It also describes how you can configure the cache by using the online forms and how the cache directory structure is maintained automatically by the Cache Monitor and Cache Manager.

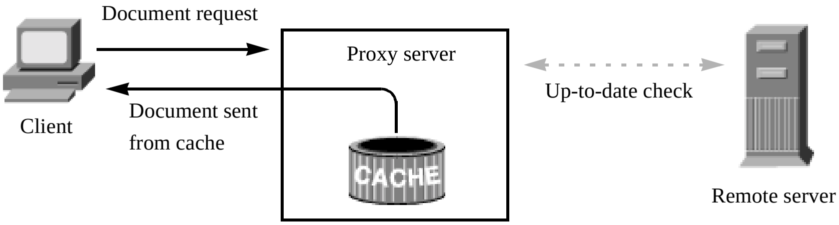
How Caching Works

Caching reduces network traffic and offers faster response time for clients who are using the proxy server instead of going directly to remote servers.

When a client requests a web page or document from the proxy server, the proxy server copies the document from the remote server to its local cache directory structure while sending the document to the client.

When a client requests a document that was previously requested and copied into the proxy cache, the proxy returns the document from the cache instead of retrieving the document from the remote server again (see Figure 9-1). If the proxy determines the file is not up to date, it refreshes the document from the remote server and updates its cache before sending it to the client.

Figure 9-1 Proxy document retrieval



Files in the cache are automatically maintained by the iPlanet Web Proxy Server Cache Manager. The Cache Manager automatically cleans the cache on a regular basis to ensure that the cache doesn’t get cluttered with out-of-date documents.

Understanding the Cache Structure

A cache consists of one or more partitions. Conceptually, a partition is a storage area on a disk that you set aside for caching. If you wish to have your cache span several disks, you need to configure at least one cache partition for each disk. Each partition can be independently administered. In other words, you can enable, disable, and configure a partition independently of all other partitions.

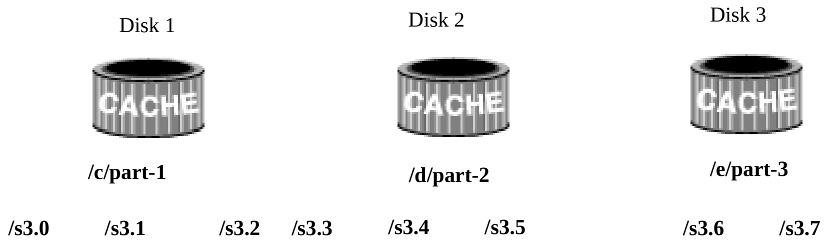
Storing a large number of cached files in a single location can slow performance; therefore, it is a good idea to create several directories, or sections, in each partition. Sections are the next level under partitions in the cache structure. You can have up to 256 sections in your cache across all partitions. The number of cache sections must be a power of 2 (for example, 1, 2, 4, 8, 16, ..., 256).

The final level in the cache structure hierarchy is the subsection. Subsections are directories within sections. If you choose to have subsections, you may have up to 256 of them, and the number of subsections must be a power of two. Cached files are stored in the lowest level in your cache.

Figure 9-2 shows an example cache structure with partitions and sections. In this figure, the cache directory structure divides the total cache into three partitions. The first partition contains four cache sections, and the second two partitions each contain two sections.

For the UNIX proxy server, each cache section is noted by s for section, and then a section number. For the section shown as s3.4, the 3 indicates the power of 2 for the number of cache sections ($2^3 = 8$), and the 4 means the number for the section (for the 8 sections labeled 0 through 7). Therefore, s3.4 means section 5 of 8.

Figure 9-2 Example of a cache structure



In summary, a cache consists of partitions. In those partitions you may have sections, and within those sections you may have subsections. Cached files are always stored in the lowest level in your cache. Therefore, if your cache has subsections within the sections, the cached files are stored in the subsections. If your cache has sections, but no subsections, the files are stored in the sections.

NOTE If you are unsure about how many cache sections and subsections to create for your cache, remember that for good cache performance, it is wise to plan for approximately 100 and no more than 500 cached files in each directory.

Distributing Files in the Cache

The proxy server uses a specific algorithm to determine the directory where a document should be stored. This algorithm ensures equal distribution of documents in the base directories, so the directories contain a small and nearly equal number of documents. Equal distribution is important because directories with large numbers of documents tend to cause performance problems.

The UNIX proxy server uses the RSA MD5 algorithm (Message Digest 5) to reduce a URL to 8 characters, which it then uses for the file name of the document it stores in the cache. The MD5 algorithm reduces the URL to 128 bits (16 bytes) of binary data. The proxy uses 48 bits (6 bytes) of this data to calculate an 8-character file name and determine the storage directory. This method allows the proxy to cache over 70 million URLs.

Setting Cache Specifics

You can enable caching and control which types of protocols your proxy server will cache by setting the cache specifics. Cache specifics include the following items:

- Whether your cache is enabled or disabled
- The cache manager's "working directory" where it stores its temporary files
- The name of the directory in which you will record the cached URLs
- The size of the cache
- The capacity of the cache
- What types of protocols will be cached
- When to refresh a cached document
- Whether the proxy should track the number of times a document is accessed and report it back to the remote server

NOTE Setting the specifics for a large cache is time-consuming and may cause the administration interface to time-out. Therefore, if you are creating a large cache, use the command line utilities to set cache specifics. For more information on the cache command line utilities, see "Using the Cache Command Line Utilities" on page 125.

To set cache specifics from the Server Manager:

1. In the Server Manager, choose Caching | Specifics.

   The Cache Specifics form appears.

2. Change the information.
3. Click OK.

The following sections describe the items listed on the Cache Specifics form. These sections include information that will help you to determine which settings will best suit your needs.

Enabling the Cache

Caching is an effective way to reduce network traffic for users of the proxy server. Caching also offers a faster response time for clients by eliminating the need to retrieve a document from a remote server. Your proxy server will function most effectively whenever caching is enabled.

You can enable the cache on the Cache Specifics form.

Creating a Cache Working Directory

If you set up caching during installation, you specified a directory for the proxy's cache structure. This directory is also used as a *working directory* for the Cache Manager. The working directory is where the proxy puts the temporary files that are related to caching. The actual cache files are under cache partitions. The working directory you specify on the Cache Specifics form is often the parent directory for the cache (though it does not need to be). All cached files appear in an organized directory structure under the caching directory. If you change the cache directory name or move it to another location, you have to tell the proxy the new location.

You can extend the cache directory structure to multiple file systems so that you can have a large cache structure divided on multiple smaller disks instead of keeping it all on one large disk. Each proxy server must have its own cache directory structure—that is, cache directories can't be concurrently shared by multiple proxy servers.

You can create the working directory on the Cache Specifics form.

Recording URLs

Your proxy server allows you to record all cached URLs in a URL list. You can identify which directory will hold all cached URL information and enable URL recording on the Cache Specifics form. For information on viewing and editing the URL list, see "Accessing Cache Manager Information" on page 122.

NOTE The proxy does not have to record URLs to function properly. This feature exists so that the proxy administrator can view which URLs are in the cache. Continually recording URLs into a list may have an impact on the proxy's performance. To avoid this negative effect on performance, you can disable URL recording on the Cache Specifics form and view or manage URLs in the cache by using the command line program: `extras/proxy/urldbgen`. This program generates the URL list on command and does not effect the proxy's performance. See "Repairing the Cache URL List" for more information about urldbgen.

Setting the Cache Size

Cache size is the maximum size the cache is allowed to grow. The maximum cache size is 64GB. The amount of disk space available for the proxy cache has a considerable effect on cache performance. If the cache is too small, the Cache Manager program must remove cached documents to make room on the disk more often, and documents must be retrieved from content servers more often; therefore slowing performance.

Large cache sizes are best because the more cached documents, the less the network traffic load and the faster the response time the proxy provides. Also, the Cache Manager removes cached documents if users no longer need them. Barring any file system limitations, cache size can never be too large; the excess space simply remains unused.

Proxy caching is designed to work efficiently at any size up to 64GB. The exact cache size you choose depends on the number of people using your proxy server. For a single user cache, 20MB to 50MB is usually enough. For a proxy that caches a multitude of documents, you might need to allocate an entire 2GB to 4GB disk partition for the cache. You can also have the cache split on multiple disk partitions. See "Adding and Modifying Cache Partitions" for more information on partitions.

You can set the cache size on the Cache Specifics form.

NOTE You might encounter problems with caching if the file system where the cache root resides has less disk space than the cache size you specify. Also, note that expanding the cache size requires a hard restart (shutdown and restart) for the changes to take effect.

CAUTION Changing the cache structure after installation requires that you reformat the structure and relocate existing files, causing any alterations to be time-consuming. If you aren’t sure what cache size to use, use 2GB as the default value in the installation forms (this default can hold more than 2GB of data and can be used with 3GB to 5GB caches).

Editing the Cache Capacity

You can edit the cache capacity through the Cache Specifics form as well as on the Cache Administration Operations form. For more information on editing the cache capacity, see “Setting the Cache Capacity” on page 120.

Caching HTTP Documents

Internally, caching HTTP documents differs from caching FTP and Gopher documents. HTTP documents offer caching features that documents of the other protocols do not. However, by setting up and configuring the cache properly, you can ensure that your proxy server will cache HTTP, FTP, and Gopher documents effectively.

All HTTP documents have a descriptive header section that the proxy server uses to compare and evaluate the document in the proxy cache and the document on the remote server. When the proxy does an up-to-date check on an HTTP document, the proxy sends one request to the server that tells the server to return the document if the version in the cache is out of date. Often, the document hasn’t changed since the last request and therefore is not transferred. This method of checking to see if an HTTP document is up-to-date saves bandwidth and decreases latency.

To reduce transactions with remote servers, the proxy server allows you to set a Cache Expiration setting for HTTP documents. The Cache Expiration setting tells the proxy to estimate if the HTTP document needs an up-to-date check before sending the request to the server. The proxy makes this estimate based on the HTTP document’s Last-Modified date found in the header.

With HTTP documents, you can also use a Cache Refresh setting. This option specifies whether the proxy always does an up-to-date check (which would override an Expiration setting) or if the proxy waits a specific period of time before doing a check. Table 9-1 shows what the proxy does if both an Expiration setting and a Refresh setting are specified. Using the Refresh setting decreases latency and saves bandwidth considerably.

Table 9-1 Using the Cache Expiration and Cache Refresh settings with HTTP

Refresh setting	Expiration setting	Results
Always do an up-to-date check	(Not applicable)	Always do an up-to-date check
User-specified interval	Use document's "expires" header	Do an up-to-date check if interval expired
	Estimate with document's Last-Modified header	Smaller value* of the estimate and expires header

* Using the smaller value guards against getting stale data from the cache for documents that change frequently.

Setting the HTTP Cache Refresh Interval

If you decide that you want your proxy server to cache HTTP documents, you need to determine whether it should always do an up-to-date check for documents in the cache or if it should check based on a Cache Refresh setting (up-to-date check interval). For HTTP documents, a reasonable refresh interval would be four to eight hours, for example. The longer the refresh interval, the fewer the number of times the proxy connects with remote servers. Even though the proxy doesn't do up-to-date checking during the refresh interval, users can force a refresh by clicking the Reload button in the client (such as Netscape Navigator); this action makes the proxy force an up-to-date check with the remote server.

You can set the refresh interval for HTTP documents on either the Cache Specifics form or the Cache Configuration form. The Cache Specifics form allows you to configure global caching procedures, and the Cache Configuration form allows you to control caching procedures for specific URLs and resources. For more information on using the Cache Specifics form, see "Setting Cache Specifics" on page 108, and for more information on using the Cache Configuration form, see "Configuring the Cache" on page 115.

Setting the HTTP Cache Expiration Policy

You can also set up your server to check if the cached document is up-to-date by using a last-modified factor or explicit expiration information only.

Explicit expiration information is a header found in some HTTP documents that specifies the date and time when that file will become outdated. Not many HTTP documents use explicit Expires headers, so it's better to estimate based on the Last-modified header.

If you decide to have your HTTP documents cached based upon the Last-modified header, you need to select a fraction to use in the expiration estimation. This fraction, known as the LM factor, is multiplied by the interval between the last modification and the time that the last up-to-date check was performed on the document. The resulting number is compared with the time since the last up-to-date check. If the number is smaller than the time interval, the document is not expired. Smaller fractions make the proxy check documents more often. For example, suppose you have a document that was last changed ten days ago. If you set the last-modified factor to 0.1, the proxy interprets the factor to mean that the document is probably going to remain unchanged for one day (10 * 0.1 = 1). The proxy would, in that case, return the document from the cache if the document was checked less than a day ago.

In this same example, if the cache refresh setting for HTTP documents is set to less than one day, the proxy does the up-to-date check more than once a day. The proxy always uses the value (cache refresh or cache expiration) that requires that it update the files more frequently.

You can set the expiration setting for HTTP documents on either the Cache Specifics form or the Cache Configuration form. The Cache Specifics form allows you to configure global caching procedures and the Cache Configuration form allows you to control caching procedures, for specific URLs and resources. For more information on using the Cache Specifics form, see “Setting Cache Specifics” on page 108, and for more information on using the Cache Configuration form, see “Configuring the Cache” on page 115.

Reporting HTTP Accesses to the Remote Server

When a document is cached by iPlanet Web Proxy Server, it can be accessed many times before it is refreshed again. For the remote server, sending one copy to the proxy that will cache it represents only one access, or “hit.” iPlanet Web Proxy Server can count how many times a given document is accessed from the proxy cache between up-to-date checks and then send that hit count back to the remote server in an additional HTTP request header (Cache-Info) the next time the document is refreshed. This way, if the remote server is configured to recognize this type of header, it receives a more accurate account of how many times a document is accessed.

You can enable HTTP access reporting on the Cache Specifics form. For more information on using the Cache Specifics form, see "Setting Cache Specifics" on page 108.

Caching FTP and Gopher Documents

FTP and Gopher do not include a method for checking to see if a document is up-to-date. Therefore, the only way to optimize caching for FTP and Gopher documents is to set a Cache Refresh interval. The Cache Refresh interval is the amount of time the proxy server waits before retrieving the latest version of the document from the remote server. If you do not set a Cache Refresh interval, the proxy will retrieve these documents even if the versions in the cache are up-to-date.

Setting FTP and Gopher Cache Refresh Intervals

If you are setting a cache refresh interval for FTP and Gopher, choose one that you consider safe for the documents the proxy gets. For example, if you store information that rarely changes, use a high number (several days). If the data changes constantly, you'll want the files to be retrieved at least every few hours. During the refresh time, you risk sending an out-of-date file to the client. If the interval is short enough (a few hours), you eliminate most of this risk while getting noticeably faster response time.

You can set the cache refresh interval for FTP and Gopher documents on either the Cache Specifics form or the Cache Configuration form. The Cache Specifics form allows you to configure global caching procedures, and the Cache Configuration form allows you to control caching procedures for specific URLs and resources. For more information on using the Cache Specifics form, see "Setting Cache Specifics" on page 108, and for more information on using the Cache Configuration form, see "Configuring the Cache" on page 115.

NOTE	If your FTP and Gopher documents vary widely (some change often, others rarely), use the Cache Configuration form to create a separate template for each kind of document (for example, create a template with resources ftp://.*.gif) and then set a refresh interval that is appropriate for that resource.

Configuring the Cache

You can configure the kind of caching you want for specific resources, using the Caching Configuration form. You can specify several configuration parameter values for URLs matching the regular expression pattern that you specify. This feature gives you fine control of the proxy cache, based on the type of document cached. Configuring the cache can include identifying the following items:

- The cache default
- How to cache pages that require authentication
- How to cache queries
- The minimum and maximum cache file sizes
- When to refresh a cached document
- The cache expiration policy
- The caching behavior for client interruptions
- The caching behavior for failed connections to origin servers

NOTE If you set the cache default for a particular resource to either Derived configuration or Don’t cache, the cache configuration options will not appear on the Caching Configuration form. However, if you choose a cache default of Cache for a resource, you can specify several other configuration items.

To configure the cache:

1. In the Server Manager, choose Caching | Configuration.

   The Caching Configuration form appears.

2. Select the resource you are editing by either choosing it from the Editing pull-down menu or by clicking the Regular Expression button, entering a regular expression, and clicking OK.

   For more information on regular expressions, see “Understanding Regular Expressions,” on page 32.

3. Change the configuration information.
4. Click OK.

The following sections describe the items listed on the Caching Configuration form. These sections include information that will help you to determine which configuration will best suit your needs.

Setting the Cache Default

The proxy server allows you to identify a cache default for specific resources. A resource is a type of file that matches certain criteria that you specify. For instance, you may want your server to automatically cache all documents from the domain company.com. If so, click the Regular Expression button on the top of the Configuration form and, in the field that appears, enter

```
[a-z] *://[^/:]\.company\.com.*.
```

Then click the Cache radio button. Your server automatically caches all cacheable documents from that domain. For more information on regular expressions, see "Understanding Regular Expressions," on page 32.

NOTE If you set the cache default for a particular resource to either Derived configuration or Don't cache, it is not necessary to configure the cache for that resource. However, if you choose a cache default of Cache for a resource, you can specify several other configuration items. For a list of these items, see "Configuring the Cache" on page 115.

You can set the cache default for any resource on the Cache Configuration form. The cache default for HTTP, FTP, and Gopher can also be set on the Cache Specifics form.

Caching Pages Retrieved Using HTTPS

You can choose to have your server cache files that are retrieved using HTTPS. Because documents that are retrieved using HTTPS are secure, they have to be encrypted by the remote server and then decrypted by the proxy before they are viewed by the client. This process can sometimes slow document retrieval. If clients frequently request a secure document through your proxy, you may want to store it in the cache. By storing the document in the cache, you avoid the encryption and decryption process, minimizing the time it takes to retrieve the document.

If you do not enable the caching of HTTPS documents, the proxy assumes the default, which is to not cache them.

You can set the policy for caching pages retrieved using HTTPS on the Cache Configuration form.

Caching Pages that Require Authentication

You can have your server cache files that require user authentication. If you choose to have your proxy server cache these files, it tags the files in the cache so that if a user asks for them, it knows that the files require authentication from the remote server.

Because the proxy server does not know how remote servers authenticate and it does not know users' IDs or passwords, it will simply force an up-to-date check with the remote server each time a request is made for a document that requires authentication. The user therefore must enter an ID and password to gain access to the file. If the user has already accessed that server earlier in the Navigator session, Navigator automatically sends the authentication information without prompting the user for it.

If you do not enable the caching of pages that require authentication, the proxy assumes the default, which is to not cache them.

You can set the policy for caching pages that require authentication on the Cache Configuration form.

Caching Queries

Cached queries only work with HTTP documents. You can limit the length of queries that are cached, or you can completely inhibit caching of queries. The longer the query, the less likely it is to be repeated, and the less useful it is to cache.

These caching restrictions apply for queries: the access method has to be GET, the document must not be protected (unless caching of authenticated pages is enabled), and the response must have at least a Last-modified header. This requires the query engine to indicate that the query result document can be cached. If the Last-modified header is present, the query engine should support a conditional GET method (with an If-modified-since header) in order to make caching effective; otherwise it should return an Expires header.

If you do not enable the caching of queries, the proxy assumes the default, which is to not cache them.

You can set the query cache policy on the Cache Configuration form.

Setting the Minimum and Maximum Cache File Sizes

You can set the minimum and maximum sizes for files cached by your proxy server. You may want to set a minimum size if you have a fast network connection. If your connection is fast, small files may be retrieved so quickly that it is not necessary for the server to cache them. In this instance, you would want to cache only larger files. You may want to set a maximum file size to make sure that large files do not occupy too much of your proxy's disk space.

You can set the minimum and maximum cache file sizes on the Cache Configuration form.

Setting the Cache Behavior for Client Interruptions

If a document is only partly retrieved and the client interrupts the data transfer, the proxy has the ability to finish retrieving the document for the purpose of caching it. The proxy's default is to finish retrieving a document for caching if at least 25 percent of it has already been retrieved. Otherwise, the proxy terminates the remote server connection and removes the partial file. You can raise or lower the client interruption percentage on the Cache Configuration form.

Setting the Cache Behavior for Failed Origin Server Connections

If an up-to-date check on a stale document fails because the origin server is unreachable, you can specify whether the proxy sends the stale document from the cache. You can specify the failure to connect to server behavior on the Cache Configuration form.

Adding and Modifying Cache Partitions

Cache partitions are reserved parts of disks or memory that are set aside for caching purposes. The largest cache capacity is 64GB with 256 cache sections. If your caching capacity changes, you may want to change or add partitions using the Cache Partition Configuration form. From this form, you can edit a partition's location, mnemonic name, and maximum and minimum sizes. You can also view the cache section table for that partition.

To add cache partitions:

1. In the Server Manager, choose Caching | Partitions.

   The Cache Partition Table appears.

2. Click the Add Cache Partition button.
3. Enter the appropriate values for the new partition.
4. Restart the proxy from the command line by going to the proxy directory and typing `./restart`.

To modify cache partitions,

1. In the Server Manager, choose Caching | Partitions.

   The Cache Partition Table appears.

2. Click on the name of the partition that you would like to change.
3. Edit the information.
4. Click Change.
5. Restart the proxy from the command line by going to the proxy directory and typing `./restart`.

Adding and Modifying Cache Sections

The proxy cache is separated into one or more cache sections. You can have up to 256 sections. The number of cache sections must be a power of two (for example, 1, 2, 4, 8, 16, ..., 256).

Each cache section can hold 100MB to 250MB of data; the optimum size is around 125MB per section. This means that if you pick a cache capacity of 500MB, the installer will create 4 cache sections (500 ÷ 125 = 4); if you choose a cache capacity of 2GB, the installer creates 16 sections (2000 ÷ 125 = 16). The smallest available capacity is 125MB with a single cache section. The largest capacity is 32GB (optimum) with 256 cache sections which can hold up to 64GB of data.

To add or modify cache sections:

1. In the Server Manager, choose Caching | Sections. The Cache Section Table appears.
2. Change the information in the table.
3. Click Make These Changes.
4. Restart the proxy from the command line by going to the proxy directory and typing `./restart`.

Setting the Cache Capacity

Cache capacity is directly related to the cache hierarchy in the cache directories. The larger the hierarchy, the bigger the capacity. The cache capacity should be equal to or greater than the cache size. Setting the capacity larger than the cache size can be helpful if you know that you plan to increase the cache size later (such as by adding an external disk).

To set the cache capacity:

1. In the Server Manager, choose Caching | Capacity.

   The Cache Administrative Operations form appears.
2. Choose a capacity from the Capacity pull-down menu.
3. Click Change Capacity.
4. Restart the proxy from the command line by going to the proxy directory and typing `./restart`.

Or

1. In the Server Manager, choose Caching | Specifics.

   The Cache Specifics form appears.
2. Click the word edit that appears next to Cache capacity.

3. Choose a capacity from the Capacity pull-down menu.
4. Click Change Capacity.
5. Restart the proxy from the command line by going to the proxy directory and typing `./restart`.

Enabling the Cache Monitor and Manager

The proxy program spawns two extra copies of itself to perform cache management. These two processes are the Cache Monitor and Cache Manager. The Cache Monitor receives data from the server process pool about cache activity and maintains information about its size and other aspects. It occasionally triggers the Cache Manager to do the actual cache clean-up tasks.

If the Cache Manager process is accidentally killed, it starts again automatically. The Cache Manager daemon uses the same configuration file as the proxy server.

You can disable the Cache Manager and Monitor if you plan to perform cache maintenance with an external program. Otherwise, the Cache Manager and Monitor should be enabled.

By accessing Cache Manager information, you can view all cached URLs, control caching for specific documents, and see an estimated size of the current cache structure. You can explicitly expire documents in the cache (so that the next time they are accessed, the proxy does an up-to-date check to determine if the document in the cache needs to be refreshed) and you can remove documents from the cache. For more information on accessing Cache Manager information, “Accessing Cache Manager Information” on page 122.

To enable or disable the Cache Monitor and Manager:

1. In the Server Manger, choose Caching | Special.

   The Special Cache Configuration form appears.

2. Click the appropriate button to either enable or disable the Cache Manager and Monitor.
3. Click OK.

Accessing Cache Manager Information

You can view the names and attributes of all cached URLs through the Cache Manager information. Cache Manager information is a list of all cached documents grouped by access protocol and site name. This list is stored in the directory that you specify on the Cache Specifics form. You can limit the URLs you view in the list by typing a domain name into the Search field. By accessing this information, you can perform various cache management functions such as expiring and removing documents from the cache.

To access cache manager information,

1. In the Server Manager, choose Caching | Cache Management.
2. Enter a DNS domain name in the Search field and click the Search button, or select a domain name from the list. A list of subdomains in that domain appears.
3. Click on the name of a subdomain. A list of the hosts in that subdomain appears.
4. Click on the name of a host. A list of all of URLs appears.
5. Click on the name of a URL. Detailed information about that URL appears.

NOTE Because continually recording URLs slows the proxy's performance, you do not have to enable URL recording to access Cache Manager information. To access this information without effecting performance, you can run the command line program: `extras/proxy/urldbgen`. This program generates a list of cached URLs on command. Once you have generated this list you can use the Cache Management form to access and manage the cache.

Caching Local Hosts

If a URL requested from a local host lacks a domain name, the proxy server will not cache it in order to avoid duplicate caching. For example, if a user requests http://machine/filename.html and http://machine.netscape.com/filename.html from a local server, both URLs might appear in the cache. Because these files are from a local server, they may be retrieved so quickly that it is not necessary to cache them anyway.

However, if your company has servers in many remote locations, you may want to cache documents from all hosts to reduce network traffic and decrease the time needed to access the files.

To enable the caching of local hosts,

1. In the Server Manager, choose Caching | Cache Local Hosts.
2. Select the resource you are editing by either choosing it from the Editing pull-down menu or by clicking the Regular Expression button and entering the name of the resource to edit.

   For more information on regular expressions, see “Understanding Regular Expressions,” on page 32.
3. Click the enabled button.
4. Click OK.

Using Cache Batch Updates

The Cache Batch Update feature allows you to pre-load files in a specified web site or do an up-to-date check on documents already in the cache whenever the proxy server is not busy. From the Cache Batch Updates form, you can create, edit, and delete batches of URLs and enable and disable batch updating.

Creating a Batch Update

You can actively (as opposed to on-demand) cache files by specifying files to be batch updated. The proxy server allows you to perform an up-to-date check on several files currently in the cache or pre-load multiple files in a particular web site.

To create a batch update:

1. In the Server Manager, choose Caching | Batch Updates.

   The Cache Batch Updates form appears.
2. Select New and Create from the pull-down menus next to “Select a configuration to edit”.
3. Click OK.

   A new Cache Batch Update form appears.
4. In the Name section of the form, enter a name for the new batch update entry.

5. In the Source section of the form, click the radio button for the type of batch update that you want to create. Click the first radio button if you want to perform an up-to-date check on all documents in the cache. Click the second radio button if you want to cache URLs recursively starting from the given source URL.

6. In the Source section fields, identify the documents that you want to use in the batch update.

7. In the Exceptions section, identify any files that you would like to exclude from the batch update.

8. In the Resources section, enter the maximum number of simultaneous connections and the maximum number of documents to traverse.

9. In the Timing section, enter the start and end times for the generation of the batch update. Only one batch update can be active at any time, so it is best to not overlap other batch update configurations.

10. Click OK.

NOTE	You can create, edit, and delete batch update configurations without having batch updates turned on. However, if you want your batch updates to be updated according to the times you set on the Cache Batch Updates form, you must turn updates on.

Editing or Deleting a Batch Update Configuration

You can edit or delete batch updates using the Cache Batch Updates form. You may want to edit a batch update if you need to exclude certain files or want to update the batch more frequently. You may also want to delete a batch update configuration completely.

To edit or delete a batch update configuration:

1. In the Server Manager, choose Caching|Batch Updates. The Cache Batch Updates form appears.

2. If you want to edit a batch, select the name of that batch and “Edit” from the pull-down menus next to “Select a configuration to edit.” If you want to delete a batch, select the name of that batch and “Delete” from the pull-down menus.

3. Click OK. The Cache Batch Updates form appears.

4. Modify the information as you wish.

5. Click OK.

Using the Cache Command Line Utilities

The proxy server comes with several command line utilities that let you configure, change, generate, and repair your cache directory structure. Most of these utilities are duplications of the Server Manager forms, but you might want to use the utilities if you need to schedule the maintenance (for example, as a cron job). All of the utilities are located in the `extras/proxy` directory. The following sections describe the various utilities.

Building the Cache Directory Structure

The utility **cbuild** creates a single directory structure for the proxy's cache. After creating the directory structure, you can use the Server Manager forms to enable the proxy to use the newly created cache.

To run the cbuild utility, at the command line, enter:

```
cbuild -d conf-dir -s user
cbuild -c cache-dir -u urldb-dir -s user
```

where:

- *conf-dir* is the directory where the proxy server instance is installed. For example, the proxy server directory could be `/usr/ns-home/proxy-`*id*. The utility determines the cache directory and location of the cache database based on the information in the directory you enter.
- *user* is the user account that the created files and directories should be owned by if running **cbuild** as root. This user ID should be the same user ID that the proxy is running as.
- *cache-dir* is the directory for your cache structure.
- *urldb-dir* is the directory where the cache management information is located.
- **cbuild** is located in the `extras/proxy` directory.

Upgrading the Cache Structure

If you have upgraded your existing 1.1 or 2.0 proxy server, you should upgrade the cache separately. Depending on the size of your cache, a cache upgrade can be a time-consuming process. You can upgrade a version 1.1 or 2.0 cache directory structure and all of its files. The **cupgrade** utility for upgrading a 1.1 structure, moves all of the files from the old directories to the new 3.5 directory structure. The **cupgrade** utility for upgrading a 2.0 cache, works in place and simply modifies the existing 2.0 cache so that it is in a 3.5 format.

NOTE The 2.5 proxy uses the same cache structure as 3.5, so you will not need to upgrade it. These instructions only apply to upgrading a 1.1 or 2.0 cache structure.

Before you upgrade a 1.1 cache structure, you must make sure you have a 3.5 structure. If you installed the proxy server by using the upgrade utility and enabling caching, then you already have a cache structure. If you don't have a cache directory structure, use the **cbuild** utility before running the **cupgrade** utility. If you are upgrading a 2.0 cache structure, you should not have a 3.5 cache. After upgrade, you should replace the 3.5 cache with the old cache.

cupgrade is located in the `extras/proxy` directory.

Upgrading a 1.1 Cache Structure

If you are using the **cupgrade** utility to upgrade a 1.1 cache structure, enter the following at the command line:

```
cupgrade -d conf-dir -o 1.1-cache-root -s user
```

conf-dir is the directory where the proxy server is installed. For example, the directory could be `/usr/ns-home/proxy-`*id*. The utility determines the new cache directory and location of the cache database based on the configuration files found in the directory you enter.

1.1-cache-root is the directory of the version 1.1 cache structure.

user is the UNIX user ID that the files in the cache should be owned as. It is optional and should be included only if you run the **cupgrade** utility as root and your proxy as another user. For example, you could run **cupgrade** as root and your proxy as nobody. In this case you would replace `<user>` with nobody.

NOTE Specifying user as nobody will not work on some systems, such as HP-UX. When using these systems, you must specify a user other than nobody for both the proxy and for **cupgrade**.

The cache upgrade can take anywhere from a few minutes to several hours depending on the size of the old cache structure.

Upgrading a 2.0 Cache Structure

If you are using the **cupgrade** utility to upgrade a 2.0 cache structure, enter the following at the command line:

`cupgrade` *sect sect ... sect*

The 2.0 upgrade should be run in the cache directory where all of the cache sections reside.

sect is a section in the cache that you want to upgrade. The number of *sect* calls depends upon how many sections are in the cache.

For example, if your cache directory is: `/usr/ns-home/cache` and you have a 1GB cache, you would then have 8 sections in your cache directory. You should type the following at the command line:

```
cd /usr/ns-home/cache
cupgrade s3.0 s3.1 s3.2 s3.3 s3.4 s3.5 s3.6 s3.7
```

Instead of typing each section, you could simply use s* to pass all of the section directory names. In this instance, you would type the following:

```
cd /usr/ns-home/cache
cupgrade s*
```

If you have multiple cache partitions you need to run an upgrade utility for each partition. For example, your cache directory may be `/usr/ns-home/cache` and you have a 2GB cache, 16 sections, and 2 partitions (with 8 sections on each partition). The partitions are `/disk1/cache-1` and `/disk2/cache-2`. The syntax for the cupgrade utility would then be:

```
cd /usr/ns-home/cache/disk1/cache-1
cupgrade s4.00 s4.01 s4.02 s4.03 s4.04 s4.05 s4.06 s4.07
```

```
cd /usr/ns-home/cache/disk2/cache-2
cupgrade s4.08 s4.09 s4.10 s4.11 s4.12 s4.13 s4.14 s4.15
```

You could also upgrade all sections on both partitions by typing the following at the command line:

```
cupgrade /disk1/cache-1/s* /disk2/cache-2/s*
```

The cache upgrade can take anywhere from a few minutes to several hours depending on the size of the old cache structure.

Repairing the Cache URL List

The proxy has a utility called **urldbgen** that goes through the entire cache directory structure and repairs the Cache Manager's URL list. Use this utility if your Cache Manager's URL list appears damaged when viewed through the Cache Management form (for example, if the URL list doesn't seem to contain all of the URLs that you know are cached or if the Cache Manager claims that the cache is empty or corrupt). You may also want to run this utility if you have disabled URL recording for the sake of performance, but want to generate a URL list on command.

The urldbgen utility is located in the `extras/proxy` directory.

You can invoke the **urldbgen** utility in one of two ways. The first way is:

`urldbgen -d` *conf-dir* `-s` *user*

conf-dir is the directory where the proxy server is installed. For example, the directory could be `/usr/ns-home/proxy-`*id*. The utility determines the cache directory and location of the cache database based on the information in the directory you enter.

user is the is the user account that the created files and directories should be owned by if running **urldbgen** as root. This user ID should be the same user ID that the proxy is running as.

The second way you can run **urldgben** is:

`urldbgen -c` *cache-dir* `-u` *urldb-dir* `-s` *user*

cache-dir is the directory for your cache structure.

urldb-dir is the directory where the cache URLs are recorded.

user is the user account that the created files and directories should be owned by if running **urldbgen** as root. This user ID should be the same user ID that the proxy is running as.

NOTE Running the URL list repair utility can take anywhere from a few seconds to a couple of hours to complete depending on the size of the cache and the speed and load of your machine and its disks.

The URL list is rarely corrupted. The only way that URL list corruption could occur is if something prevents the proxy from updating its URL list after it has completed writing a file to the cache. This could happen if the disk is full, if the proxy users' permissions prevent the proxy from writing to the list file, or if the system suddenly goes down. The URL list is located in the `urldb` directory under the cache root directory.This utility can recreate the entire URL list from scratch if it is accidentally deleted.

Cleaning the URL List

The proxy server has a command line utility called **urldbgc** that goes through the URL database and purges any old files. It is good to run this utility if, for some reason, the database is out of sync with the actual files in the cache. You can run this utility as a cron job and schedule it for the lowest peak time for your proxy server.

To clean the URL list using the **urldbgc** utility, type the following at the command line:

```
urldbgc -d conf-dir -s user
urldbgc -c cache-dir -u urldb-dir -s user
```

conf-dir is the directory where the proxy server is installed. For example, the directory could be `/usr/ns-home/proxy-`*id*. The utility determines the cache directory and location of the cache database based on the directory you enter.

user is the user account that the created files and directories should be owned by if running **urldbgc** as root. This user ID should be the same user ID that the proxy is running as.

cache-dir is the directory for your cache structure.

urldb-dir is the directory where the cache URL database is kept

NOTE If you do not want to garbage collect, but you want to fully delete all of the files in your cache, type the following at the command line:

```
cd proxy directory/cache
find s* -type f -exec rm {} \.;
```

where *proxy directory* is the directory where your proxy is stored.

Routing through Proxy Arrays

Proxy arrays for distributed caching allow multiple proxies to serve as a single cache. In other words, each proxy in the array will contain different cached URLs that can be retrieved by a browser or downstream proxy server. Proxy arrays prevent the duplication of caches that often occurs with multiple proxy servers. Through hash-based routing, proxy arrays route requests to the correct cache in the proxy array.

Proxy arrays also allow incremental scalability. In other words, if you decide to add another proxy to your proxy array, each member's cache is not invalidated. Only 1/*n* of the URLs in each member's cache, where *n* is the number of proxies in your array, will be reassigned to other members.

For each request through a proxy array, a hash function assigns each proxy in the array a score that is based on the requested URL, the proxy's name and the proxy's load factor. The request is then routed to the proxy with the highest score.

Since requests for URLs can come from both clients and proxies, there are two types of routing through proxy arrays: *client to proxy routing* and *proxy to proxy routing*.

In client to proxy routing, the client uses the Proxy Auto Configuration (PAC) mechanism to determine which proxy to go through. However, instead of using the standard PAC file, the client uses a special PAC file which computes the hash algorithm to determine the appropriate route for the requested URL. Figure 9-3 shows client to proxy routing. For more information about the PAC file, see Chapter 11, "Using the Client Autoconfiguration File." The proxy server can automatically generate the special PAC file from the Proxy Array Membership Table (PAT) specifications made through the administration interface.

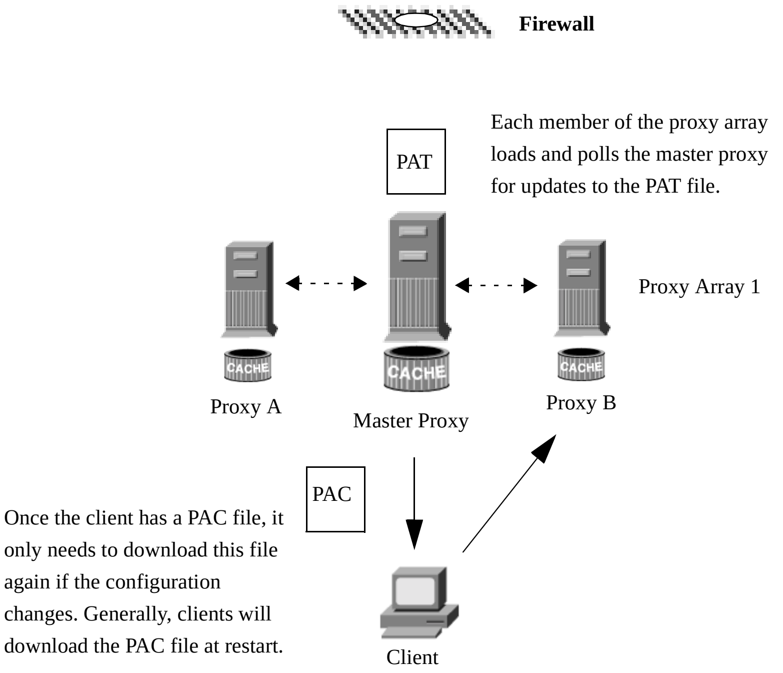
In proxy to proxy routing, proxies use a PAT (Proxy Array Table) file to compute the hash algorithm instead of the PAC file used by clients. The PAT file is an ASCII file that contains information about a proxy array, including the proxies' machine names, IP addresses, ports, load factors, cache sizes, etc. For computing the hash algorithm at the server, it is much more efficient to use a PAT file than a PAC file (which is a JavaScript file that has to be interpreted at run-time). However, most clients do not recognize the PAT file format, and therefore, must use a PAC file. Figure 9-4 shows proxy to proxy routing.

The PAT file will be created on one proxy in the proxy array - the master proxy. The proxy administrator must determine which proxy will be the master proxy. The administrator can change the PAT file from this master proxy server and all other members of the proxy array can then manually or automatically poll the master proxy for these changes. You can configure each member to automatically generate a PAC file from these changes.

You can also chain proxy arrays together for hierarchical routing. If a proxy server routes an incoming request through an upstream proxy array, the upstream proxy array is then known as a parent array. A parent array is a proxy array that a proxy server goes through. In other words, if a client requests a document from Proxy X, and Proxy X does not have the document, it sends the request to Proxy Array Y instead of sending it directly to the remote server. So, Proxy Array Y is a parent array. In Figure 9-4, Proxy Array 1 is a parent array to Proxy Array 2.

All of the proxy servers in a proxy array should be in a single administrative domain. Two proxy arrays in separate administrative domains can communicate, however if the requesting proxy can retrieve cached URLs from more than one proxy array, ICP should be used to determine which array to go to.

Figure 9-3 Client to Proxy Routing

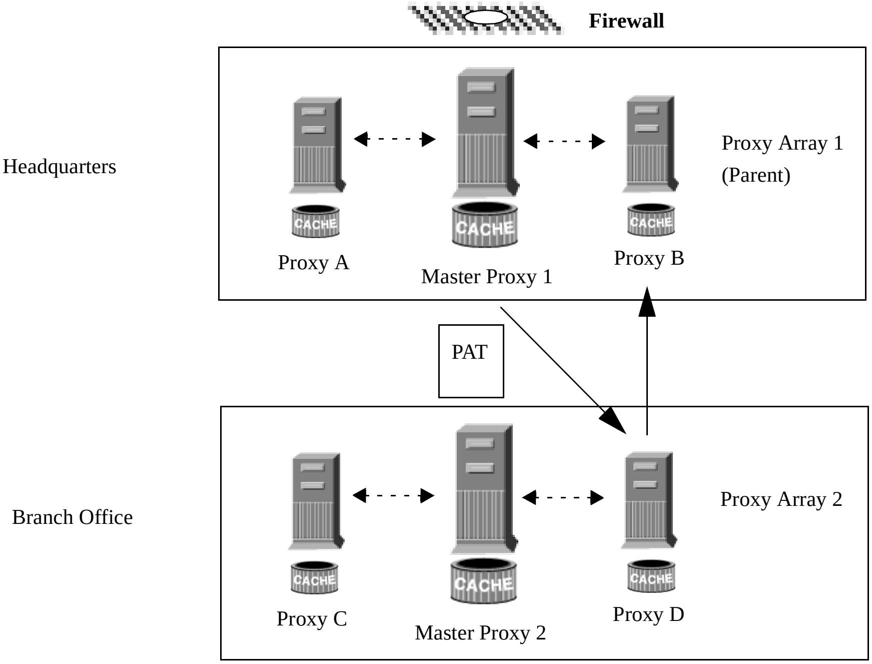


Each member of the proxy array loads and polls the master proxy for updates to the PAT file.

Once the client has a PAC file, it only needs to download this file again if the configuration changes. Generally, clients will download the PAC file at restart.

If the client does not already have a current copy of the special PAC file, it downloads the file from a member of the proxy array (usually the master proxy). The hash algorithm for the requested URL is computed for each proxy in the array using the PAC file and the client then retrieves the requested URL from whichever proxy has the highest score. In this diagram, Proxy B has the highest score for the URL requested by the client.

Figure 9-4 Proxy to Proxy Routing



A member of Proxy Array 2 loads and polls for updates to the parent array's PAT file. Usually, it polls the master proxy in the parent array. The hash algorithm for the requested URL is computed using the downloaded PAT file and the member in Proxy Array 2 then retrieves the requested URL from whichever proxy in Proxy Array 1 has the highest score. In this diagram, Proxy B has the highest score for the URL requested by the client.

To set up a proxy array:

1. From the master proxy, create the member list. For more information on creating the member list, see "Creating a Proxy Array Member List" on page 133.
2. From the master proxy, create a PAT mapping to map the URL "/pat" to the PAT file. For information on creating a PAT mapping, see "Proxying and Routing URLs," on page 63.
3. Configure each non-master member of the array. For more information on configuring non-master members, see "Configuring Proxy Array Members" on page 135.

4. Enable routing through a proxy array. For more information on enabling routing through a proxy array, see “Enabling Routing Through a Proxy Array” on page 136.
5. Enable your proxy array. For more information on enabling a proxy array, see “Enabling a Proxy Array” on page 137.
6. Generate a PAC file from your PAT file. You only need to generate a PAC file if you are using client to proxy routing. For more information on generating a PAC file from a PAT file, see “Generating a PAC File from a PAT File” on page 138.

NOTE If your proxy array is going to route through a parent array, you also need to enable the parent array and configure each member to route through a parent array for desired URLs. For more information on parent arrays, see “Routing Through a Parent Array” on page 140.

Creating a Proxy Array Member List

You should create and update the proxy array member list from the master proxy of the array only. You only need to create the proxy array member list once, but you can modify it at any time. By creating the proxy array member list, you are generating the PAT file to be distributed to all of the proxies in the array and to any downstream proxies.

CAUTION You should only make changes or additions to the proxy array member list through the master proxy in the array. All other members of the array can only read the member list.

1. From the Server Manager, choose Caching | Proxy Array Configuration. The Proxy Array Configuration form appears.
2. In the Array name field, enter the name of the array.
3. In the “Reload Configuration Every” field, enter the number of minutes between each polling for the PAT file.
4. Click OK.

NOTE Be sure to click OK before you begin to add members to the member list.

5. Click the Add button. The Proxy Array Member form appears.
6. For each member in the proxy array, enter the following and then click OK:
   - Name - the name of the proxy server you are adding to the member list
   - IP Address - the IP address of the proxy server you are adding to the member list
   - Port - This is the port on which the member polls for the PAT file.
   - Load Factor - an integer that reflects the relative load that should be routed through the member.
   - Status - the status of the member. This value can be either on or off. If you disable a proxy array member, the member's requests will be re-routed through another member.

NOTE Be sure to click OK after you enter the information for each proxy array member you are adding.

Deleting Proxy Array Members

Deleting proxy array members will remove them from the proxy array. You can only delete proxy array members from the master proxy.

CAUTION You should only make changes or additions to the proxy array member list through the master proxy in the array. If you modify this list from any other member of the array, all changes will be lost.

To delete members of a proxy array:

1. From the Server Manager, choose Caching | Proxy Array Configuration. The Proxy Array Configuration form appears.
2. In the Member List, select the radio button next to the member that you want to delete.
3. Click the Delete Button.

NOTE If you want your changes to take effect and to be distributed to the members of the proxy array, you need to update the Configuration ID on the Proxy Array Configuration form and click OK. To update the configuration ID, you can simply increase it by one.

Editing Proxy Array Member List Information

At any time, you can change the information for the members in the proxy array member list. You can only edit the proxy array member list from the master proxy.

CAUTION You should only make changes or additions to the proxy array member list through the master proxy in the array. If you modify this list from any other member of the array, all changes will be lost.

To edit member list information for any of the members in a proxy array:

1. From the Server Manager, choose Caching | Proxy Array Configuration. The Proxy Array Configuration form appears.
2. In the Member List, select the radio button next to the member that you want to edit.
3. Click the Edit Button. The Proxy Array Member form appears.
4. Edit the appropriate information.
5. Click OK.

NOTE If you want your changes to take effect and to be distributed to the members of the proxy array, you need to update the Configuration ID on the Proxy Array Configuration form and click OK. T update the configuration ID, you can simply increase it by one.

Configuring Proxy Array Members

You only need to configure each member in the proxy array once, and you must do so from the member itself. You cannot configure a member of the array from another member. You also need to configure the master proxy.

You should follow this process for each member of the array:

1. From the Server Manager, choose Caching | Member Configuration. The Proxy Array Member Configuration form appears.
2. In the Proxy Array section, indicate whether or not the member needs to poll for the PAT file by selecting the appropriate radio button. The choices are:
   - Non-master member - You should select this option if the member you are configuring is *not* the master proxy. Any proxy array member that is not a master proxy will need to poll for the PAT file in order to retrieve it from the master proxy.
   - Master member - You should select this option if you are configuring the master proxy. If you are configuring the master proxy, the PAT file is local and does not need to be polled.
3. If, in Step 2, you chose "Don't Poll", Click OK, you are finished with this form. If you chose "Poll for PAT file", continue with Step 4.
4. In the Poll Host field, enter the name of the master proxy that you will be polling for the PAT file.
5. In the Port field, enter the port at which the master proxy accepts HTTP requests.
6. In the URL field, enter the URL of the PAT file on the master proxy. If on your master proxy, you have created a PAT mapping to map the PAT file to the URL "/pat," you should enter "`/pat`" into this URL field.
7. In the Headers File field, enter the full pathname for a file with any special headers that must be sent with the HTTP request for the PAT file (such as authentication information). This field is optional.
8. Click OK.

Enabling Routing Through a Proxy Array

To enable routing through a proxy array:

1. From the Server Manager, choose Routing | Routing. The Routing Configuration form appears.
2. Select the resource you want to route by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.

3. Select the radio button next to the text “Route through”.

4. Select the checkboxes for proxy array and / or parent array.

NOTE You can only enable proxy array routing if the proxy server you are configuring is a member of a proxy array. You can only enable parent routing if a parent array exists. Both routing options are independent of eachother.

5. If you choose to route through a proxy array and you want to redirect requests to another URL, select the redirect checkbox. Redirecting means that if a member of a proxy array receives a request that it should not service, it tells the client which proxy to contact for that request.

CAUTION Redirect is not currently supported by any clients, so you should not use the feature at this time.

6. Click OK.

Enabling a Proxy Array

To enable a proxy array:

1. From the Server Manager, choose Server Preferences | System Specifics. The System Specifics form appears.

2. Select the Yes radio button for the type of array or arrays you want to enable - either a normal proxy array or a parent array.

NOTE If you are not routing through a proxy array, you should make sure that all clients use a special PAC file to route correctly before you disable the proxy array option. If you disable the parent array option, you should have valid alternative routing options set in the Routing form, such as explicit proxy or a direct connection.

3. Click OK.

Redirecting Requests in a Proxy Array

If you choose to route through a proxy array, you need to designate whether you want to redirect requests to another URL. Redirecting means that if a member of a proxy array receives a request that it should not service, it tells the client which proxy to contact for that request.

CAUTION Redirect is not currently supported by any clients, so you should not use the feature at this time.

Generating a PAC File from a PAT File

Because most clients do not recognize the PAT file format, the clients in client to proxy routing use the Proxy Auto Configuration (PAC) mechanism to receive information about which proxy to go through. However, instead of using the standard PAC file, the client uses a special PAC file derived from the PAT file. This special PAC file computes the hash algorithm to determine the appropriate route for the requested URL.

You can manually or automatically generate a PAC file from the PAT file. If you manually generate the PAC file from a specific member of the proxy array, that member will immediately re-generate the PAC file based on the information currently in the PAT file. If you configure a proxy array member to automatically generate a PAC file, the member will automatically re-generate the file after each time it detects a modified version of the PAT file.

NOTE If you are not using the proxy array feature for your proxy server, then you should use the Proxy Client Autoconfiguration form to generate your PAC file. For more information see Chapter 11, "Using the Client Autoconfiguration File."

Manually Generating a PAC File from a PAT File

NOTE The PAC file can be generated only from the master proxy.

To manually generate a PAC file from a PAT file:

1. From the Server Manager of the master proxy, choose Caching | Proxy Array Configuration. The Proxy Array Configuration form appears.

2. Click the Generate PAC button.

   The PAC Generation form appears.

3. If you want to use custom logic in your PAC file, in the Custom Logic File field, enter the name of the file containing the customized logic you would like to include in the generation of your PAC file. This logic is inserted before the proxy array selection logic in the FindProxyForURL function. This function is typically used for local requests which need not go through the proxy array.

   If you have already entered the custom logic file on the Member Configuration form, this field will be populated with that information. You may edit the custom logic filename if you wish, and the changes you make will transfer to the Member Configuration form as well.

4. In the Default Route field, enter the route a client should take if the proxies in the array are not available.

   If you have already entered the default route on the Member Configuration form, this field will be populated with that information. You may edit the default route if you wish, and the changes you make will transfer to the Member Configuration form as well.

5. Click OK.

Automatically Generating a PAC File from a PAT File

To automatically generate a PAC file from a PAT file each time a change is detected:

1. From the Server Manager, choose Caching | Member Configuration. The Member Configuration form appears.

2. Select the checkbox next to “Auto-generate PAC file”.

3. In the Default Route field, enter the route a client should take if the proxies in the array are not available.

   If you have already entered and saved the default route on the Member Configuration form, this field will be populated with that information. You may edit the default route if you wish, and the changes you make will transfer to the Member Configuration form as well.

4. If you want to use custom logic in your PAC file, in the Custom Logic File field, enter the name of the file containing the customized logic you would like to include in the generation of your PAC file. This logic is inserted before the proxy array selection logic in the FindProxyFor URL function.

   If you have already entered and saved the custom logic file on the Member Configuration form, this field will be populated with that information. You may edit the custom logic filename if you wish, and the changes you make will transfer to the Member Configuration form as well.

5. Click OK.

Routing Through a Parent Array

You can configure your proxy or proxy array to route through an upstream parent array instead of going directly to a remote server. To configure a proxy or proxy array member to route through a parent array,

1. Enable the parent array. For instructions on enabling an array, see "Enabling a Proxy Array" on page 137.
2. Enable routing through the parent array. For instructions on enabling routing through an array, see "Enabling Routing Through a Proxy Array" on page 136.
3. From the Server Manager, choose Caching | Member Configuration. The Proxy Array Member Configuration form appears.
4. In the Poll Host field in the Parent Array section of the form, enter the host name of the proxy in the parent array that you will poll for the PAT file. This proxy is usually the master proxy of the parent array.
5. In the Port field in the Parent Array section of the form, enter the Port number of the proxy in the parent array that you will poll for the PAT file.
6. In the URL field, enter the URL of the PAT file to be polled.
7. In the URL field, enter the URL of the PAT file on the master proxy. If on your master proxy, you have created a PAT mapping, you should enter the mapping into this URL field.
8. In the Headers File field in the Parent Array section of the form, full pathname for a file with any special headers that must be sent with the HTTP request for the PAT file (such as authentication information). This field is optional.
9. Click OK.

Viewing Parent Array Information

If your proxy array is routing through a parent array, you need information about the members of the parent array. This information is sent from the parent array in the form of a PAT file. The information in this PAT file is displayed on the Parent Array Configuration form.

To view parent array information,

1. From the Server Manager, choose Caching | Parent Array Configuration. The Parent Array Configuration form appears.

2. View the information.

Routing Through ICP Neighborhoods

The Internet Cache Protocol (ICP) is an object location protocol that enables caches to communicate with one another. Caches can use ICP to send queries and replies about the existence of cached URLs and about the best locations from which to retrieve those URLs. In a typical ICP exchange, one cache will send an ICP query about a particular URL to all neighboring caches. Those caches will then send back ICP replies that indicate whether or not they contain that URL. If they do not contain the URL, they send back a “MISS.” If they do contain the URL, they send back a “HIT.”

ICP can be used for communication among proxies located in different administrative domains. It allows a proxy cache in one administrative domain to communicate with a proxy cache in another administrative domain. It is effective for situations in which several proxy servers want to communicate, but cannot all be configured from one master proxy (as they are in a proxy array). Figure 9-5 shows an ICP exchange between proxies in different administrative domains.

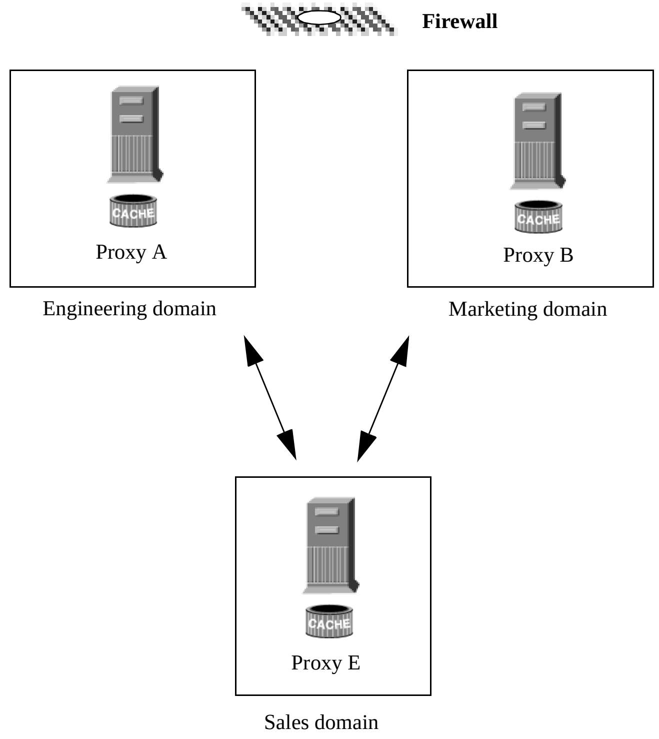
The proxies that communicate with each other via ICP are called *neighbors*. You cannot have more than 64 neighbors in an ICP neighborhood. There are two types of neighbors in an ICP neighborhood, *parents* and *siblings*. Only parents can access the remote server if no other neighbors have the requested URL. Your ICP neighborhood can have no parents or it can have more than one parent. Any neighbor in an ICP neighborhood that is *not* a parent is considered a sibling. Siblings cannot retrieve documents from remote servers unless the sibling is marked as the default route for ICP, and ICP uses the default.

You can use *polling rounds* to determine the order in which neighbors receive queries. A polling round is an ICP query cycle. For each neighbor, you must assign a polling round. If you configure all neighbors to be in polling round one, then all neighbors will be queried in one cycle. In other words, they will all be queried at the same time. If you configure some of the neighbors to be in polling round 2, then all of the neighbors in polling round one will be queried first and if none of them return a “HIT,” all round two proxies will be queried. The maximum number of polling rounds is two.

Since ICP parents are likely to be network bottlenecks, you can use polling rounds to lighten their load. A common setup is to configure all siblings to be in polling round one and all parents to be in polling round two. That way, when the local proxy requests a URL, the request goes to all of the siblings in the neighborhood first. If none of the siblings have the requested URL, the request goes to the parent. If the parent does not have the URL, it will retrieve it from a remote server.

Each neighbor in an ICP neighborhood must have at least one ICP server running. If a neighbor does not have an ICP server running, it cannot answer the ICP requests from their neighbors. Enabling ICP on your proxy server starts the ICP server if it is not already running.

Figure 9-5 An ICP exchange



Proxy E sends an ICP query for a URL to the proxies in the Marketing domain and the Engineering domain. The proxies in the Engineering and Marketing domains then send ICP replies back to Proxy E to indicate whether they contain the requested URL in their caches.

To set up ICP, follow these steps:

1. Add parent(s) to your ICP neighborhood. (This step is only necessary if you want parents in your ICP neighborhood.) For more information on adding parents to an ICP neighborhood, see “Adding Parents to an ICP Neighborhood” on page 144.
2. Add sibling(s) to your ICP neighborhood. For more information on adding siblings to your ICP neighborhood, see “Adding Siblings to an ICP Neighborhood” on page 146.

3. Configure each neighbor in the ICP neighborhood. For more information on configuring ICP neighbors, see “Configuring Individual ICP Neighbors” on page 147.

4. Enable ICP. For information on enabling ICP, see “Enabling ICP” on page 149.

5. If your proxy has siblings or parents in its ICP neighborhood, enable routing through an ICP neighborhood. For more information on enabling routing through an ICP neighborhood, see “Enabling Routing Through an ICP Neighborhood” on page 149.

Adding Parents to an ICP Neighborhood

To add parent proxies to an ICP neighborhood:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.

2. In the Parent List section of the form, click the Add Parent button. The ICP Parent form appears.

3. In the Machine Address field, enter the IP address or host name of the parent proxy you are adding to the ICP neighborhood.

4. In the ICP Port field, enter the port number on which the parent proxy will listen for ICP messages.

5. In the Multicast Address field, you can enter the multicast address to which the parent listens. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors who are listening to that multicast address can see; therefore, eliminating the need to send a query to each neighbor separately. Using multicast is optional.

NOTE Neighbors in different polling rounds should not listen to the same multicast address.

6. In the TTL field, enter the number of subnets that the multicast message will be forwarded to. If the TTL is set to 1, the multicast message will only be forwarded to the local subnet. If the TTL is 2, the message will go to all subnets that are one level away, and so on.

NOTE Multicast makes it possible for two unrelated neighbors to send ICP messages to eachother. Therefore, if you want to prevent unrelated neighbors from receiving ICP messages from the proxies in your ICP neighborhood, you should set a low TTL value in the TTL field.

7. In the Proxy Port field, enter the port for the proxy server on the parent.
8. From the Polling Round pull-down, choose the polling round that you want the parent to be in. The default polling round is 1. For more information on polling rounds see page 141.
9. Click OK.

Removing Parents from an ICP Neighborhood

To remove parent proxies from an ICP neighborhood:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.
2. Click the radio button next to the parent you want to remove.
3. Click the Remove button.

Editing Configurations for Parents in an ICP neighborhood

To edit the machine address, port number, multicast address, time to live value, proxy port number, or polling round value for a parent proxy:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.
2. Click the radio button next to the parent you want to edit.
3. Click the Edit button.
4. Modify the appropriate information.
5. Click OK.

Adding Siblings to an ICP Neighborhood

To add sibling proxies to an ICP neighborhood:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.
2. In the Sibling List section of the form, click the Add Sibling button. The ICP Sibling form appears.
3. In the Machine Address field, enter the IP address or host name of the sibling proxy you are adding to the ICP neighborhood.
4. In the Port field, enter the port number on which the sibling proxy will listen for ICP messages.
5. In the Multicast Address field, enter the multicast address to which the sibling listens. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors who are listening to that multicast address can see; therefore, eliminating the need to send a query to each neighbor separately.

NOTE	Neighbors in different polling rounds should not listen to the same multicast address.

6. In the TTL field, enter the number of subnets that the multicast message will be forwarded to. If the TTL is set to 1, the multicast message will only be forwarded to the local subnet. If the TTL is 2, the message will go to all subnets that are one level away.

NOTE	Multicast makes it possible for two unrelated neighbors to send ICP messages to eachother. Therefore, if you want to prevent unrelated neighbors from receiving ICP messages from the proxies in your ICP neighborhood, you should set a low TTL value in the TTL field.

7. In the Proxy Port field, enter the port for the proxy server on the sibling.
8. From the Polling Round pull-down, choose the polling round that you want the sibling to be in. The default polling round is 1. For more information on polling rounds see page 141.
9. Click OK.

Removing Siblings from an ICP Neighborhood

To remove sibling proxies from an ICP neighborhood:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.
2. Click the radio button next to the sibling you want to remove.
3. Click the Remove button.

Editing Configurations for Siblings in an ICP Neighborhood

To edit the machine address, port number, multicast address, time to live value, proxy port number, or polling round value for a sibling proxy:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.
2. Click the radio button next to the sibling you want to edit.
3. Click the Edit button.
4. Modify the appropriate information.
5. Click OK.

Configuring Individual ICP Neighbors

You need to configure each neighbor, or local proxy, in your ICP neighborhood.

To configure the local proxy server in your ICP neighborhood:

1. From the Server Manager, choose Caching | ICP. The ICP Configuration form appears.
2. In the Binding Address field, enter the IP address to which the neighbor server will bind.
3. In the Port field, enter the port number to which the neighbor server will listen for ICP.

4. In the Multicast Address field, enter the multicast address to which the neighbor listens. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors who are listening to that multicast address can see; therefore, eliminating the need to send a query to each neighbor separately.

   If both a multicast address and bind address are specified for the neighbor, the neighbor uses the bind address to send replies and uses multicast to listen. If neither a bind address or a multicast address is specified, the operating system will decide which address to use to send the data.

5. In the Default Route field, enter the name or IP address of the proxy to which the neighbor should route a request when none of the neighboring proxies respond with a "hit." If you enter the word "origin" into this field, or if you leave it blank, the default route will be to the origin server.

NOTE If you choose "first responding parent" from the No Hit Behavior pull-down discussed in Step 7, the route you enter in the Default Route field will have no effect. The proxy only uses this route if you choose the default no hit behavior.

6. In the second Port field, enter the port number of the default route machine that you entered into the Default Route field.

7. From the "On no hits, route through" pull-down, choose the neighbor's behavior when none of the siblings in the ICP neighborhood have the requested URL in their caches. You can choose:

   ❍ first responding parent - the neighbor will retrieve the requested URL through the parent that first responds with a "miss"

   ❍ default - the neighbor will retrieve the requested URL through the machine specified in the Default Route field.

8. In the Server Count field, enter the number of processes that will service ICP requests.

9. In the Timeout field, enter the maximum amount of time the neighbor will wait for an ICP response in each round.

10. Click OK.

Enabling ICP

To enable ICP:

1. From the Server Manager, choose Server Preferences | System Specifics. The System Specifics form appears.
2. Select the Yes radio button for ICP.
3. Click OK.

Enabling Routing Through an ICP Neighborhood

To enable routing through an ICP neighborhood:

1. From the Server Manager, choose Routing | Routing. The Routing Configuration form appears.
2. Select the resource you want to route by either choosing it from the Editing pull-down menu or clicking the Regular Expression button, entering a regular expression, and clicking OK.
3. Select the radio button next to the text "Route through."
4. Select the checkbox next to ICP.
5. If you want the client to retrieve a document directly from the ICP neighbor that has the document instead of going through another neighbor to get it, select the checkbox next to the text "redirect."

CAUTION Redirect is not currently supported by any clients, so don't use the feature at this time.

6. Click OK.

NOTE You need to enable routing through an ICP neighborhood only if your proxy has other siblings or parents in the ICP neighborhood. If your proxy is a parent to another proxy and does not have any siblings or parents of its own, then you need to enable ICP only for that proxy. You do not need to enable routing through an ICP neighborhood.

Chapter 10

Filtering Content Through the Proxy

This chapter describes how to filter URLs so that your proxy server either doesn't allow access to the URL or modifies the HTML and JavaScript content it returns to the client. This chapter also describes how you can restrict access through the proxy based on the web browser (user agent) that the client is using.

The proxy server lets you use a URL filter file to determine which URLs the server supports. For example, instead of manually typing in wildcard patterns of URLs to support, you can create or purchase one text file that contains URLs you want to restrict. This feature lets you create one file of URLs that you can use on many different proxy servers.

You can also filter URLs based on their MIME type. For example, you might allow the proxy to cache and send HTML and GIF files but not allow it to get binary or executable files because of the risk of computer viruses.

Filtering URLs

You can use a file of URLs to configure what content the proxy server retrieves. You can set up a list of URLs the proxy always supports and a list of URLs the proxy never supports.

For example, if you're an Internet service provider who runs a proxy server with content appropriate for children, you might set up a list of URLs that are approved for viewing by children. You can then have the proxy server retrieve only the approved URLs; if a client tries to go to an unsupported URL, either you can have the proxy return the default "Forbidden" message or you can create a custom message explaining why the client could not access that URL.

To restrict access based on URLs, you need to create a file of URLs to allow or restrict. You can do this through the Server Manager. Once you have the file, you can set up the restrictions. These processes are discussed in the following sections.

Creating a Filter File of URLs

A *filter file* is a file that contains a list of URLs. The filter files the proxy server uses are plain text files with lines of URLs in the following pattern:

protocol`://`*host*`:`*port*`/`*path*`/`*filename*

You can use regular expressions in each of the three sections: protocol, host:port, and path/filename. For example, if you want to create a URL pattern for all protocols going to the netscape.com domain, you'd have the following line in your file:

```
.*://.*\.netscape\.com/.*
```

This line works only if you don't specify a port number. For more information on regular expressions, see "Understanding Regular Expressions" on page 32.

NOTE	When these regular expression patterns get written to the `obj.conf` file as ppath parameters, back slash characters are replaced by double back slashes. For example, the above pattern, when written in the `obj.conf` file, would appear as: `.*://.*\\.netscape\\.com/.*`

You can use the Server Manager forms to create a file. If you want to create your own file without using the Server Manager, you should use the Server Manager forms to create an empty file, and then add your text in that file or replace the file with one containing the regular expressions.

To create a file using the Server Manager:

1. In the Server Manager, choose Filters | URL Filters. In the URL Filter Access Restriction form that appears, choose New Filter from the drop-down list next to the Create/Edit URL Filter button.
2. Type a name for the filter file in the text box to the right of the drop-down list and then click the Create/Edit URL Filter button.
3. The Filter Editor form appears. Use the Filter Content scrollable text box to enter URLs and regular expressions of URLs. The Reset button clears all the text in this field.

   For more information on regular expressions, see "Understanding Regular Expressions" on page 32.

4. When finished, click OK and confirm your changes.

   The proxy server creates the file and returns you to the URL Filter Access Restriction form. The filter file is created in the *server-root*/`admin-serv/proxy-`*id*.

Setting Default Access for a Filter File

Once you have a filter file that contains the URLs you want to use, you can set the default access for those URLs.

To set default access for a filter file:

1. In the Server Manager, choose Filters | URL Filters.
2. Choose the template you want to use with the filters.

   Typically, you'll want to create filter files for the entire proxy server, but you might want one set of filter files for HTTP and another for FTP.
3. Use the URL filter to allow list to choose a filter file that contains the URLs you want the proxy server to support.
4. Use the URL filter to deny list to choose a filter file that contains the URLs to which you want the proxy server to deny access.
5. Choose the text you want the proxy server to return to clients who request a denied URL. You can choose one of two options:
   - You can send the default "Forbidden" message that the proxy generates.
   - You can send a text or HTML file with customized text. Type the absolute path to this file using the text box on the form.

Restricting Access to Specific Web Browsers

You can restrict access to the proxy server based on the type and version of the client's web browser. For example, you can specify that all proxy server users must use Netscape Navigator 3.0. Restriction occurs based on the user-agent header that all web browsers send to servers when making requests.

To restrict access to the proxy based on the client's web browser:

1. In the Server Manager, choose Filter | User-Agent.
2. Check the allow only User-agents matching radio button.

3. Type a regular expression that matches the user-agent string for the browsers you want the proxy server to support. If you want to specify more than one client, enclose the regular expression in parentheses and use the | character to separate the multiple entries. For more information on regular expressions, see "Understanding Regular Expressions" on page 32.

Request Blocking

You may want to block file uploads and other requests based on the upload content type.

To block requests based on MIME type:

1. From the Server Manager, choose Filters | Request Blocking. The Request Blocking form appears.
2. Click the radio button for the type of request blocking you want. The options are:
   - disabled - disables request blocking
   - multipart MIME (file upload) - blocks all file uploads
   - MIME types matching regular expression - blocks requests for MIME types that match the regular expression you enter. For more information on creating regular expressions, see "Understanding Regular Expressions" on page 32.
3. Choose whether you want to block requests for all clients or for user-agents that match a regular expression you enter.
4. Click the radio button for the methods for which you want to block requests. The options are:
   - any method with request body - blocks all requests with a request body, regardless of the method
   - only for:
     POST - blocks file upload requests using the POST method
     PUT - blocks file upload requests using the PUT method
   - methods matching - blocks all file upload requests using the method you enter
5. Click OK.

Suppressing Outgoing Headers

You can configure the proxy server to remove outgoing headers from the request (usually for security reasons). For example, you might want to prevent the “from” header from going out because it reveals the user’s email address (although Netscape Navigator does not send the “from” header unless specifically configured to do so). Or, you might want to filter out the user-agent header so external servers can’t determine what web browsers your organization uses. You may also want to remove logging or client-related headers that are to be used only in your intranet before a request is forwarded to the Internet.

This feature doesn’t affect headers that are specially handled or generated by the proxy itself or that are necessary to make the protocol work properly (such as If-Modified-Since and Forwarded).

Although it’s not possible to stop the forwarded header from originating from a proxy, this isn’t a security problem. The remote server can detect the connecting proxy host from the connection. In a proxy chain, a forwarded header coming from an inner proxy can be suppressed by an outer proxy. Setting your servers up this way is recommended when you don’t want to have the inner proxy or client host name revealed to the remote server.

To suppress outgoing headers:

1. In the Server Manager, choose Filters | Suppress Outgoing Headers.
2. In the form that appears, type a regular expression that matches the headers you want to suppress. For example, to suppress the from and user-agent headers, type `(from|user-agent)`. The headers you type are not case-sensitive. For more information on regular expressions, see “Understanding Regular Expressions” on page 32.

Appending Customized Outgoing Headers

You can configure the proxy server to add your customized headers to the list of headers in the request being forwarded to the Web Server. For example, you might want to understand from the header where a certain request is coming from.

To add a customized header, use the Server Access Function `append-header,` making sure to specify it before the `Service` directive in the *proxy-id*`/config/obj.conf` file.

Syntax

```
<Directive> fn="append-header" name="<header_name>",
value="<header_value>"
```

Example

```
<Object ppath="http://.*">

ObjectType fn="cache-enable"

ObjectType fn="cache-setting" max-uncheck="7200" lm-factor="0.100"

ObjectType fn="append-header" name="MyHeader" value="MyValue"

Service fn="proxy-retrieve"

</Object>
```

Filtering by MIME Type

You can configure the proxy server to block certain files that match a MIME type. For example, you could set up your proxy server to block any executable or binary files so that any clients using your proxy server can't download a possible computer virus.

If you want the proxy server to support a new MIME type, in the Server Manager, choose System Settings | MIME Types and add the type. See "Creating MIME Types" on page 46 for more information.

You can combine filtering MIME types with templates, so that only certain MIME types are blocked for specific URLs. For example, you could block executables coming from any computer in the .edu domain.

To filter by MIME type:

1. In the Server Manager, choose Filters | MIME Filters.

2. Choose the template you want to use for filtering MIME types, or make sure you're editing the entire server.

3. In the Current filter text box, you can type a regular expression that matches the MIME types you want to block.

   For example, to filter out all applications, you could type (application/.*) for the regular expression. This is faster than checking each MIME type for every application type (as described in the following step). The regular expression is not case-sensitive. For more information on regular expressions, see "Understanding Regular Expressions" on page 32.

4. Check the MIME types you want to filter. When a client attempts to access a file that is blocked, the proxy server returns a “forbidden” message.
5. Click OK to submit the form. Be sure to save and apply your changes.

Filtering out HTML Tags

The proxy server lets you specify HTML tags you want to filter out before passing the file to the client. This lets you filter out objects such as Java applets and JavaScript embedded in the HTML file. To filter HTML tags, you specify the beginning and ending HTML tags. The proxy then substitutes blanks for all text and objects in those tags before sending the file to the client.

NOTE The proxy stores the original (unedited) file in the cache, if the proxy is configured to cache that resource.

To filter out HTML tags:

1. In the Server Manager, choose Filters | HTML Tag Filters.
2. In the form that appears, choose the template you want to modify. You might choose HTTP, or you might choose a template that specifies only certain URLs (such as those from hosts in the .edu domain).
3. Check the filter box for any of the default HTML tags you want to filter. These are the default tags:
   - APPLET usually surrounds Java applets.
   - SCRIPT indicates the start of JavaScript code.
   - IMG specifies an inline image file.
4. You can enter any HTML tags you want to filter. Type the beginning and ending HTML tags.

   For example, to filter out forms, you could type `FORM` in the Start Tag box (the HTML tags are not case-sensitive) and `/FORM` in the End Tag box. If the tag you want to filter does not have an end tag, such as OBJECT and IMG, you can leave the End Tag box empty.
5. Click OK to submit the form. You need to save and apply your changes and restart the proxy before the filtering will begin.

Chapter 11

Using the Client Autoconfiguration File

If you have multiple proxy servers that support many clients, you can use a client autoconfiguration file to configure all of your Netscape Navigator clients. The autoconfiguration file contains a JavaScript function that determines which proxy, if any, Navigator uses when accessing various URLs.

When Netscape Navigator starts, it loads the autoconfiguration file. Each time the user clicks a link or types in a URL, the Navigator uses the configuration file to determine if it should use a proxy and, if so, which proxy it should use. This feature lets you provide an easy way to configure all copies of Netscape Navigator in your organization. There are several ways you can get the autoconfiguration file to your clients.

- You can use the proxy server as a web server that returns the autoconfiguration file. You point Netscape Navigator to the proxy's URL. Having the proxy act as a web server lets you keep the autoconfiguration file in one place so that when you need to make updates, you need to change only one file.
- You can store the file on a web server, an FTP server, or any network directory to which Navigator has access. You configure Navigator to find the file by giving it the URL to the file, so any general URL will do. If you need to do complex calculations (for example, if you have large proxy chains in your organization), you might write a web server CGI program that outputs a different file depending on who accesses the file.
- You can store the autoconfiguration file locally with each copy of Netscape Navigator; however, if you need to update the file, you'll have to distribute copies of the file to each client.

You can create the autoconfiguration file two ways: you can use a form in the proxy Server Manager, or you can create the file manually. Directions for creating the files appear later in this chapter.

Understanding Autoconfiguration Files

Unlike the other files described in this book, the autoconfiguration file is primarily a feature of Netscape Navigator 2.0 and later versions. However, this feature is documented in this book because it's likely that you, as the person administering the proxy server, will also create and distribute the client autoconfiguration files.

What Does the Autoconfiguration File Do?

The autoconfiguration file is written in JavaScript, a compact, object-based scripting language for developing client and server Internet applications. Netscape Navigator interprets the JavaScript file.

When Netscape Navigator is first loaded, it downloads the autoconfiguration file. The file can be kept anywhere that Navigator can get to it by using a URL. For example, the file can be kept on a web server. The file could even be kept on a network file system, provided the Navigator can get to it using a file:// URL.

The proxy configuration file is written in JavaScript. The JavaScript file defines a single function (called **FindProxyForURL**) that determines which proxy server, if any, Navigator should use for each URL. Navigator sends the JavaScript function two parameters: the host name of the computer from which Navigator is running and the URL it's trying to obtain. The JavaScript function returns a value to Navigator that tells it how to proceed.

The autoconfiguration file makes it possible to specify different proxies (or no proxy at all) for various types of URLs, various servers, or even various times of the day. In other words, you can have multiple specialized proxies so that, for example, one serves the .com domain, another the .edu domain, and yet another serves everything else. This lets you divide the load and get more efficient use of your proxies' disks because there is only a single copy of any file in the cache (instead of multiple proxies all storing the same documents).

Autoconfiguration files also support proxy failover, so if a proxy server is unavailable, Navigator will transparently switch to another proxy server.

Accessing the Proxy as a Web Server

You can store one or more autoconfiguration files on the proxy server and have the proxy server act as a web server whose only documents are autoconfiguration files. This lets you, the proxy administrator, maintain the proxy autoconfiguration files needed by the clients in your organization. It also lets you keep the files in a central location, so if you have to update the files, you do it once and all Netscape Navigator clients automatically get the updates.

You keep the proxy autoconfiguration files in the *server root*`/proxy-`*id*`/pac/` directory (for example, `usr/ns-home/proxy-proxy1/pac`). In Netscape Navigator, you enter the URL to the proxy autoconfiguration file by choosing Options | Network Preferences and then typing the URL to the file in the Proxies tab. The URL for the proxy has this format:

`http://`*proxy*`.`*domain*`:`*port*`/`*URI*

For example, the URL could be http://proxy.netscape.com. You don't need to specify a URI (part of the URL following the host:port combination); however, if you do use a URI, you can then use a template to control access to the various autoconfiguration files. For example, if you create a URI called /test that contains an autoconfiguration file called `/proxy.pac`, you can create a template with the resource pattern http://proxy.mysite.com:8080/test/.*. You can then use that template to set up access control specifically to that directory.

You can create multiple autoconfiguration files and have them accessed through different URLs. Table 11-1 lists some example URIs and the URLs the clients would use to access them.

Table 11-1 Sample URIs and corresponding URLs

URI (path)	URL to the proxy
/	http://proxy.mysite.com
/employees	http://proxy.mysite.com/employees
/group1	http://proxy.mysite.com/group1
/managers	http://proxy.mysite.com/managers

Using Pac Files with a Reverse Proxy

Because of the way a reverse proxy works, it can be very difficult to have a proxy server work as a reverse proxy and server .pac files. This is because the proxy server gets a request for a file and it needs to determine if the request is for a local .pac file or if the request is for a remote document.

In order to have the proxy server act as a reverse proxy in addition to maintaining and serving a .pac file, you need to manually edit the obj.conf file to make sure the order of the NameTrans functions is correct.

Create a regular mapping to have the proxy server act as a reverse proxy. This typically tells the proxy to route all requests to the remote content server. You can add a proxy autoconfiguration file and map it to a specific directory, such as /pac. In this case, any client who wants to get the .pac file would use a URL such as:

http://proxy.mysite.com/pac

CAUTION With this mapping, however, you must be sure that the remote content server doesn't have a similar directory.

Edit the obj.conf file to make sure that the directive and function for the proxy autoconfiguration file appear before any other mappings. This directive and function must be first because the proxy server normally runs through all NameTrans functions before servicing the request. However, with autoconfiguration files, the proxy immediately recognizes the path and returns the .pac file.

Here's an example from an obj.conf file that uses a reverse proxy and maintains an autoconfiguration file:

```
<Object name="default">
NameTrans from="file:" fn="map" to="ftp:"
NameTrans from="/pac" fn="pac-map" name="file" to="/ns-home/proxy/pac/proxy.pac"
NameTrans fn="redirect" from="http://foo.*" url="http://www.acme.com"
NameTrans from="/ns-icons" fn="pfx2dir" dir="/ns-home/ns-icons" name="file"
NameTrans fn="reverse-map" from="http://web.acme.com"
to="http://proxy.acme.com:8080"
NameTrans fn="map" from="http://proxy.acme.com:8080" to="http://web.acme.com"
NameTrans fn="map" from="/" to="http://web.acme.com"
PathCheck fn="url-check"
Service fn="deny-service"
AddLog fn="flex-log" name="access"
AddLog fn="urldb-record"
</Object>
```

Using the Server Manager Forms to Create an Autoconfiguration File

To create an autoconfiguration file using the Server Manager forms:

1. In the Server Manager choose Routing | Client Autoconfiguration. The form that appears lists any autoconfiguration files you have on your proxy's computer. You can click the autoconfiguration file to edit it. The remaining steps tell you how to create a new file.

2. Type an optional URI (the path portion of a URL) that clients will use when getting the autoconfiguration file from the proxy. For example, type `/` to let clients access the file as the proxy's main document (similar to an `index.html` file for a web server); clients would then use only the domain name when accessing the proxy for the autoconfiguration file. You can use multiple URIs and create separate autoconfiguration files for each URI.

3. Type a name for the autoconfiguration file using the `.pac` extension. If you have one file, you might call it simply `proxy.pac` (pac is short for proxy autoconfiguration). All autoconfiguration files are ASCII text files with a single JavaScript function (see "Creating the Autoconfiguration File Manually" on page 165 for more information on the syntax of the files).

4. Click OK. Another form appears. Use this form to create an autoconfiguration file. The items on the form are followed in order by the client. These are the items on the form:

- "Never go direct to remote server" tells Navigator to always use your proxy. You can specify a second proxy server to use in case your proxy server isn't running.

- "Go direct to remote server when" lets you bypass the proxy server on certain occasions. Navigator determines those occasions in the order the options are listed on the form:

  - "Connecting to non-fully qualified host names" tells Navigator to go to a server directly when the user specifies only the computer name. For example, if there's an internal web server called winternal.mysite.com, the user might type only `http://winternal` instead of the fully qualified domain name. In this case, Navigator goes directly to the web server instead of to the proxy.

  - "Connecting to a host in domain" lets you specify up to three domain names that Navigator can access directly. When specifying the domains, begin with the dot character. For example, you could type `.netscape.com`.

- "Connecting to a resolvable host" makes Navigator go directly to the server when the client can resolve the host. This option is typically used when DNS is set to resolve only local (internal) hosts. The clients would use a proxy server when connecting to servers outside of the local network.

CAUTION The above option causes the client to consult DNS for every request. It therefore, negatively impacts the performance witnessed by the client. Because of the performance impact, you should avoid using this option.

- "Connecting to a host in subnet" makes Navigator go directly to the server when the client accesses a server in a particular subnet. This option is useful when an organization has many subnets in a geographical area. For example, some companies might have one domain name that applies to subnets around the world, but each subnet is specific to a particular region.

CAUTION The above option causes the client to consult DNS for every request. It therefore, negatively impacts the performance witnessed by the client. Because of the performance impact, you should avoid using this option.

- "Except when connecting to hosts" lets you specify exceptions to the rule of going directly to a server. For example, if you type `.netscape.com` as a domain to which to go directly, you could make an exception for going to home.netscape.com. This tells Navigator to use your proxy when going to home.netscape.com but go directly to any other server in the netscape.com domain.
- "Secondary failover proxy" specifies a second proxy to use if your proxy server isn't running.
- "Failover direct" tells Navigator to go directly to the servers if your proxy server isn't running. If you specify a secondary failover proxy, Navigator tries the second proxy server before going directly to the server.

5. Click OK to create the autoconfiguration file. The file is stored in the directory *server-root*`/proxy-`*id*`/pac`. You'll get a confirmation message saying the file was created correctly. Repeat the preceding steps to create as many autoconfiguration files as you need.

Once you create your autoconfiguration file, make sure you either tell all the people using your proxy server to point to the correct autoconfiguration file or configure the copies of Navigator yourself.

Creating the Autoconfiguration File Manually

This section describes how you can manually create autoconfiguration files.

The proxy autoconfiguration file is written using client-side JavaScript. Each file contains a single JavaScript function called **FindProxyForURL** that determines which proxy server, if any, Navigator should use for each URL. Navigator sends the JavaScript function two parameters: the host name of the destination origin server and the URL it's trying to obtain. The JavaScript function returns a value to Navigator that tells it how to proceed. The following section describes the function syntax and the possible return values.

The FindProxyForURL Function

For more information on writing JavaScript, see the *JavaScript Guide* that comes with most versions of Netscape Navigator. The syntax of the **FindProxyFor URL** function is:

```
function FindProxyForURL(url, host)
{
   ...
}
```

For every URL Netscape Navigator accesses, it sends the **url** and **host** parameters and calls the function in the following way:

```
ret = FindProxyForURL(url, host);
```

url is the full URL being accessed in Netscape Navigator.

host is the host name extracted from the URL that is being accessed. This is only for convenience; it is the same string as between **://** and the first **:** or **/** after that. The port number is not included in this parameter. It can be extracted from the URL when necessary.

ret (the return value) is a string describing the configuration.

The Function Return Values

The autoconfiguration file contains the function **FindProxyForURL**. As parameters, this function uses the client host name and the URL it's accessing. The function returns a single string that tells Navigator how to proceed. If the string is null, no proxies should be used. The string can contain any number of the building blocks shown in Table 11-2, separated by semicolons.

Table 11-2 FindProxyForURL Return Values

Return values	Resulting action of Netscape Navigator
`DIRECT`	Make connections directly to the server without going through any proxies.
`PROXY` *host:port*	Use the specified proxy and port number. If multiple values are separated by semicolons, the first proxy is used. If that proxy fails, then the next proxy is used, and so on.
`SOCKS` *host:port*	Use the specified SOCKS server. If there are multiple values separated by semicolons, the first proxy is used. If that proxy fails, then the next proxy is used, and so on.

If Netscape Navigator encounters an unavailable proxy server, Navigator will automatically retry the previously unresponsive proxy after 30 minutes, then after one hour, and so on, at 30-minute intervals. This means that if you temporarily shut down a proxy server, your clients will resume using the proxy no later than 30 minutes after it was restarted.

If all of the proxies are down and the DIRECT return value isn't specified, Netscape Navigator will ask the user if it should temporarily ignore proxies and attempt direct connections instead. Navigator will ask if proxies should be retried after 20 minutes, then again in another 20 minutes, and so on at 20-minute intervals.

In the following example, the return value tells Netscape Navigator to use the proxy called w3proxy.netscape.com on port 8080, but if that proxy is unavailable, Navigator uses the proxy called mozilla.netscape.com on port 8080:

```
PROXY w3proxy.netscape.com:8080; PROXY mozilla.netscape.com:8080
```

In the next example, the primary proxy is w3proxy.netscape.com:8080; if that proxy is unavailable, Navigator uses mozilla.netscape.com:8080. If both proxies are unavailable, then Navigator goes directly to the server (and after 20 minutes, Navigator asks the user if it should retry the first proxy):

```
PROXY w3proxy.netscape.com:8080; PROXY mozilla.netscape.com:8080;
DIRECT
```

JavaScript Functions and Environment

JavaScript has several predefined functions and environmental conditions that are useful with proxying. Each of these functions checks whether or not a certain condition is met and then returns a value of true or false. The related utility functions are an exception because they return a DNS host name or IP address. You can use these functions in the main **FindProxyForURL** function to determine what return value to send to Netscape Navigator. See the examples later in this chapter for ideas on using these functions.

Each of the functions or environmental conditions is described in this section. The functions and environmental conditions that apply to Netscape Navigator integration with the proxy are:

host name-based conditions:

- dnsDomainIs()
- isInNet()
- isPlainhost name()
- isResolvable()
- localHostOrDomainIs()
- Related utility functions:
- dnsDomainLevels()
- dnsResolve()
- myIpAddress()
- URL / host name-based condition:
- shExpMatch()
- Time-based conditions:
- dateRange()
- timeRange()
- weekdayRange()

Hostname-based Functions

The hostname-based functions let you use the host name or IP address to determine which proxy, if any, to use.

dnsDomainIs(host, domain)

The **dnsDomainIs()** function detects whether the URL host name belongs to a given DNS domain. This function is useful when you are configuring Netscape Navigator not to use proxies for the local domain as illustrated in examples 1 and 2 on page 176.

This function is also useful when you are using multiple proxies for load balancing in situations where the proxy that receives the request is selected from a group of proxies based on which DNS domain the URL belongs to. For example, if you are load balancing by directing URLs containing .edu to one proxy and those containing .com to another proxy, you can check the URL host name using **dnsDomainIs()**.

Parameters:

host is the host name from the URL.

domain is the domain name to test the host name against.

Returns:

true or false

Examples:

The following statement would be true:

```
dnsDomainIs("www.netscape.com", ".netscape.com")
```

The following statements would be false:

```
dnsDomainIs("www", ".netscape.com")
dnsDomainIs("www.mcom.com", ".netscape.com")
```

isInNet(host, pattern, mask)

The **isInNet()** function enables you to resolve a URL host name to an IP address and test if it belongs to the subnet specified by the mask. This is the same type of IP address pattern matching that SOCKS uses. See example 4 on page 177.

Parameters:

host is a DNS host name or IP address. If a host name is passed, this function will resolve it into an IP address.

pattern is an IP address pattern in the dot-separated format

mask is the IP address pattern mask that determines which parts of the IP address should be matched against. A value of 0 means ignore; 255 means match. This function is true if the IP address of the host matches the specified IP address pattern.

Returns:

true or false

Examples:

This statement is true only if the IP address of the host matches exactly 198.95.249.79:

```
isInNet(host, "198.95.249.79", "255.255.255.255")
```

This statement is true only if the IP address of the host matches 198.95.*.*:

```
isInNet(host, "198.95.0.0", "255.255.0.0")
```

isPlainhost name(host)

The **isPlainhost name()** function detects whether the host name in the requested URL is a plain host name or a fully qualified domain name. This function is useful if you want Netscape Navigator to connect directly to local servers as illustrated in examples 1 and 2 on page 176.

Parameters:

host is the host name from the URL (excluding port number) only if the host name has no domain name (no dotted segments).

Returns:

true if **host** is local; false if **host** is remote

Example:

```
isPlainhost name("host")
```

If **host** is something like www, then it returns true; if host is something like www.netscape.com, it returns false.

isResolvable(host)

If the DNS inside the firewall recognizes only internal hosts, you can use the **isResolvable()** function to test whether a host name is internal or external to the network. Using this function, you can configure Netscape Navigator to use direct connections to internal servers and to use the proxy only for external servers. This is useful at sites where the internal hosts inside the firewall are able to resolve the DNS domain name of other internal hosts, but all external hosts are unresolvable. The **isResolvable()** function consults DNS, attempting to resolve the host name into an IP address. See example 3 on page 177.

Parameters:

host is the host name from the URL. This tries to resolve the host name and returns true if it succeeds.

Returns:

true if it can resolve the host name, false if it cannot

Example:

```
isResolvable("host")
```

If **host** is something like www and can be resolved through DNS, then this function returns true.

localHostOrDomainIs(host, hostdom)

The **localHostOrDomainIs()** function specifies local hosts that might be accessed by either the fully qualified domain name or the plain host name. See example 2 on page 176.

The **localHostOrDomainIs()** function returns `true` if the host name matches the specified host name exactly or if there is no domain name part in the host name that the unqualified host name matches.

Parameters:

host is the host name from the URL.

hostdom is the fully qualified host name to match.

Returns:

true or false

Examples:

The following statement is true (exact match):

```
localHostOrDomainIs("www.netscape.com", "www.netscape.com")
```

The following statement is true (host name match, domain name not specified):

```
localHostOrDomainIs("www", "www.netscape.com")
```

The following statement is false (domain name mismatch):

```
localHostOrDomainIs("www.mcom.com", "www.netscape.com")
```

The following statement is false (host name mismatch):

```
localHostOrDomainIs("home.netscape.com", "www.netscape.com")
```

Related Utility Functions

The related utility functions enable you to find out domain levels, the host on which Netscape Navigator is running, or the IP address of a host.

dnsDomainLevels(host)

The **dnsDomainLevels()** function finds the number of DNS levels (number of dots) in the URL host name.

Parameters:
host is the host name from the URL.

Returns:
number (integer) of DNS domain levels.

Examples:

```
dnsDomainLevels("www")
```

returns 0.

```
dnsDomainLevels("www.netscape.com")
```

returns 2.

dnsResolve(host)

The **dnsResolve()** function resolves the IP address of the given host (typically from the URL). This is useful if the JavaScript function has to do more advanced pattern matching than can be done with the existing functions.

Parameters:
host is the host name to resolve. Resolves the given DNS host name into an IP address, and returns it in the dot-separated format as a string.

Returns:
dotted quad IP address as a string value

Example:
The following example would return the string 198.95.249.79.

```
dnsResolve("home.netscape.com")
```

myIpAddress()

The **myIpAddress()** function is useful when the JavaScript function has to behave differently depending on what host on which Netscape Navigator is running. This function returns the IP address of the computer that is running Navigator.

Returns:
dotted quad IP address as a string value

Example:
The following example returns the string 198.95.249.79 if you are running Navigator on the computer home.netscape.com.

```
myIpAddress()
```

URL/host-name-based Condition

You can match host names or URLs for load balancing and routing.

shExpMatch(str, shexp)

The **shExpMatch()** function matches either the URL host names or the URLs themselves. The main use of this function is for load balancing and intelligent routing of URLs to different proxy servers.

Parameters:
str is any string to compare (for example, the URL or the host name).

shexp is a shell expression against which to compare.

This expression is true if the string matches the specified shell expression. See example 6 on page 179.

Returns:
true or false

Examples:
The first example returns true; the second returns false.

```
shExpMatch("http://home.netscape.com/people/index.html",
            ".*/people/.*")
```

```
shExpMatch("http://home.netscape.com/people/yourpage/index.html",
            ".*/mypage/.*")
```

Time-based Conditions

You can make the **FindProxyForURL** function behave differently depending on the date, time, or day of the week.

dateRange (day, month, year...)

The **dateRange()** function detects a particular date or a range of dates, such as April 19th, 1996 through May 3rd, 1996. This is useful if you want the **FindProxyForURL** function to act differently depending on what day it is, such as if maintenance down time is regularly scheduled for one of the proxies.

The date range can be specified several ways:

```
dateRange(day)
dateRange(day1, day2)
dateRange(mon)
dateRange(month1, month2)
dateRange(year)
dateRange(year1, year2)
dateRange(day1, month1, day2, month2)
dateRange(month1, year1, month2, year2)
dateRange(day1, month1, year1, day2, month2, year2)
dateRange(day1, month1, year1, day2, month2, year2, gmt)
```

Parameters:

day is an integer between 1 and 31 for the day of month.

month is one of the month strings:
`JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC`

year is a four-digit integer for the year number (for example, 1996).

gmt is either the string GMT, which makes time comparisons occur in Greenwich Mean Time, or is left blank so that times are assumed to be in the local time zone. The GMT parameter can be specified in any of the call profiles, always as the last parameter. If only a single value is specified (from each category: day, month, year), the function returns a true value only on days that match that specification. If two values are specified, the result is true from the first time specified through the second time specified.

Examples:

This statement is true on the first day of each month, local time zone.
```
dateRange(1)
```

This statement is true on the first day of each month, Greenwich Mean Time.
```
dateRange(1, "GMT")
```

This statement is true for the first half of each month.
```
dateRange(1, 15)
```

This statement is true on the 24th of December each year.
```
dateRange(24, "DEC")
```

This statement is true on the 24th of December, 1995.

```
dateRange(24, "DEC", 1995)
```

This statement is true during the first quarter of the year.

```
dateRange("JAN", "MAR")
```

This statement is true from June 1st through August 15th, each year.

```
dateRange(1, "JUN", 15, "AUG")
```

This statement is true from June 1st, 1995, until August 15th, 1995.

```
dateRange(1, "JUN", 15, 1995, "AUG", 1995)
```

This statement is true from October 1995 through March 1996.

```
dateRange("OCT", 1995, "MAR", 1996)
```

This statement is true during the entire year of 1995.

```
dateRange(1995)
```

This statement is true from the beginning of 1995 until the end of 1997.

```
dateRange(1995, 1997)
```

timeRange (hour, minute, second...)

The **timeRange** function detects a particular time of day or a range of time, such as 9 p.m. through 12 a.m. This is useful if you want the **FindProxyForURL** function to act differently depending on what time it is.

```
timeRange(hour)
timeRange(hour1, hour2)
timeRange(hour1, min1, hour2, min2)
timeRange(hour1, min1, sec1, hour2, min2, sec2)
```

Parameters:

hour is the hour from 0 to 23. (0 is midnight, 23 is 11:00 p.m.)

min is the number of minutes from 0 to 59.

sec is the number of seconds from 0 to 59.

gmt is either the string GMT for GMT time zone, or not specified for the local time zone. This parameter can be used with each of the parameter profiles and is always the last parameter.

Returns:

true or false

Examples:

This statement is true from noon to 1:00 p.m.

```
timerange(12, 13)
```

This statement is true noon to 12:59 p.m. GMT.

```
timerange(12, "GMT")
```

This statement is true from 9:00 a.m. to 5:00 p.m.

```
timerange(9, 17)
```

true between midnight and 30 seconds past midnight.

```
timerange(0, 0, 0, 0, 0, 30)
```

weekdayRange(wd1, wd2, gmt)

The **weekdayRange()** function detects a particular weekday or a range of weekdays, such as Monday through Friday. This is useful if you want the **FindProxyForURL** function to act differently depending on the day of the week.

Parameters:

wd1 and **wd2** are any one of these weekday strings:

```
SUN MON TUE WED THU FRI SAT
```

gmt is either GMT for Greenwich Mean Time, or is left out for local time.

Only the first parameter, wd1, is mandatory. Either wd2, gmt, or both can be left out.

If only one parameter is present, the function returns a true value on the weekday that the parameter represents. If the string GMT is specified as a second parameter, times are taken to be in GMT otherwise the times are in your local time zone.

If both wd1 and wd2 are defined, the condition is true if the current weekday is between those two weekdays. Bounds are inclusive. The order of parameters is important; “MON”, “WED” is Monday through Wednesday, but “WED”, “MON” is from Wednesday to the Monday of the next week.

Examples:

The following is true Monday through Friday (local time zone).

```
weekdayRange("MON", "FRI")
```

The following is true Monday through Friday, in Greenwich Mean Time.

```
weekdayRange("MON", "FRI", "GMT")
```

The following is true on Saturdays, local time.

```
weekdayRange("SAT")
```

The following is true on Saturdays, in Greenwich Mean Time.

```
weekdayRange("SAT", "GMT")
```

The following is true Friday through Monday (the order is important)

```
weekdayRange("FRI", "MON")
```

Example 1: Proxy All Servers Except Local Hosts

In this example, Netscape Navigator connects directly to all hosts that aren't fully qualified and the ones that are in the local domain. Everything else goes through the proxy called w3proxy.netscape.com:8080.

NOTE If the proxy goes down, connections become direct automatically.

```
    function FindProxyForURL(url, host)
    {
       if (isPlainhost name(host) ||
          dnsDomainIs(host, ".netscape.com") ||
          dnsDomainIs(host, ".mcom.com"))
          return "DIRECT";
       else
          return "PROXY w3proxy.netscape.com:8080;
DIRECT";
    }
```

Example 2: Proxy Local Servers Outside the Firewall

This example is like the previous one, but it uses the proxy for local servers that are outside the firewall. If there are hosts (such as the main web server) that belong to the local domain but are outside the firewall and are only reachable through the proxy server, those exceptions are handled using the **localHostOrDomainIs()** function:

```
function FindProxyForURL(url, host)
{
   if ((isPlainhost name(host) ||
   dnsDomainIs(host, ".netscape.com")) &&
   !localHostOrDomainIs(host, "www.netscape.com") &&
   !localHostOrDoaminIs(host, "merchant.netscape.com"))
      return "DIRECT";
   else
      return "PROXY w3proxy.netscape.com:8080; DIRECT";
}
```

This example uses the proxy for everything except local hosts in the netscape.com domain. The hosts www.netscape.com and merchant.netscape.com also go through the proxy.

The order of the exceptions increases efficiency: **localHostOrDomainIs()** functions get executed only for URLs that are in the local domain, not for every URL. In particular, notice the parentheses around the *or* expression before the *and* expression.

Example 3: Proxy Only Unresolved Hosts

This example works in an environment where internal DNS is set up so that it can resolve only internal host names, and the goal is to use a proxy only for hosts that aren’t resolvable:

```
function FindProxyForURL(url, host)
{
   if (isResolvable(host))
         return "DIRECT";
      else
         return "PROXY proxy.mydomain.com:8080";
}
```

This example requires consulting the DNS every time, so it should be grouped with other rules so that DNS is consulted only if other rules do not yield a result:

```
function FindProxyForURL(url, host)
{

   if (isPlainhost name(host) ||
      dnsDomainIs(host, ".mydomain.com") ||
      isResolvable(host))
      return "DIRECT";
   else
      return "PROXY proxy.mydomain.com:8080";
}
```

Example 4: Connect Directly to a Subnet

In this example all the hosts in a given subnet are connected to directly others go through the proxy:

```
function FindProxyForURL(url, host)
{
   if (isInNet(host, "198.95.0.0", "255.255.0.0"))
      return "DIRECT";
   else

      return "PROXY proxy.mydomain.com:8080";
}
```

You can minimize the use of DNS in this example by adding redundant rules in the beginning:

```
function FindProxyForURL(url, host)
{
   if (isPlainhost name(host) ||
      dnsDomainIs(host, ".mydomain.com") ||
      isInNet(host, "198.95.0.0", "255.255.0.0"))
      return "DIRECT";
   else
      return "PROXY proxy.mydomain.com:8080";
}
```

Example 5: Balance Proxy Load with dnsDomainIs()

This example is more sophisticated. There are four proxy servers, with one of them acting as a hot standby for the others, so if any of the remaining three goes down, the fourth one takes over. The three remaining proxy servers share the load based on URL patterns, which makes their caching more effective (there is only one copy of any document on the three servers, as opposed to one copy on each of them). The load is distributed as shown in Table 11-3.

Table 11-3 Balance Proxy Load

Proxy	Purpose
#1	.com domain
#2	.edu domain
#3	all other domains
#4	hot stand-by

All local accesses should be direct. All proxy servers run on port 8080. You can concatenate strings by using the + operator in JavaScript.

```
function FindProxyForURL(url, host)
{
   if (isPlainhost name(host) || dnsDomainIs(host, ".mydomain.com"))
      return "DIRECT";

   else if (dnsDomainIs(host, ".com"))
      return "PROXY proxy1.mydomain.com:8080; " +
           "PROXY proxy4.mydomain.com:8080";

   else if (dnsDomainIs(host, ".edu"))
      return "PROXY proxy2.mydomain.com:8080; " +
           "PROXY proxy4.mydomain.com:8080";
   else
      return "PROXY proxy3.mydomain.com:8080; " +
            "PROXY proxy4.mydomain.com:8080";
}
```

Example 6: Balance Proxy Load with shExpMatch()

This example is essentially the same as example 5, but instead of using **dnsDomainIs()**, this example uses **shExpMatch()**.

```
function FindProxyForURL(url, host)
{

if (isPlainhost name(host) || dnsDomainIs(host, ".mydomain.com"))
   return "DIRECT";
else if (shExpMatch(host, "*.com"))
   return "PROXY proxy1.mydomain.com:8080; " +
         "PROXY proxy4.mydomain.com:8080";
else if (shExpMatch(host, "*.edu"))
   return "PROXY proxy2.mydomain.com:8080; " +
         "PROXY proxy4.mydomain.com:8080";
else
   return "PROXY proxy3.mydomain.com:8080; " +
         "PROXY proxy4.mydomain.com:8080";
}
```

Example 7: Proxying a Specific Protocol

You can set a proxy to be for a specific protocol. Most of the standard JavaScript functionality is available for use in the **FindProxyForURL()** function. For example, to set different proxies based on the protocol, you can use the **substring()** function:

```
function FindProxyForURL(url, host)
{
   if (url.substring(0, 5) == "http:") {
      return "PROXY http-proxy.mydomain.com:8080";
   }
   else if (url.substring(0, 4) == "ftp:") {
      return "PROXY ftp-proxy.mydomain.com:8080";
   }
   else if (url.substring(0, 7) == "gopher:") {
      return "PROXY gopher-proxy.mydomain.com:8080";
   }
   else if (url.substring(0, 6) == "https:" ||
          url.substring(0, 6) == "snews:") {
      return "PROXY security-proxy.mydomain.com:8080";
   }
```

```
    else {
        return "DIRECT";
    }
}
```

You can also accomplish this using the **shExpMatch()** function; for example:

```
...
if (shExpMatch(url, "http:*")) {
    return "PROXY http-proxy.mydomain.com:8080;
}

...
```

Chapter 12

Monitoring the Server’s Status

You can monitor your server’s status in realtime by using the *Simple Network Management Protocol* (SNMP). SNMP is an Internet network management protocol used to monitor network devices. You can also monitor your server by recording and viewing log files.

Monitoring the Server Using HTTP

You can monitor your server’s process usage by using the interactive server monitor. You can see how your server is handling its traffic and whether or not the processes currently assigned are sufficient. If traffic increases and the server becomes sluggish, you’ll need to adjust the server configuration or the system’s network kernel.

To monitor your server from the Server Manager:

1. In the Server Manager, choose Server Status | Monitor Current Activity.
2. Click Monitor.

On the page that appears, you can see the server usage, a breakdown of server activity, and some server totals.

Server Usage

You can monitor the following server usage areas:

- Server size—Shows how much of the server resources are being used.
- Process utilization—Shows the percentage of processes being used.

Activity Breakdown

You can see the number of processes (in terms of percentages) currently being used in the following server activity tasks:

- Awaiting connection (idle)
- Reading request
- Writing response
- Resolving host name
- Processing request (includes general HTTP work and logging)

Totals

You can see the following server totals:

- Bytes transferred
- Total requests
- Bad requests
- 2xx—Status codes from 200 to 299.
- 3xx—Status codes from 300 to 399.
- 4xx—Status codes from 400 to 499.
- 5xx—Status codes of 500 and higher.
- xxx—All 2xx, 3xx, 4xx and 5xx responses (total connections minus timeouts and other errors that did not give back an HTTP status code)
- 200—OK status code (successful transaction).
- 302—Relocated URL status code.
- 304—Local copy of URL was used.
- 401—Unauthorized status code.
- 403—Forbidden URL status code.

Working with Log Files

Server log files record your server's activity. You can use these logs to monitor your server and help you when troubleshooting. The error log file, located in *server root*/proxy-*id*/logs, lists all the errors the server has encountered. The access log, also located in proxy-*id*/logs in the server root directory, records information about requests to the server and responses from the server. You can specify what is included in the access log file from the Server Manager. Use the log analyzer to generate server statistics. You can back up server error and access log files by archiving them.

Viewing the Error Log File

The error log file contains errors the server has encountered since the log file was created; it also contains informational messages about the server, such as when the server was started. Incorrect user authentication is also recorded in the error log.

To view the error log file from the Server Manager,

1. In the Server Manager, choose Server Status | View Error Log.
2. If you want to see more or less than 25 lines of the error log, use the Number of errors to view field to enter the number of lines you'd like to see.
3. If you'd like to filter the error messages for a particular word, type the word in the "Only show entries with" field. Make sure the case for your entry matches the case of the word for which you're searching. (For example, if you want to see only error messages that contain "warning," type warning.)

This is an example of an error log:

```
[13/Feb/1996:16:56:51] info: successful server startup
[20/Mar/1996 19:08:52] warning: for host wiley.a.com trying to GET /report.html,
append-trailer reports: error opening /usr/ns-home/docs/report.html(No such file
or directory)
[30/Mar/1996 15:05:43] security: for host arrow.a.com trying to GET /, basic-ncsa
    reports: user jane password did not match database /usr/ns-home/authdb/mktgdb
```

In this example, the first line is an informational message—the server started up successfully. The second log entry shows that the client wiley.a.com requested the file report.html, but the file wasn't in the primary document directory on the server. The third log entry shows that the password entered for the user jane was incorrect.

Viewing an Access Log File

You can view the server’s active and archived access log files from the Server Manager.

To view an access log,

1. In the Server Manager, choose Server Status | View Access Log.
2. Choose the access log file you want to see. Active log files for resources and archived log files appear in the list.
3. To limit how much of the access log you’ll see, type the number of lines you want to see in the Number of entries field.
4. If you’d like to filter the access log entries for a particular word, type the word in the Only show entries with field. Make sure the case for your entry matches the case of the word for which you’re searching. (For example, if you want to see only access log entries that contain “POST,” type `POST`.)

This is a sample of an access log in the common logfile format:

```
wiley.a.com - - [16/Feb/1996:21:18:26 -0800] “GET / HTTP/1.0” 200 751
wiley.a.com - - [17/Feb/1996:1:04:38 -0800] “GET /docs/grafx/icon.gif HTTP/1.0”
204 342
wiley.a.com - - [20/Feb/1996:4:36:53 -0800] “GET /help HTTP/1.0” 401 571
arrow.a.com - john [29/Mar/1996:4:36:53 -0800] “GET /help HTTP/1.0” 401 571
```

Table 12-1 describes the last line of the sample access log.

Table 12-1 The last line of the sample access log file has several components.

Access Log field	Example
host name or IP address of client	arrow.a.com. In this case, the host name is shown because DNS is enabled; if DNS were disabled, the client’s IP address would appear.
RFC 931 information	- (RFC 931 identity—not implemented)
User name	john (user name entered by the client for authentication)
Date/time of request	29/Mar/1996:4:36:53 -0800
Request	GET /help
Protocol	HTTP/1.0
Status code	401
Bytes transferred	571

Understanding Access Logfile Syntax

There are three predetermined logfile formats:

- Common
- Extended
- Extended-2

Common

Common format is the most basic of the log formats.

Syntax

host - usr [*time*] *"req" s1 c1*

Fields

host is the client's DNS host name. If reverse DNS lookup is not enabled on your proxy, host is the client's IP address.

- is the RFC 931 style remote identity. (This parameter is not supported unless you are running your proxy as a SOCKS server.)

usr is the name of the user authenticated to the proxy.

time is the time and date of the request.

req is the first HTTP request line as it came into the proxy.

s1 is the proxy's HTTP response status code to the client.

c1 is the content-length sent to the client by the proxy.

Extended

Extended format is more detailed than common format because it includes all of the fields of the common format as well as some additional fields.

Syntax

host - usr [*time*] *"req" s1 c1 s2 c2 b1 b2 h1 h2 h3 h4 xt*

Fields

The following are the fields that the extended format includes that the common format does not include:

s2 is the remote server's HTTP response status code to the proxy whenever the proxy makes a request in part of the client.

c2 is the content-length received from the remote server by the proxy.

b1 is the size of the client’s HTTP request message body. (In other words, it is POST-data that will be forwarded to the remote server. This data will also be passed to the remote server if no error occurs.)

b2 is the size of the proxy’s HTTP request message body. It is the amount of data in the body that was sent to the remote server. (This data is the same as b1 if no error occurs.)

h1 is the size of the client’s HTTP request header to the proxy.

h2 is the size of the proxy server’s response header to the client.

h3 is the size of the proxy server’s request header to the remote server.

h4 is the size of the remote server’s HTTP response header to the proxy.

xt is the total transfer time, in seconds.

Extended-2

Extended-2 format is the most detailed log format because it includes all of the fields of the extended format as well as some additional fields.

Syntax

host - *usr* [*time*] *“req”* s*1 c1 s2 c2 b1 b2 h1 h2 h3 h4 xt route cs ss cs*

Fields

The following are the fields that the extended-2 format includes that the other two formats do not include:

route is the route used to retrieve the resource. The route field can hold one of the following:

- DIRECT means that the resource was retrieved directly.
- PROXY(host:port) means that the resource was retrieved through a proxy server at a specified host and port.
- SOCKS(host:port) means that the resource was retrieved through a SOCKS server at a specified host and port.

cs is the client finish status. This field specifies if the request to the client was successfully carried out to completion, interrupted by the client clicking the Stop button in Navigator, or aborted by an error condition. The cs field can hold one of the following:

- **-** means that the request was never started.
- FIN means that the request was completed successfully.

- INTR means that the request was interrupted by the client or terminated by a proxy or server time out.

ss is the remote server finish status. This field specifies if the request to the remote server was successfully carried out to completion, interrupted by the client clicking the Stop button in Navigator, or aborted by an error condition. The **ss** field can hold one of the following:

- - means that the request was never started.
- FIN means that the request was completed successfully.
- INTR means that the request was interrupted by the client or terminated by the proxy.
- TIMEOUT means that the request was timed out by the proxy.

cs is the cache finish status. This field specifies whether the cache file was written, refreshed, or returned by an up-to-date check. The **cs** field can hold one of the following:

- - means that the resource was not cacheable.
- WRITTEN means that the cache file was created.
- REFRESHED means that the cache file was updated or refreshed.
- NO-CHECK means that the cache file was returned without an up-to-date check.
- UP-TO-DATE means that the cache file was returned with an up-to-date check.
- HOST-NOT-AVAILABLE means that the remote server was not available for an up-to date check, so the cache file was returned without a check.
- CL-MISMATCH means that the cache file write was aborted due to a content-length mismatch.
- ERROR means that the cache file write was aborted due to any error other than the above. These errors include a client interruption and a server timeout.

Understanding Status Codes

Table 12-2 lists and defines all of the status codes specified in the HTTP/1.1 RFC 2068. For more detailed descriptions of the codes, see the HTTP/1.1 specification, RFC 2068.

NOTE The 1xx status codes are not supported by HTTP/1.0.

Table 12-2 Status Codes

Code	Description
100	Continue - The client can continue its request.
101	Switching Protocols - The server has complied with the client's request to switch protocols.
200	OK - The request was successful.
201	Created - The request was successful and a new resource was created as a result.
202	Accepted - The request was accepted for processing.
203	Non-Authoritative Information - The meta-information in the entity-header is from a local or third-party copy.
204	No Content - The server serviced the request but there is no information to return.
205	Reset Content - The request was successful and the user agent should clear the input form for further input.
206	Partial Content - The server serviced a (byte) range request for the resource.
300	Multiple Choices - The requested resource could be one of multiple resources.
301	Moved Permanently - The requested resource has permanently moved to a new location.
302	Moved Temporarily - The requested resource has temporarily moved to a new location.
303	See Other - The response to the request is under a different URI and can be retrieved with a GET request.
304	Not Modified - The requested resource has not been modified since it was last requested.
305	Use Proxy - The requested resource must be accessed through a proxy server.

Table 12-2 Status Codes

Code	Description
400	Bad Request - The server cannot read the request because its syntax is incorrect.
401	Unauthorized - The server must authenticate the user before servicing the request.
402	Payment Required - This code is reserved but not yet defined in detail in the HTTP/1.1 specification.
403	Forbidden - The server refused to service the request.
404	Not Found - The server cannot find the requested resource.
405	Method Not Allowed - The method specified in the Request-Line is not permitted for the requested resource.
406	Not Acceptable - The requested resource can only generate response entities that have unacceptable content characteristics according to the accept headers sent in the request.
407	Proxy Authentication Required - The proxy server must authenticate the user before servicing the request.
408	Request Timeout - The client did not make its request within the amount of time the server will wait for requests.
409	Conflict - The server could not service the request due to a current conflict with the requested resource.
410	Gone - The requested resource is no longer available on the server.
411	Length Required - The server will not service the request without a Content-Length specified in the request.
412	Precondition Failed - A precondition specified in one or more of the request-header fields failed.
413	Request Entity Too Large - The server will not service the request because the requested resource is too large.
414	Request-URI Too Long - The server will not service the request because the requested URL is too long.
415	Unsupported Media Type - The server will not service the request because the format of the request is not supported by the requested resource for the requested method.
500	Internal Server Error - The server could not service the request because of an unexpected internal error.
501	Not Implemented - The sever cannot service the request because it does not support the request method.

Table 12-2 Status Codes

Code	Description
502	Bad Gateway - The proxy server received an invalid response from the content server or another proxy server in a proxy chain.
503	Service Unavailable - The server could not service the request because it was temporarily overloaded or undergoing maintenance.
504	Gateway Timeout - The proxy server did not receive a response from a chained proxy server or the origin content server within an acceptable amount of time.
505	HTTP Version Not Supported - The server does not support the HTTP version specified in the request.

Setting Access Log Preferences

During installation, an access log file named access was created for the server. You can customize access logging for your server by specifying whether or not to log accesses, who not to record accesses from, and whether or not the server should spend time looking up the domain names of clients when they access a resource.

Server access logs can be in common logfile format, extended log format, extended-2 log format, a format that includes only specified information, or a custom format of your own design. For more information about these logfile formats, see “Understanding Access Logfile Syntax” on page 185.

To set access logging preferences,

1. From the Server Manager, choose Server Status | Log Preferences.
2. Use the template to which you’d like to apply custom logging.
3. Select whether or not to log client accesses.
4. Type the full path for the log file. By default, the log files are kept in the logs directory in the server root directory.
5. Choose whether or not to record domain names or IP addresses in the log.
6. Choose which format the log file should be: common, extended, extended-2, only specified information (“Only log” radio button), or custom. If you click Only log, the following flexible log format items are available:
    - Client host name—The host name (or IP address if DNS is disabled) of the client requesting access.

- Authenticate user name—If authentication was necessary, you can have the authenticated user name listed in the access log.
- System date—The date and time of the client request.
- Full request—The exact request the client makes.
- Status—The status code the server returned to the client.
- Content length—The length, in bytes, of the document sent to the client.
- HTTP header, "referer"—The referer tells you the page the client used previously to access the current page. For example, if a user is looking at the results from a text search query, the referer is the page from which the user accessed the text search engine. Referers allow the server to create a list of backtracked links.
- HTTP header, "user-agent"—The user-agent information, which includes the type of browser the client is using, its version, and what operating system it's running on, comes from the user-agent field in the HTTP header information the client sends to the server.
- Method—The request method used.
- URI—Universal Resource Identifier is the path part of a URL. For example, for http://www.a.com:8080/special/docs, the URI is /special/docs.
- Query string of the URI—Anything after the question mark in a URI. For example, for http://www.a.com:8080/special/docs?find_this, the query string of the URI is find_this.
- Protocol—The transport protocol and version used.
- Cache finish status—The method by which a document is placed in the cache. It can be written, refreshed, or returned by an up-to-date check.
- Status code from server—The status code returned from the server.
- Route to proxy—The route used to retrieve the resource. The document can be retrieved directly, through a proxy, or through a SOCKS server.
- Transfer time—The length of time of the transfer, in seconds or milliseconds.
- Header length from server response—The length of the header from the server response.
- Request header size from proxy to server—The size of the request header from the proxy to the server.

- Response header size sent to client—The size of the response header sent to the client.
- Request header size received from client—The size of the request header received from the client.
- Content-length from proxy to server request—The length, in bytes, of the document sent from the proxy to the server.
- Content-length received from client—The length, in bytes, of the document received from the client.
- Content-length from server response—The length, in bytes, of the document from the server.
- Unverified user from client—The user name given to the remote server during authentication.

7. If you choose a custom format, type it in the “Custom format” field.
8. If you don’t want to log client access from certain host names or IP addresses, type them in the host names and IP Addresses fields. Type a wildcard pattern of hosts from which the server should not record accesses. For example, *.netscape.com doesn’t log accesses from people whose domain is netscape.com. You can type wildcard patterns for host names, IP addresses, or both.
9. Choose whether to include the format string in the logfile. If you are using the proxy server’s log analyzer, you should include a format string. If you are using a third-party analyzer, you may not want to include a format string in your logfile.
10. Click OK.

Working with the Log Analyzer

Use the log analyzer to generate statistics about your server, such as a summary of activity, most commonly accessed URLs, times during the day when the server is accessed most frequently, and so on. You can run the log analyzer from the Server Manager, as described in “Running the Log Analyzer from the Server Manager” on page 198 or if you are using a UNIX proxy server, you can run the log analyzer from the command line. You may want to do so if you are analyzing a large log file because the process may take several hours. For more information on using this feature from the command line, go to “Running the Log Analyzer from the Command Line” on page 200.

NOTE Before running the log analyzer, you should archive the server logs. For more information about archiving server logs, see "Archiving Log Files" on page 201.

If you use the extended or extended-2 logging format, the log analyzer generates several reports within the output file in addition to the information that you designate to be reported. The following sections describe these reports.

Transfer Time Distribution Report

The transfer time distribution report shows the time it takes your proxy server to transfer requests. This report displays the information categorized by service time and by percentage finished. The following is a sample transfer time distribution report.

```
By service time category:

   < 1 sec [64.4%] ......................................
   < 2 sec [33.3%] ...................
   < 3 sec [ 2.7%] .
   < 4 sec [ 1.7%] .
   < 5 sec [ 0.6%]
   < 6 sec [ 0.4%]
   < 7 sec [ 0.2%]
   < 8 sec [ 0.0%]
   < 9 sec [ 0.0%]

By percentage finished:

   < 1 sec [64.4%] ......................................
   < 2 sec [97.7%] ..................................
   < 3 sec [100.4%]............................................
```

Status Code Report

The status code report shows which and how many status codes the proxy server received from the remote server and sent to the client. The status code report also provides explanations for all of these status codes. The following is a sample status code report.

```
Code       -From remote-      -To client-   -Explanation-

200        338  [70.7%]      352  [73.6%]   OK

302        33  [ 6.9%]       36  [ 7.5%]    Redirect
```

```
Code      -From remote-     -To client-   -Explanation-
304       90  [18.8%]      99  [20.7%]    Not modified
404       3  [ 0.6%]       3  [ 0.6%]     Not found
407                        5  [ 1.0%]     Proxy authorization
                                          required
500                        2  [ 0.4%]     Internal server error
504                        6  [ 1.3%]     Gateway timeout
```

Data Flow Report

The data flow report shows the data flow (the number of bytes transferred) from the client to the proxy, the proxy to the client, the proxy to the remote server, and the remote server to the proxy. For each of these scenarios, the report shows how much data was transferred in the form of headers and content. The data flow report also shows the data flow from the cache to the client. The following is a sample data flow report.

```
                              Headers     Content     Total
- Client -> Proxy.........    0 MB        0 MB        0 MB
- Proxy  -> Client........... 0 MB        2 MB        3 MB
- Proxy  -> Remote........... 0 MB        0 MB        0 MB
- Remote -> Proxy..........   0 MB        2 MB        2 MB

Approx:
- Cache  -> Client........... 0 MB        0 MB        0 MB
```

Requests and Connections Report

The requests and connections report shows the number of requests the proxy server receives from clients, the number of connections the proxy makes to a remote server (initial retrievals, up-to-date checks, and refreshes), and the number of remote connections the proxy server avoids by using cached documents. The following is a sample requests and connections report.

```
- Total requests.............     478
- Remote connections.........     439
- Avoided remote connects....      39 [ 8.2%]
```

Cache Performance Report

The cache performance report shows the performance of the clients' caches, the proxy server's cache, and the direct connections.

Client Cache

NOTE A client cache hit occurs when a client performs an up-to-date check on a document and the remote server returns a 304 message telling the client that the document was not modified. An up-to-date check initiated by a client indicates that the client has its own copy of the document in the cache.

For the client's cache, the report shows:

- **client and proxy cache hits:** a client cache hit in which the proxy server and the client both have a copy of the requested document and the remote server is queried for an up-to-date check with respect to the proxy's copy and the client's request is then evaluated with respect to the proxy's copy. The cache performance report shows the number of requests of this type that the proxy serviced and the average amount of time it took to service these requests.
- **proxy shortcut no-check:** a client cache hit in which the proxy server and the client both have a copy of the requested document and the proxy server tells the client (without checking with the remote server) that the document in the client's cache is up-to-date. The cache performance report shows the number of requests of this type that the proxy serviced and the average time it took to service these requests.
- **client cache hits only:** a client cache hit in which only the client has a cached copy of the requested document. In this type of request, the proxy server directly tunnels the client's If-modified-since GET header. The cache performance report shows the number of requests of this type that the proxy serviced and the average time it took to service these requests.
- **total client cache hits:** the total number of client cache hits and the average amount of time it took to service these requests.

Proxy Cache

A proxy cache hit occurs when a client requests a document from a proxy server and the proxy server already has the document in its cache. For the proxy server's cache hits, the report shows:

- **proxy cache hits with check:** a proxy cache hit in which the proxy server queries the remote server for an up-to-date check on the document. The cache performance report shows the number of requests of this type that the proxy serviced and the average time it took to service these requests.
- **proxy cache hits without check:** a proxy cache hit in which the proxy server does *not* query the remote server for an up-to-date check on the document. The cache performance report shows the number of requests of this type that the proxy serviced and the average time it took to service these requests.
- **pure proxy cache hits:** a proxy cache hit in which the client does not have a cached copy of the requested document. The cache performance report shows the number of requests of this type that the proxy serviced and the average time it took to service these requests.

Proxy Cache Hits Combined

For the proxy cache hits combined, the report shows:

- **total proxy cache hits:** the total number of hits to the proxy server's cache and the average amount of time it took to service these requests.

Direct Transactions

Direct transactions are those that go directly from the remote server to the proxy server to the client without any cache hits. For the direct transactions, the report shows:

- **retrieved documents:** documents retrieved directly from the remote server. The cache performance report shows the number of requests of this type that the proxy serviced, the average time it took to service these requests, and the percentage of total transactions.
- **other transactions:** transactions that are returned with a status code other than 200 or 304. The cache performance report shows the number of requests of this type that the proxy serviced and the average time it took to service these requests.

- **total direct traffic:** requests (both failed requests and successfully retrieved documents) that went directly from the client to the remote server. The cache performance report shows the number of requests of this type that the proxy serviced, the average time it took to service these requests, and the percentage of total transactions.

The following is a sample cache performance report.

```
CLIENT CACHE:
- Client & proxy cache hits... 86 reqs [18.0%] 0.21 sec/req
- Proxy shortcut no-check........ 13 reqs [ 2.7%] 0.00 sec/req
- Client cache hits only.....
- TOTAL client cache hits.......... 99 reqs [20.7%] 0.18 sec/req
PROXY CACHE:
- Proxy cache hits w/check........ 4 reqs [ 0.8%] 0.50 sec/req
- Proxy cache hits w/o check.. 10 reqs [ 2.1%] 0.00 sec/req
- Pure proxy cache hits...... 14 reqs [ 2.9%] 0.14 sec/req
PROXY CACHE HITS COMBINED:
- TOTAL proxy cache hits....... 113 reqs [23.6%] 0.18 sec/req
DIRECT TRANSACTIONS:
- Retrieved documents..313 reqs [65.5%]  0.90 sec/req 2 MB
- Other transactions.. 52 reqs [10.9%] 7.79 sec/req
- TOTAL direct traffic..365 reqs [76.4%] 1.88 sec/req 2 MB
```

Transfer Time Report

The transfer time report shows the information about the time it takes for the proxy server to process a transaction. This report shows values for the following categories:

average transaction time: the average of all transfer times logged.

average transfer time without caching: the average of transfer times for transactions which are not returned from the cache (200 response from remote server).

average with caching, without errors: the average of transfer times for all non-error transactions (2xx and 3xx status codes).

average transfer time improvement: the average transaction time minus the average transfer time with caching, without errors.

The following is a sample transfer time report.

```
- Average transaction time... 1.48 sec/req
- Ave xfer time w/o caching.. 0.90 sec/req
- Ave w/caching, w/o errors.. 0.71 sec/req
- Ave xfer time improvement.. 0.19 sec/req
```

Hourly Activity Report

For each analyzed hour, the hourly activity report shows:

- the load average
- the number of cache hits with no up-to-date check to the remote server
- the number of hits to the proxy server’s cache with an up-to-date check to the remote sever that proves that the document is up-to-date and the document is in the client cache
- the number of hits to the proxy server’s cache with an up-to-date check to the remote sever that proves that the document is up-to-date and the document is *not* in the client cache
- the number of hits to the proxy server’s cache with an up-to-date check to the remote server that caused part of the document to be updated.
- the number of hits to the proxy server’s cache with an up-to-date check to the remote server that returned a new copy of the requested document with a 200 status code.
- the number of requests for which documents are directly retrieved from the remote server without any hits to the proxy server’s cache

Running the Log Analyzer from the Server Manager

To run the log analyzer from the Server Manager:

1. In the Server Manager, choose Server Status | Generate Report.
2. Type the name of your server; this name appears in the generated report.
3. Choose whether or not the report will appear in HTML or ASCII format.
4. Select the log file you want to analyze.

5. If you want to save the results in a file, type an output filename in the Output file field. If you leave the field blank, the report results print to the screen. For large log files, you should save the results to a file because printing the output to the screen might take a long time.

6. Select whether or not to generate totals for certain server statistics. The following totals can be generated:

   - Total hits—The total number of hits the server received since access logging was enabled.
   - 304 (Not Modified) status codes—The number of times a local copy of the requested document was used, rather than the server returning the page.
   - 302 (Redirects) status codes—The number of times the server redirected to a new URL because the original URL moved.
   - 404 (Not Found) status codes—The number of times the server couldn't find the requested document or the server didn't serve the document because the client was not an authorized user.
   - 500 (Server Error) status codes—The number of times a server-related error occurred.
   - Total unique URLs—The number of unique URLs accessed since access logging was enabled.
   - Total unique hosts—The number of unique hosts who have accessed the server since access logging was enabled.
   - Total kilobytes transferred—The number of kilobytes the server transferred since access logging was enabled.
   - Total kilobytes saved by caches—The number of kilobytes that have been saved in the client cache.
   - Choose to generate general statistics.
   - Top number of one-second periods—You can specify the number of one-second periods that had the highest number of requests.
   - Top number of one-minute periods—You can specify the number of one-minute periods that had the highest number of requests.
   - Top number of one-hour periods—You can specify the number of one-hour periods that had the highest number of requests.

- Top number of users—You can specify the maximum number of users that accessed your server, provided that you included this as an item to log when you enabled access logging.
- Top number of referers—You can specify the number of referers that appear in your log analysis, provided that you included this as an item to log when you enabled access logging.
- Top number of user agents—You can specify the number of user agents that appear in your log analysis, provided that you included this as an item to log when you enabled access logging.
- Top number of miscellaneous logged items—You can specify the number of items that appear in your log, provided you included this as an item to log when you enabled access logging. These miscellaneous items include the request method, the URI, and the URI query.

7. Select whether or not to generate a list of server access statistics. You can generate a list of the following:
    - Most commonly accessed URLs—You can have the log analyzer show the most commonly accessed URLs or URLs that were accessed more than a specified number of times.
    - Hosts most often accessing your server—You can have the log analyzer show the hosts most often accessing your server or hosts that have accessed your server more than a specified number of times.
8. Specify the order in which you want to see the results.
9. Click OK.

Running the Log Analyzer from the Command Line

To analyze access log files from the command line, run flexanlg, which is in `extras/flexanlg` in your server root directory.

To run `flexanlg`, type the following command and options at the command prompt:

```
% flexanlg [ -P ] [-n name] [-x] [-r] [-p order] [-i file]* [ -m metafile ]* [-o
file][-c opts] [-t opts] [-l opts]
```

The following describes the syntax of the function. (You can get this information online by typing `flexanlg -h`.)

```
-P: proxy log format                                 Default: no
-n servername: The name of the server
-x : Output in HTML                                  Default: no
-r : Resolve IP addresses to host names               Default: no
-p [c,t,l]: Output order (counts, time stats, lists)  Default: ctl
-i filename: Input log file(s)                       Default: none
-o filename: Output log file                         Default: stdout
-m filename: Meta file(s)                            Default: none
-c [h,n,r,f,e,u,o,k,c,z]: Count these item(s) -      Default: hnreuokc
    h: total hits
    n: 304 Not Modified status codes (Use Local Copy)
    r: 302 Found status codes (Redirects)
    f: 404 Not Found status codes (Document Not Found)
    e: 500 Server Error status codes (Misconfiguration)
    u: total unique URL’s
    o: total unique hosts
    k: total kilobytes transferred
    c: total kilobytes saved by caches
    z: Do not count any items.
-t [sx,mx,hx, xx,z]: Find general stats - Default:s5m5h24x10
    s(number): Find top (number) seconds of log
    m(number): Find top (number) minutes of log
    h(number): Find top (number) hours of log
    u(number): Find top (number) users of log
    a(number): Find top (number) user agents of log
    r(number): Find top (number) referers of log
    x(number): Find top (number) for miscellaneous keywords
    z: Do not find any general stats.
-l [cx,hx]: Make a list of -                         Default: c+3h5
    c(x,+x): Most commonly accessed URLs
             (x: Only list x entries)
             (+x: Only list if accessed more than x times)
    h(x,+x): Hosts (or IP addresses) most often accessing your server
             (x: Only list x entries)
             (+x: Only list if accessed more than x times)
    z: Do not make any lists
```

Archiving Log Files

You can archive the access and error log files and have the server create new ones.

When you archive log files, the server renames the current log files and then creates new log files with the original names. You can back up or archive (or delete) the old log files, which are saved with the original filename appended with the date the file was archived. For example, access becomes
access.24-Apr.

You can archive log files immediately or have the server archive them at a specific time on specific days. The information about when to archive log files is stored in the `cron.conf` file in the `admin-serv` directory in the server root directory; the server's cron configuration options are stored in `ns-cron.conf` in the `admin-serv` directory.

NOTE Before running the log analyzer, you should archive the server logs.

To archive log files:

1. From the Server Manager, choose Server Status | Archive Log.
2. Click Archive if you want to archive the log files immediately. Or, if you want archiving to occur at a specific time on specific days, click the Rotate log at button, choose times from the list, and select the days for archiving to occur.
3. Click OK.
4. Shut down and restart the administration server.

NOTE If you chose to archive your server logs at specific times on specific days, step 4 is necessary in order for archiving to take place.

Monitoring the Server Using SNMP

You can monitor your server in real time by using the *Simple Network Management Protocol* (SNMP). SNMP is a protocol used to exchange data about network activity. With SNMP, data travels between a managed device and a network management station (NMS) where users remotely manage the network.

A managed device is anything that runs SNMP (for example, hosts, or routers). Your proxy server is a managed device. An NMS is usually a powerful workstation with one or more network management applications installed. A network management application graphically shows information about managed devices (which device is up or down, which and how many error messages were received, and so on).

Every managed device contains an SNMP *agent* that gathers information regarding the network activity of the device. This agent is known as the subagent. Each iPlanet server (except the administration server) has a subagent.

Another SNMP agent exchanges information between the subagent and NMS. This agent is called the master agent. A master agent runs on the same host machine as the subagents it talks to. You can have multiple subagents installed on a host machine. All of these subagents can communicate with the master agent.

Values for various variables that can be queried are kept on the managed device and reported to the NMS as necessary. Each variable is known as a managed object, which is anything the agent can access and send to the NMS. All managed objects are defined in a management information base (MIB), which is a database with a tree-like hierarchy. The top level of the hierarchy contains the most general information about the network. Each branch underneath is more specific and deals with separate network areas.

How Does SNMP Work?

SNMP exchanges network information in the form of protocol data units (PDUs). PDUs contain information about various variables stored on the managed device. These variables, also known as managed objects, have values and titles that are reported to the NMS as necessary. Communication between an NMS and managed device can take place in one of two forms:

NMS-initiated communication: NMS-initiated communication is the most common type of communication between an NMS and a managed device. In this type of communication, the NMS either requests information from the managed device or changes the value of a variable stored on the managed device.

These are the steps that make up an NMS-initiated SNMP session:

1. The NMS searches the server’s MIB to determine which managed devices and objects need to be monitored.
2. The NMS sends a PDU to the managed device’s subagent through the master agent. This PDU either requests information from the managed device or tells the subagent to change the values for variables stored on the managed device.

3. The subagent for the managed device receives the PDU from the master agent.
4. If the PDU from the NMS is a request for information about variables, the subagent gives information to the master agent and the master agent sends it back to the NMS in the form of another PDU. The NMS then displays the information textually or graphically.

   If the PDU from the NMS requests that the subagent set variable values, the subagent sets these values.

Managed device-initiated communication: This type of communication occurs when the managed device needs to inform the NMS of an event that has occurred. A managed device such as a terminal would initiate communication with an NMS to inform the NMS of a shut down or start up. Communication initiated by a managed device is also known as a “trap.”

These are the steps that make up a managed device-initiated SNMP session:

1. An event occurs on the managed device.
2. The subagent informs the master agent of the event.
3. The master agent sends a PDU to the NMS to inform the NMS of the event.
4. The NMS displays the information textually or graphically.

For information on setting up and configuring your server to use SNMP, see *Managing Netscape Servers*.

The Proxy Server MIB

Each iPlanet server has its own MIB (management information base). The proxy server’s MIB is a file called `ns-proxy.mib`. This MIB contains the definitions for various variables pertaining to network management for the proxy server. These variables are known as managed objects. Using the proxy server MIB and network management software, such as HP OpenView, you can monitor your web server like all other devices on your network.

The proxy server MIB has an object identifier of *netscape 1* (i.e. `http OBJECT IDENTIFIER : := { netscape 1 }`) and is located in the *server-root*`/plugins/snmp` directory.

You can see administrative information about your web server and monitor the server in real time using the proxy server MIB.

Installing Subagents on AIX

If your SNMP daemon is running on AIX, it supports SMUX. For this reason, you don't need to install a master agent. However, you do need to change the AIX SNMP daemon configuration.

AIX uses several configuration files to screen its communications. One of them, `snmpd.conf`, needs to be changed so that the SNMP daemon accepts the incoming messages from the SMUX subagent. For more information, see the online manual page for `snmpd.conf`. You need to add a line to define each subagent.

For example, you might add this line to the `snmpd.conf`:

`smux 1.3.6.1.4.1.1.1450.1 ""` *IP_address net_mask*

IP_address is the IP address of the host the subagent is running on, and *net_mask* is the network mask of that host.

NOTE Do not use the loopback address 127.0.0.1; use the real IP address instead.

If you need more information, see your related system documentation for details.

Enabling the Subagent

After you've installed the master agent that comes with your administration server, you need to enable the subagent for your web server. For more information on installing the master agent, see *Managing Netscape Servers*. You can use the Server Manager to enable the subagent.

To enable the SNMP subagent:

1. From the Server Manager, choose Server Status | SNMP Subagent Configuration. The SNMP Configuration form appears.
2. Type the name of the system that has the master agent installed on it. (The default is the local system.)
3. Type a description.
4. Type your organization name.
5. Type the web server's location.
6. Type the contact person for the web server.
7. Click the On radio button.

8. Click OK.

9. Start the subagent from the SNMP Subagent Control form. For more information on starting the subagent, see “Starting, Stopping, and Restarting the Subagent” on page 206.

Starting, Stopping, and Restarting the Subagent

Once you have enabled the subagent, you can start, stop or restart it from the SNMP Subagent Control Form.

To start, stop, or restart the subagent:

1. From the Server Manager, choose Server Status | SNMP Subagent Control. The SNMP Subagent Control form appears.

2. Click the Start, Stop, or Restart button.

Chapter 13

Proxy Error Log Messages

This chapter defines some of the errors the proxy commonly reports. They are listed alphabetically by the words of the message. The errors are categorized also by severity.

The categories of severity for proxy server error log messages are:

- **Catastrophe** is a fatal error, a software crash, or other serious error that causes the client to receive no service, partial service, or totally invalid service.
- **Failure** means something failed, the proxy handled the error, but the error may still cause the proxy to function improperly or to fail to process a request.
- **Inform** is an informational log entry.
- **Misconfig** means something was misconfigured in a configuration source such as `magnus.conf` or `obj.conf`.
- **Warning** flags something that could be a normal operational error, but may also be a more serious error such as misconfiguration (e.g., host unreachable).
- **Security** is information or a warning that indicates if there’s reason to believe that someone is trying to intrude through the proxy. This category of errors is UNIX-specific.

Proxy Error Messages

The following errors are those that commonly appear in the proxy server's error log.

Catastrophe

cache file size not in sync with cache information.

The system suddenly went down or the file system became full during the cache write, or the cache file has otherwise been truncated. Normally the proxy notices any abnormal conditions, but if an outside agent causes cache files to become corrupt, the proxy will issue this message. The corrupt cache file will be removed and a new one created during the next request.

cannot open file .../.cache-size for writing—the cache may overflow if the condition persists

The proxy failed to write the current cache size to the file that contains it. This could be a temporary condition, but if it persists the proxy will not be able to keep track of its cache size. This can cause the cache to overflow on high-impact systems. It's possible the write permissions aren't correct for the user account the proxy uses.

cannot read header section from the cache file

The cache file is truncated, or permissions are such that the cache file cannot be read. Care should be taken that the cache hierarchy is entirely readable and writable by the proxy user.

caught SIGSEGV or SIGBUS, trying to dump core in `admin/config`

The proxy encountered an internal software error. Contact iPlanet for help with this error. If you rarely encounter this error and it doesn't affect the proxy service, you can ignore this message.

failed to write cache status file

The write to the cache data directory (`CacheRoot/.mc-data`) failed. The condition might be temporary (for example, it was caused by a full file system), but if it persists, the proxy might stop caching until the situation is fixed. It's possible the permissions for the proxy's user account aren't specified correctly.

filesystem is full

The cache file system has become full. The proxy will halt any cache writes, and an attempt is made to signal the Cache Manager to activate immediately. Cache writes are resumed after the condition no longer persists. Consider allocating more space to the cache system.

filesystem permission problem in subdir `.mc-data` under `Cache Root`

The file system permissions are wrong under `CacheRoot/.mc-data`. Care should be taken that the entire CacheRoot directory and recursively all its subdirectories are readable and writable by the proxy user.

Failure

cache write aborted

Cache write was aborted because the remote server failed to send the entire document, the client disconnected, or some other error condition occurred.

called with no host name or address (or corrupt) [SSL proxy]

Proxy received an invalid SSL proxying request.

cannot create lock file ... (...)

The proxy failed to create a lock file; this might happen if the system resources are exhausted or the machine load is so high that the process holding the lock cannot get it. In the short term this error is harmless, and the proxy will automatically recover from it. However, if the condition persists for long periods of time it might cause cache overflow or other abnormal behavior. Check the permissions for the proxy's user account and the files under the cache root directory.

cannot open ... for writing -- caching disabled as long as error persists
cannot open cache output file ...
cannot open file .../.cache-size for appending - the cache may overflow if the condition persists
cannot open gc pid file -- cannot signal gc
cannot remove file ... -- may cause disk full detection to fail

The file system permissions under the cache root or the server root are wrong, and the proxy cannot open the cache file for writing. On a heavily loaded system this can also be caused by a temporary failure to do disk I/O, which means this error could be ignored unless the condition persists. In the long term, this error can cause various malfunctions of the caching subsystem.

cannot signal gc pid ...; running start-proxy to respawn gc

After the file system full condition the Cache Manager couldn't be signaled to start cleaning the cache. The proxy will automatically attempt to spawn a new cache management process.

can't create socket (...)
can't bind (...)
can't connect (...)
can't get peer name (...)
can't get socket name (...)
can't make ... connection non-blocking
can't make client socket non-blocking (...)
can't make connection to ... non-blocking
can't make identd connection non-blocking (...)
connect failed (...)
connect to ... failed (...)
timed out sending ident request

The SSL proxying module or the SOCKS daemon couldn't successfully execute the system call in question, as part of establishing a connection to either a remote host, or a remote identity daemon (identd).

can't connect to identd at ... -- access denied

Remote host is not running the identity daemon, and strict identity check is enabled.

can't connect to identd at ... -- error ignored

Remote host is not running the identity daemon, but loose identity check allows the request to be serviced.

can't locate host ...

The SSL proxy module is unable to locate the remote host.

connection timed out after ... seconds idle

The SSL proxy or SOCKS connection has been idle too long.

content-length mismatch; too many bytes received

The proxy received an incorrect amount of data while it was writing to the cache file. This is due to erroneous behavior on the remote server side and causes the cache file to be discarded.

disconnected by client/server/timeout/internal error condition with ... bytes in/outbound data undelivered

SOCKS or SSL proxying connection was terminated prematurely by one of the parties before all the pending data was transferred.

internal netlib timeout; process terminated

Proxy retrieval lasted too long. This is an internal timeout that cleans up processes that suddenly get blocked due to an unexpected error in the network or one of the proxy subsystems.

method without URI

The client sent an invalid proxy request.

no port number specified for host...
bad port number specified in ...

Proxy received an invalid SSL proxying request (CONNECT method with no or a bad port number).

proxy retrieve failed: ...

A generic message when the retrieval failed due to a mistyped URL, a nonexisting host name, unreachable host or network, a disabled or overloaded server, or other unexpected network error.

proxy timeout; closing connection

The proxy didn't receive any data from the remote server in the proxy timeout period.

remote closed the connection prematurely (timeout?)

The remote server closed the connection before all the data was received, causing the cache write to abort and discard the cache file.

select over the two connections failed (...)

Unexpected error in SOCKD or SSL proxying module while passing data through the proxy.

while scanning proxy HTTP headers, ...

An error occurred while reading the request headers from the client.

... already locked

The cache file is already locked -- another process is already writing the cache file. This is merely informative, not an error.

cache-size sync in progress; abandoning scheduled sync
cache-size sync lock timed out; breaking it
removing timed-out lock file...
signalled gc to start immediately

These errors are informative and self-explanatory.

Warning

terminated, shutting down

Proxy was shut down by the TERM signal.

Security

cannot attempt to access the proxy as a normal HTTP server, URL: ...

Attempt to access the proxy as a normal server; this is usually simply a mistyped admin URL but might also reflect somebody trying to intrude in the local file system of the proxy using evil URLs, such as the ones containing **/../**. Netscape guards against any such attempts, and accessing the local file system is impossible, except for the admin interface.

denying service of ...

Service was denied by configuration.

SOCKS Error Messages

The SOCKS log file contains both error and access messages. The following are the error messages that may appear in this log.

accept failed on the bound socket (...)

The SOCKS daemon failed to establish the connection from a remote server, requested by the client (SOCKS BIND request).

fatal: error in config file

The configuration file had one or more errors (listed earlier in the log file) that made it futile to start up the SOCKS server

fatal: can’t create listening socket

A TCP socket could not be created.

fatal: can’t bind to socks port

Another application or daemon is using the SOCKS port.

fatal: can’t listen at socket

An internal error occurred during startup.

error: unknown request type 0x0D from *host name:port number*

Someone tried to use the SOCKS server for something that does not use the SOCKS protocol.

error: auth: can’t open password file /etc/*filename* !

The specified password file does not exist.

error: illegal route: *route*

The route specified in the configuration file isn’t a valid IP address or interface.

error: unknown field in config: *text*

Something in the configuration file unrecognized.

error: can’t open config file ’/etc/*filename*’

The SOCKS server cannot open the specified configuration file.

error: ldap: can’t authenticate to server (*specific reason*)

The bind DN or password was rejected by the LDAP server.

error: ldap: can’t connect to *servername:port*

The specified LDAP server did not answer.

error: ldap: failed LDAP close (*specific reason*)

The SOCKS server could not close the connection to the LDAP server

error: ldap: server is down -- turning off LDAP auth

The LDAP server has vanished and ns-sockd cannot get in touch with it. ns-sockd will try to contact the LDAP server every few minutes, and once it is contacted, will enable LDAP authentication.

warning: ident: request from *host name:port number* is *some text*

The RFC 1413 ident response from that client was *some text*, not the user name

warning: auth: user *user name* tried to auth as *user name*

The user tried to authenticate as a user name even though the ident response was another user name

warning: socks4 request from *host name:port number* can't authenticate

The configuration file specifies that user name/password authentication is required for this connection. However, the client is using SOCKS4 and cannot authenticate that way. Thus, the client's request is denied and the SOCKS server logs a warning.

warning: request from *host name:port number* arrived via bad route!

A request arrived from the wrong interface meaning that someone is spoofing an IP address, or the route information in the configuration file is wrong.

warning: request from *host name:port number* failed ident check

The client did not send the required ident response, so the connection was dropped.

warning: passwd file: line *number* is bad

The format of the SOCKS5 password file is incorrect at or near the specified line.

Chapter 14

Understanding Encryption and SSL

iPlanet servers use an encryption system called Secure Sockets Layer (SSL) to ensure privacy when communicating with other SSL-enabled products, such as Netscape Navigator and Netscape Communicator.

For a complete discussion of encryption and SSL, see *Managing Netscape Servers*.

What is Encryption?

Encryption is the process of transforming information so it can't be decrypted or read by anyone except the intended recipient. This encrypted information is called *ciphertext*. It is the ciphertext that you send across the network. For example, say you have a financial report stored at your web site. If SSL is enabled on your server, your server encrypts the report and sends the ciphertext to a client, who then decrypts the ciphertext back into the financial report.

Decryption reverses the process, turning the ciphertext back into the original message. The recipient is the only one who can do this because only the recipient has a *key* to "unlock" a message.

Using Encryption in the Proxy Server

Before you can use encryption in your proxy server, you must first use the administration server to get and install a certificate. See *Managing Netscape Servers* for information on setting up certificates.

There are two scenarios in which the proxy server deals with encrypted data; it can tunnel encrypted data in a forward proxy scenario, or use encryption in a secure reverse proxy scenario. For more information on SSL tunneling, see “Tunneling SSL through the Proxy Server” on page 216. For more information on secure reverse proxying, see “Secure Reverse Proxying,” on page 78.

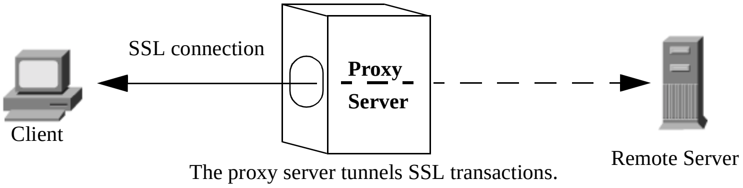
What is SSL?

SSL (Secure Sockets Layer) is a communication system that ensures privacy when communicating with other SSL-enabled products. Technically speaking, SSL is a protocol that runs above TCP/IP and below HTTP or other top-level protocols. It is symmetric encryption nested within public-key encryption, authenticated through the use of certificates. An SSL connection can only occur between an SSL-enabled client and an SSL-enabled server. In fact, when a server is running in SSL mode, it can only communicate through SSL. For more information on SSL, see *Managing Netscape Servers*.

Tunneling SSL through the Proxy Server

When you are running a proxy in the forward direction and a client requests an SSL connection to a secure server through the proxy, the proxy opens a connection to the secure server and then simply copies data in both directions without intervening in the secure transaction. This process is known as *SSL tunneling*.

Figure 14-1 With an SSL connection, the proxy can’t view the data it transfers.



To use SSL tunneling with HTTPS URLs, the client must support both SSL and HTTPS (such as the Netscape Navigator). HTTPS is implemented using SSL with normal HTTP. Clients without HTTPS support can still access HTTPS documents using Netscape Proxy’s HTTPS proxying capability.

SSL tunneling is a lower-level activity that doesn't affect the application-level (HTTPS). SSL tunneling is just as secure as SSL without proxying; the existence of the proxy in between does not in any way compromise security or reduce the functionality of SSL.

With SSL, the data stream is encrypted, so the proxy has no access to the actual transaction. Consequently, the access log cannot list the status code or the header length received from the remote server. This also prevents the proxy, or any other third party, from eavesdropping on the transactions.

NOTE At a later date, when more protocols are enhanced with SSL, you can use the standard port numbers for those protocols.

Because the proxy never sees the data, it can't verify that the protocol spoken between the client and the remote server is SSL. This means the proxy also can't prevent other protocols from being passed through. You should restrict SSL connections to only well-known SSL ports, namely port number 443 for HTTPs and 563 for SNEWS, as assigned by the Internet Assigned Numbers Authority (IANA). If there are sites that run the secure server on some other port, you can make explicit exceptions to allow connections to other ports on certain hosts. You would do this using the `connect://.*` resource.

The SSL tunneling capability is actually a general, SOCKS-like capability that is protocol-independent, so you can use this feature for other services, too. Netscape Proxy Server can handle SSL tunneling for any application that has SSL support, not only the HTTPS and SNEWS protocols.

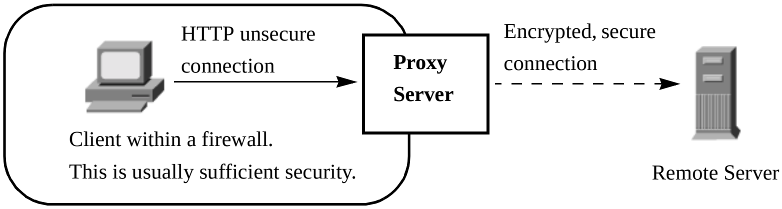
What is HTTPS?

HTTPS is normal HTTP wrapped in a secure SSL layer. If you use Netscape Navigator (or other SSL-enabled browsers) when accessing the proxy server, HTTPS URLs are handled by using the SSL tunneling feature, *not* using the HTTPS proxy feature.

NOTE Netscape Navigator doesn't use this proxy HTTPS option because it fully supports HTTPS and SSL proxying.

Clients without native HTTPS support or without SSL tunneling support can use Netscape Proxy Server's direct HTTPS proxying feature. HTTPS proxying is similar to proxying other protocols, such as HTTP or FTP. In the HTTPS case, the protocol spoken between the client and the proxy is always HTTP, but only the proxy establishes the secure connection to the remote server. That is, transactions between the proxy server and the remote server are encrypted, while the transactions between the client and the proxy are sent in the clear.

Figure 14-2 The proxy establishes the secure connection if the client doesn't support HTTPS.



This means that in order to achieve maximum security, the network between the client and the proxy must be secure (or trusted) because documents are passed unencrypted between the proxy and the client. For example, an organization's network behind a firewall could be considered secure because outsiders have no access to the internal network. And, transactions outside of the organization's network are encrypted.

Enabling HTTPS Proxying

If you are using the proxy server as a secure reverse proxy, you need to enable HTTPS proxying. To enable HTTPS proxying:

1. In the Server Manager, choose Routing|Enable, Disable.
2. Select the https://.* template from the list of existing templates. If you want to allow connections to other ports, you can use similar URL patterns in a template.
3. Click Enable proxying of this resource.
4. Click OK.

Enabling SSL on Your Server

By enabling SSL on your server, you are allowing your server to act as a secure reverse proxy. To enable SSL on your server, you must complete these steps:

1. Generate your server's key pair (public and private keys). You can complete this step using the administration server forms. See *Managing Netscape Servers* for information on generating key pairs.
2. Request a certificate from a CA. You can complete this step using the administration server forms. See *Managing Netscape Servers* for information on requesting certificates.
3. Install the certificate the CA sends to you. You can complete this step using the administration server forms. See *Managing Netscape Servers*for information on installing certificates.
4. Activate SSL for your server. You can complete this step using the proxy server forms. See "Activating SSL" on page 219 for more information on activating SSL.

Activating SSL

After your certificate is installed, you can activate SSL for your server.

1. Shut down the proxy server.
2. In the Server Manager, choose Server Preferences | Encryption On / Off. The Encryption On / Off form appears.
3. Click the On button.

NOTE The only times you will want to enable the "Initialize certificates only" radio button are if you are running an unsecure reverse proxy where you have a secure connection to the content server and the content server authenticates the proxy, or if HTTPS proxying is enabled (as opposed to SSL tunneling). Initializing certificates is necessary in this instance so that the proxy will authenticate to the client even though incoming SSL is not enabled. For more information on secure reverse proxying, see "Secure Reverse Proxying," on page 78. For more information on client authentication and the scenario listed above, see "Client authentication in a reverse proxy" in this chapter.

4. Type the port number you want your server to use if it's different from the one you specified upon setup.
5. From the Alias pull-down, select the alias you want to use for the encryption.
6. Click OK. Save and apply your changes.

NOTE	Often, you want your server to run with SSL enabled. You might, at other times, want to disable it. If you temporarily disable SSL, make sure you re-enable it before processing transactions that require confidentiality, authentication, or data integrity.

Now that SSL is enabled on your server, you can configure your overall SSL preferences (page 220), and specify different strengths of encryption for different parts of your server (page 222).

Setting Encryption Preferences

You can set a number of system-wide preferences for SSL. To do so, choose Server Preferences | Encryption Preferences in the Server Manager. After you make your changes, click OK and confirm your changes. You can configure settings for SSL version, client certificates, and ciphers.

SSL Version

You can specify which versions of SSL your server can communicate with. The latest and most secure version is SSL version 3, but many older clients use only SSL version 2. You will probably want to enable your server to use both versions.

Client Certificates

You can refuse any client that doesn't have a client certificate from a trusted CA. This differs from access control in that all requests must be through SSL connections and they must be from clients who have certificates from trusted CAs. If your server is running on an internal company intranet, you might have an internal CA, so all your clients would have certificates. In this case you would allow only people with client certificates to connect to your server. For more information on trusted CAs, see *Managing Netscape Servers*.

Ciphers

A *cipher* is an algorithm used in encryption. Some ciphers are more secure, or *stronger*, than others. Generally speaking, the more bits a cipher uses during encryption, the harder it is to decrypt the data. The list of available ciphers doesn't appear on the Encryption Preferences form unless you've enabled SSL.

When initiating an SSL connection with a server, a client lets the server know what ciphers it prefers for encrypting information. In any two-way encryption process, both parties must use the same ciphers. Because a number of ciphers are available, your server needs to be able to use the most popular ones.

You can choose ciphers from the SSL 2 protocol, as well as from SSL 3. To specify which ciphers your server can use, check them in the list. Unless you have a compelling reason not to use a specific cipher, you should check them all.

The SSL 2.0 ciphers are:

- RC4 cipher with 128-bit encryption and MD5 message authentication. RC4 ciphers are the fastest ciphers. This cipher, because it has 128-bit encryption, is the second strongest cipher next to Triple DES (Data Encryption Standard) with 168-bit encryption. It has approximately 3.4 * 10^{38} possible keys, making it very difficult to crack. As added security, all SSL 2.0 ciphers use MD5 (Message Digest 5) message authentication. MD5 message authentication detects attempts to modify data while it is in transit.

- RC4 cipher with 40-bit encryption and MD5 message authentication. This cipher is also an RC4 cipher, making it one of the fastest available ciphers. It has 40-bit encryption, which has approximately 1.1 * 10^{12} (a trillion) possible keys, making it easier to crack than encryption with more possible keys, such as 128-bit encryption. This cipher also uses MD5 message authentication to detect attempts to modify data in transit.

- RC2 cipher with 128-bit encryption and MD5 message authentication. The RC2 ciphers are slower than the RC4 ciphers. This RC2 cipher, because it has 128-bit encryption, is the second strongest cipher next to Triple DES with 168-bit. It has approximately 3.4 * 10^{38} possible keys, making it very difficult to crack. This cipher also uses MD5 message authentication to detect attempts to modify data in transit.

- RC2 cipher with 40-bit encryption and MD5 message authentication. This cipher is also an RC2 cipher, making it is slower than the RC4 cipher. It has 40-bit encryption, which is not as strong as 168-bit, 128-bit, or 56-bit encryption. 40-bit encryption has approximately 1.1 * 10^{12} (a trillion) possible keys. This cipher also uses MD5 message authentication to detect attempts to modify data in transit.

- DES with 56-bit encryption and MD5 message authentication. DES (Data Encryption Standard) is a U.S. government standard for data encryption. This cipher does not have as many possible keys as does 128-bit encryption, and therefore is not as strong. 56-bit encryption has approximately 7.2 * 10^{16} possible keys. This cipher also uses MD5 message authentication to detect attempts to modify data in transit.
- Triple DES with 168-bit encryption and MD5 message authentication. Triple DES is the strongest cipher available, but it is not as fast as RC4. Triple DES uses a key three times as long as the key for standard DES. Because the key size is so large, there are more possible keys than for any other cipher - approximately 3.7 * 10^{50}. This cipher also uses MD5 message authentication to detect attempts to modify data in transit.

The SSL 3.0 ciphers are:

- RC4 with 128-bit encryption and MD5 message authentication. This cipher is the same as the SSL 2.0 version of RC4 with 128-bit encryption but uses a more secure implementation of MD5 message authentication to detect attempts to modify data in transit.
- RC4 with 40-bit encryption and MD5 message authentication. This cipher is the same as the SSL 2.0 version of RC4 with 40-bit encryption but uses a more secure implementation of MD5 message authentication to detect attempts to modify data in transit.
- Triple DES with 168-bit encryption and SHA message authentication. This cipher is the same as the SSL 2.0 version of Triple DES with 168-bit encryption, but uses SHA (Secure Hash Algorithm) message authentication instead of MD5 message authentication. SHA is a government standardized algorithm that is used to construct a message authentication code that detects attempts to modify data while it is in transit. SHA is slower than MD5, but it is stronger.
- DES with 56-bit encryption and SHA message authentication. This cipher is the same as the SSL 2.0 version of DES with 56-bit encryption but uses SHA message authentication instead of MD5 message authentication.
- RC2 with 40-bit encryption and MD5 message authentication. This cipher is the same as the SSL 2.0 version of RC2 with 40-bit encryption but uses a more secure implementation of MD5 message authentication to detect attempts to modify data in transit.
- No encryption, only MD5 message authentication. This cipher uses only MD5 message authentication to secure data. Any data sent using this cipher is not encrypted. The data is protected from modification, but it can be viewed by eavesdroppers.

CAUTION You might not want to check “No Encryption, only MD5 message authentication”. If no other ciphers are available on the client side, the server will use this, and no encryption will occur.

Keeping Clients from Caching SSL Files

Pre-encrypted files can be prevented from being cached by a client by adding the following line inside the head statement of a file in HTML:

```
<meta http-equiv="pragma" content="no-cache">
```

Configuring SSL Tunneling

To tunnel SSL, you need to configure your proxy server to do so. To configure SSL proxying,

1. In the Server Manager, choose Routing | Enable, Disable.
2. Select the `connect://.*:443` resource from the list of existing templates. The `connect://` method is an internal proxy notation and doesn’t exist outside of the proxy. See the following sidebar text for more information on connect.

   If you want to allow connections to other ports, you can use similar URL patterns in a template.
3. Click “Enable proxying of this resource”.
4. Click OK, and then restart the proxy.

CAUTION If the proxy is misconfigured, it is possible to abuse the SSL proxy to achieve “telnet-hopping.” Someone can use the proxy to make it appear that a telnet connection is coming from the proxy host, rather than the actual connecting host. This is why you have to pay extra attention to allow no more ports than absolutely necessary and to use access control on your proxy (restricting the client hosts).

SSL tunneling protocol: technical details

Internally, SSL tunneling uses the CONNECT method with the destination host name and port number as a parameter followed by an empty line:

```
CONNECT energy.netscape.com:443 HTTP/1.0
```

A successful response from the proxy server is

```
HTTP/1.0 200 Connection established
Proxy-agent: iPlanet-Proxy/3.6
```

followed by an empty line. The connection is then set up between the client and the remote server, and they can transfer data in both directions until either closes the connection.

Internally, to benefit from the normal configuration mechanism based on URL patterns, the host name and port number (energy.netscape.com:443) are automatically mapped into a URL like this:

```
connect://energy.netscape.com:443
```

`connect://` is only an internal notation used by iPlanet Web Proxy Server to make configuration easier and uniform with other URL patterns. Outside of the proxy server, `connect` URLs do not exist, and if the Netscape Proxy receives such a URL from the network, it marks it as invalid and refuses to service the request.

Increasing Server Security

There are other security risks besides someone trying to break your encryption. The modern network faces risk from external and internal hackers, using a variety of tactics to gain access to your server and the information on it.

So in addition to enabling SSL on your server, you should take extra security precautions. The following list describes the most important things you can do to make your server more secure. For more information on server security see *Managing Netscape Servers.*

- Limit physical access. Keep the server machine in a locked room that only authorized people can enter. This prevents anyone from hacking the server machine itself.

- Limit administration access. If you plan on remotely configuring your server, be sure to use your administration server's access control to allow administration from a very small number of locations. You should also make the administrative connection a mandatory SSL connection.
- Choose good passwords. It's important to choose passwords that are difficult to guess and never to reveal them to anyone. Your most important passwords should not contain words from any language because numerous password-cracking programs exist that can run through millions of possible word combinations in seconds. Your important passwords also should be at least eight characters long, and contain a mix of uppercase and lowercase letters, punctuation marks, mathematical symbols, or numerals.
- Secure your private key. Make sure your private key file is protected. Store the key file in a directory that only you or authorized administrators have access to. It's also important to know whether the file is stored on backup tapes or is otherwise available for someone to intercept. If so, you must protect your backups as much as you protect your server.
- Limit other applications on the server. It's possible to circumvent your server's security by exploiting holes in other programs running on your server. Therefore, it is wise to disable all unnecessary programs and services. Some applications that you should be cautious of are telnet, rlogin, and rdist. Also be careful about which CGI, Java, and JavaScript programs are on your server.
- Limit ports. Disable extraneous services operating on ports for the machine. Use routers or firewall configurations to prevent incoming connections from accessing services available on different ports.
- Know your server's limits. A server can't control the security of information once it reaches the client, nor can it control which individuals have access to directories and files on the server. Therefore, it is your responsibility to secure any information clients send to you through SSL.

What is Client Authentication?

To establish identity over the Internet or intranet, clients and servers use a digital file known as a certificate. Certificates are exchanged between parties prior to communication to ensure that both parties are actually who they claim to be. When a client sends a certificate to a server, the process is called *client authentication*.

Client authentication is not essential to an SSL connection, but it does give extra assurance to both parties that they are sending encrypted information to the correct parties. You can use client authentication in a reverse proxy to make sure that your content server does not share information with unauthorized proxies or clients.

Client Authentication in a Reverse Proxy

In a reverse proxy, you can set up client authentication according to any one of the following scenarios:

- **Proxy authenticates client -** This scenario enables you to allow access to all clients with acceptable certificates or to allow access to only those clients who have acceptable certificates and are recognized users on the access control list for your proxy.
- **Content server authenticates proxy -** This scenario enables you to make sure that your content server is actually connecting with your proxy and not some other server.
- **Proxy authenticates client and content server authenticates proxy -**This scenario provides the maximum security and authentication for your reverse proxy.

For information on how to set up these scenarios, see “Setting up Client Authentication in a Reverse Proxy” on page 226 of this chapter.

Setting up Client Authentication in a Reverse Proxy

Client authentication in a secure reverse proxy provides further insurance that your connections are secure. The following instructions explain how to set up client authentication according to the scenario you choose.

NOTE	Each scenario assumes that you have both a secure client to proxy connection and a secure proxy to content server connection.

Proxy Authenticates Client

To set up this scenario:

1. Follow the directions for setting up the secure client to proxy and secure proxy to content server scenario in the section entitled “Setting up a Secure Reverse Proxy,” on page 84.
2. In the Server Manager, choose Server Preferences | Encryption Preferences. The Encryption Preferences form will appear.

   If you want to permit access to all users with valid certificates:

   a. Click the Yes radio button for “require client certificates.”

      If you want to permit access to only those users who have both valid certificates and are specified as acceptable users in access control:

   b. Click the No radio button for “require client certificates.”

   c. Go to the Server Manager and choose Server Preferences | Restrict Access.

   d. Turn on Access Control.

   e. Set your proxy to authenticate as a reverse proxy.

   f. Click the Permissions button for the first access type, which is GET, HEAD, POST, INDEX, CONNECT.

   g. List any permissions making sure that for all of them, you identify “Client Certificate” as the authentication method. For more information on setting up access control, see Chapter 5, “Controlling Access to Your Server.”
3. Save and apply your changes.
4. Restart the proxy from the command line by going to the proxy directory and typing `./start`.

Content Server Authenticates Proxy

To set up this scenario:

1. Follow the directions for setting up the secure client to proxy and secure proxy to content server scenario in the section entitled “Setting up a Secure Reverse Proxy,” on page 84.
2. On your content server, turn on client authentication.

NOTE You can modify this scenario so that you have an unsecure client connection to proxy, secure connection to content server, and the content server authenticates proxy. To do so, you need to turn off encryption and tell the proxy to initialize certificates only.

To initialize certificates only:

1. In the Server Manager, choose Server Preferences | Encryption On / Off. The Encryption On / Off form will appear.
2. Click the Initialize Certificates Only radio button.
3. Save and apply your changes.
4. Restart the proxy from the command line. by going to the proxy directory and typing `./start`.

Proxy Authenticates Client and Content Server Authenticates Proxy

To set up this scenario,

1. Follow the above directions for setting up the proxy authenticates client scenario.
2. On your content server, turn on client authentication.

Effects of an SSL-Enabled Server

This section describes what effects you need to know about while running an SSL-enabled server.

Secure URL Construction

URLs to an SSL-enabled server are constructed using `https` instead of simply `http`. URLs that point to documents on an SSL-enabled server have this format:

`https://`*servername.domain.dom/pathname/document*

Secure Server Document Root

After SSL is installed and enabled on a server, all communications between the server and SSL-enabled browsers (such as Netscape Navigator) are private, authenticated, and checked for message integrity. This means that any document sent to a user with an SSL-enabled browser is automatically encrypted. The only way around this is discussed in “Setting Encryption Preferences” on page 220.

NOTE Browsers not enabled with SSL can't communicate with an SSL-enabled server because they can't enter an SSL transaction. However, they can communicate with the server when the server isn't using SSL.

The Secure Log

Once SSL is enabled, a new log file, (`secure.log`) is created in the normal log directory. Entries in the log look like:

```
198.93.92.99: [02/Nov/1994:23:51:46 -0800] using keysize 40
```

where the IP address is first, followed by the date and time of access, and then the key size. The key size represents a level of security. Generally, the bigger the key size, the higher the level of security. See page 221 for a list of supported key sizes.

Unprotected Server Document Directory

If you want to have both secure and unsecure servers, you should operate the unsecure server on a different machine from the secure server. If your resources are limited and you must run an unsecure server on the same machine as your secure server, do the following.

- Assign proper port numbers—Make sure that the secure server and the unsecure server are assigned different port numbers. For example, use 443 for the secure server and 80 for the unsecure one.

NOTE Use CHROOT on the document root directory—The unprotected server should have references to its document root redirected using the `chroot` command.

Changes to the magnus.conf File

With an SSL-enabled server installed, there are several changes to the `magnus.conf` file (the server's main configuration file). These new directives are briefly described in the following sections.

Security

The Security directive tells the server whether SSL is enabled or disabled.

Syntax

```
Security value
```

`value` specifies if SSL is on or off. `Security on` enables SSL; `Security off` disables SSL.

SSL2

The SSL2 directive tells the server that SSL2 is enabled or disabled.

Syntax

```
SSL2 value
```

`value` specifies if SSL version 2 is enabled or disabled. `SSL2 on` enables SSL 2. `SSL2 off` disables SSL 2.

SSL3

The SSL3 directive tells the server that SSL3 is enabled or disabled.

Syntax

```
SSL3 value
```

`value` specifies if SSL version 3 is enabled or disabled. `SSL3 on` enables SSL 3. `SSL3 off` disables SSL 3.

Keyfile

The Keyfile directive tells the server where the key file is located.

Syntax

```
Keyfile keyfile
```

`keyfile` is the server's key file, specified as a relative path from the server root or as an absolute path.

Certfile

The Certfile directive specifies where the certificate file is located.

Syntax

```
Certfile certfile
```

`certfile` is the server's certificate file, specified as a relative path from the server root or as an absolute path.

Ciphers

The Ciphers directive specifies the ciphers enabled for your server. For a discussion of these ciphers, refer to “Setting Encryption Preferences” on page 220.

Syntax

```
Ciphers +rc4 +rc4export -rc2 -rc2export +idea +des +desede3
```

A + means the cipher is active, and a - means the cipher is inactive. Any cipher with `export` as part of its name is not stronger than 40 bits.

SSL3Ciphers

The SSL3Ciphers directive specifies the SSL 3 ciphers enabled for your server. For a discussion of these ciphers, refer to “Setting Encryption Preferences” on page 220.

Syntax

```
SSL3Ciphers +rsa_rc4_128_md5 +rsa_3des_sha +rsa_des_sha
+rsa_rc4_40_md5 +rsa_rc2_40_md5 -rsa_null_md5
```

A + means the cipher is active, and a - means the cipher is inactive. Any cipher with `40` as part of its name is 40 bits.

SSLClientAuth

The SSLClientAuth directive specifies whether a client must have a certificate in order to communicate with the server. Note that you don’t need to turn this directive on to use client authentication with access control.

Syntax

```
SSLClientAuth value
```

`value` specifies if certificates are always required. `SSLClientAuth on` requires certificates. `SSLClientAuth off` specifies that certificates are not required.

Chapter 15

Tuning Server Performance

This chapter explains how to tune your server's performance using the online forms as well as the configuration files. It also provides recommendations for performance tuning. By tuning your server's performance parameters, you can optimize the speed and efficiency of your proxy server.

Using Timeouts Effectively

Timeouts have a significant impact on server performance. Setting the optimal timeout for your proxy server will help to conserve network resources.

Read Timeout

The *read timeout* is the number of seconds the proxy server will wait for an incoming request. The default value for this timeout is 60 seconds. Setting the timeout a few seconds shorter will allow proxy resources to be released faster if the connection stays idle (e.g., if you telnet to the proxy port and let it just sit there).

You can view or modify the read timeout on the Proxy Tuning form. You can access this form by choosing System Setting | Tuning from the Server Manager.

CAUTION The read timeout value should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Proxy Timeout

The proxy timeout tells the server how long to wait before aborting an idle connection. A high proxy timeout value commits a valuable proxy process to a potentially dead client for a long time. A low timeout value will abort CGI scripts that take a long time to produce their results, i.e., a database query gateway.

To determine the best proxy timeout for your server, you should consider these issues:

- Will your proxy be handling many database queries or CGI scripts?
- Will your proxy server be handling a small enough amount of requests that it can spare a process at any given time?

If you answered yes to the above questions, then you may decide to set a high proxy timeout value. The highest proxy timeout value recommendedis 1 hour. You can view or modify the proxy timeout value on the System Specifics form. You can access this form by choosing Server Preferences | System Specifics from the Server Manager.

Timeout After Interrupt

The timeout after interrupt value tells the proxy how much time it has to continue writing a cache file after a client has aborted the transaction. In other words, if the proxy server has almost finished caching a document and the client aborts the connection, the proxy can continue caching the document until it reaches the timeout after interrupt value.

To determine the best timeout after interrupt value for your proxy server, use `sitemon,` as described below. If `sitemon` reports that the proxy server is using too much of its process pool, you should reduce your timeout after interrupt accordingly. The highest recommended timeout after interrupt value is 5 minutes.

You can set the timeout after interrupt value on the System Specifics form. You can access this form by choosing Server Preferences | System Specifics from the Server Manager.

The `sitemon` utility is located in the `/extras/proxy` directory and can be invoked as follows:

```
sitemon [-p port] [-a addr] [-u sec] [-c] [-t] [-n]
```

port is the server port number. By default, this is 8080.

addr specifies the server IP address (BIND address) that is set using Server-Preferences. If the server IP address has not been set, the IP address of the host where proxy server is installed is taken as the default value.

sec specifies the screen update interval in seconds.

`-c` specifies that the display is in curses mode (default)

`-t` specifies that the display is in plain text mode

`-n` specifies that the twirling indicator will not be used

If the two parameters, *addr* and *port* are not set to their default values, then you must explicitly specify their values, as shown in the following example:

```
./sitemon -a 129.158.224.48 -p 8085
```

Keep-Alive Timeout

The *keep-alive timeout* is the number of seconds the proxy server will wait for the next request from a keep-alive connection. The default value for this timeout is 5 seconds. Shorter timeouts let the proxy release resources that are tied up by idle keep-alive connections. Longer timeouts will make keep-alive more effective, but can tie up valuable proxy resources, and thus, bog down the server. You should not use the keep-alive feature because it can easily cause problems when the entire process pool is tied up by idle keep-alive connections. This problem can occur quickly - especially with clients that open multiple simultaneous connections, such as Navigator, which by default, opens 4 connections.

You can view or modify the keep-alive timeout on the Proxy Tuning form. You can access this form by choosing Server Preferences | Tuning from the Server Manager.

CAUTION The keep-alive timeout value should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Global Netlib Timeout

The *global netlib timeout* is the absolute maximum number of seconds that the proxy will wait for any HTTP, FTP, Gopher or HTTPS retrieval. The default value for this timeout is 10800 seconds (3 hours). You may want to increase this number if you are using a modem to download large applications.

You can view or modify the global netlib timeout on the Proxy Tuning form. You can access this form by choosing System Setting | Tuning from the Server Manager.

CAUTION The global netlib timeout value should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Stall Timeout Override

You should not change this value under any circumstances.

Controlling Up-to-Date Checks

For the sake of performance, it is not recommended that you configure your proxy server to check if a cached document is up-to-date each time that document is requested. Frequent up-to-date checks may unnecessarily consume network resources. Therefore, you may not want to have your server perform up-to-date checks all of the time. To improve the server's performance while ensuring that a document is up-to-date, choose a reasonable document lifetime in conjunction with the last-modified factor. For more information on the last-modified factor, see "Setting the Last-modified Factor" on page 236.

You should set an up-to-date check range between 8 and 24 hours.

For more information on controlling up to date checks, see Chapter 9, "Caching."

Setting the Last-modified Factor

The last-modified factor is a fraction which is multiplied by the interval between a document's last modification and the time that the last up-to-date check was performed on the document. The resulting number is compared with the time since the last up-to-date check. If the number is smaller than the time interval, the document is not expired. The last-modified factor allows you to ensure that recently changed documents are checked more often than old documents.

You should set a last-modified factor between 0.1 and 0.2. For more information on setting the last-modified factor, see Chapter 9, "Caching."

Using DNS Effectively

DNS (Domain Name Service) is the system used to associate standard IP addresses with host names. This system can tie up valuable proxy resources if not configured wisely. To optimize the performance of DNS:

- Enable DNS Caching

  You can enable DNS Caching by choosing Server Preferences | DNS Caching from the Server Manager and selecting the enabled radio button for DNS caching. For more information on DNS caching, see "Understanding DNS Caching" on page 47.

  You can specify whether or not your DNS cache file is visible with the *DNS cache file visible value* on the Proxy Tuning form. By default, the cache file is invisible. You can view or modify the DNS cache file visible value on the Proxy Tuning form. You can access this form by choosing Sever Preferences | Tuning from the Server Manager.

- Do not log client DNS names

  You can disable client DNS name logging by choosing Server Status | Log Preferences from the Server Manager and deselecting the radio button for recording client domain names. For more information on logging client domain names, see "Setting Access Log Preferences" on page 190.

- Log only client IP addresses

  You can enable client IP address logging by choosing Server Status | Log Preferences from the Server Manager and selecting the radio button for recording client IP addresses. For more information on logging client IP addresses, see "Setting Access Log Preferences" on page 190.

- Disable reverse DNS

  Reverse DNS translates an IP address into a host name. You can disable reverse DNS by choosing Server Preferences | System Specifics from the Server Manager and selecting the "No" radio button for Enable DNS.

- Avoid access control based on client host names.

  Use clients' IP addresses instead, if possible. You can configure access control by choosing Server Preferences | Restrict Access from the Server Manager. For more information on access control, see Chapter 5, "Controlling Access to Your Server

Determining the Number of Processes

On the UNIX platform, administrators must specify the number of processes that will be pre-forked on the server. Pre-forking enhances the performance of the proxy server because the number of processes limits the concurrent requests the proxy can handle simultaneously. The chart below indicates how many processes might be necessary for a certain number of users.

Table 15-1 Processes per User

Users	Processes	Memory (MB)	Swap (MB)
0-300	32	10	64
300-500	64	20	128
500-1,000	96	30	192
Over 1,000	128	40	256

While you can estimate the number of processes you will need, it is important that you properly scale the proxy server to meet your load in the peak periods. Typically, if you have a need to tune the server, your current process allocation is insufficient. Sitemon will usually display 100% processes-in-use, but this does not show you the number of clients that are queued by the operating system.

You can estimate the number of clients in waiting by using netstat. Information on the correct command line options to list all TCP sessions can be found in your system's manual page for netstat. From this snapshot, count up all connections to port 8080, or your designated proxy port, that are in TCP states between SYN_RECVD and CLOSE_WAIT, including ESTABLISHED. Your system may vary slightly, so check your documentation. The count you now have should be a snapshot of accepted connections, those that have been established, plus any that have been queued by the system because there are not enough processes available.

Run the log analyzer for a few days to get a good distribution of load on your server. The pstats utility located in `/extras/flexanlg/pstats` is the fastest way to calculate these statistics. Run the utility when your server has a moderate load. This means that your process utilization is between 60% and 80%. It is important to consider that in peak times, under extreme load, connections will take longer than expected due to things like thrashing, swapping, or OS listen queue overload. You want to make sure that your system is not swapping, so this baseline configuration should not have more processes than can fit in RAM. For this time period, look at your average transaction time from the flex analyzer. Add 10% to this estimate if you want to be conservative.

The section titled “Processes” on page 41 contains a chart of the suggested number of processes given as a function of average number of new requests per second versus average number of seconds of service time per request. You should refer to this section for recommendations on determining the optimum number of processes for your server.

For more information on setting the number of processes, see “Processes” on page 41.

Disabling Keep-Alives

HTTP Keep-Alives are a TCP/IP feature that keeps a connection open after the request is complete so that the client can quickly reuse the open connection. For optimal server performance, it is wise to disable HTTP keep-alives.

For more information on enabling and disabling keep-alives, see “Disabling HTTP Keep-Alive” on page 49.

Using SOCKS Effectively

Using the `socks5.conf` file, you can determine the number of worker and accept threads your SOCKS server uses. These numbers will influence the performance of your SOCKS server.

Worker Threads

Worker threads perform authentication and access control for new SOCKS connections. If the SOCKS request is granted, the worker thread passes the connection to the I/O thread which passes the data outside the firewall.

If the SOCKS server is too slow, you should increase the number of worker threads. If it is unstable, decrease the number of worker threads. When changing the number of worker threads, you should start at the default number and increase or decrease as necessary.

The default number of worker threads is 40, and the typical number of worker threads falls between 10 and 150. The absolute maximum number of worker threads is 512. However, having more than 150 tends to be wasteful and unstable.

Accept Threads

Accept threads sit on the SOCKS port listening for new SOCKS requests. They pick up the connections to the SOCKS port and hand each new connection to a worker thread.

If the SOCKS server is dropping connections, you should increase the number of accept threads. If it is unstable, decrease the number of accept threads. When changing the number of accept threads, you should start at the default number and increase or decrease as necessary.

The default number of accept threads is 1, and the typical number of accept threads falls between 1 and 10. The absolute maximum number of accept threads is 512, however, having more than 60 tends to be wasteful and unstable.

Tuning FTP Listing Width

You may want to modify the width of FTP listings to better suit your needs. Increasing listing width allows longer file names and thus reduces filename truncation. The default width is 80 characters.

You can modify the FTP listing width on the Proxy Tuning form. You can access this form by choosing Server Preferences | Tuning from the Server Manager.

CAUTION The FTP listing width should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Using the Cache Effectively

You can ensure that you are effectively using your cache by architecting the cache wisely and by determining the best cache tuning settings for your cache.

Optimizing Cache Architecture

You can improve the performance of your server by architecting your cache wisely. Some suggestions to keep in mind when architecting your cache are:

- Distribute the load
- Use multiple proxy cache partitions
- Use multiple disk drives
- Use multiple disk controllers

Proper cache setup is critical to the performance of your proxy server. The most important rule to remember when laying out your proxy cache is to distribute the load. Caches should be set up with approximately 1 GB per partition and should be spread across multiple disks and multiple disk controllers. This type of arrangement will provide faster file creation and retrieval than is possible with a single, larger cache. For more information on setting up your cache, see Chapter 9, "Caching

The Cache Batch Update feature in iPlanet Web Proxy Server allows you to proactively download content from a specified web site or perform scheduled up-to-date checks on documents already in the cache. This gives you the ability to cache content in large quantities at times when traffic on the server is low. Use batch updates to download the most commonly accessed sites at the end of each business day for quick access the following morning. You can use the log files to help determine which sites are frequently accessed. For more information on batch updates, see "Using Cache Batch Updates" on page 123.

Tuning the Cache

You may need to tune your cache to improve its performance. You can view or modify all of the following settings on the Cache Tuning form.

Add'l Cch-status Values

The *add'l cch-status values* setting determines whether to use additional cache status values in the access log. You can log these additional cache status values:

DO-NOT-CACHE - pre-determined non-cacheable (by configuration)

NON-CACHEABLE - post-determined non-cacheable (by response fields)

NOT-IN-CACHE - with disconnected operation, cache miss

Currently, the add'l cch-status values logging option causes all UP-TO-DATE log entries to be reverted to NON-CACHEABLE. For this reason, tune to "original".

You can enable add'l cch-status values on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The add'l cch-status values setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Mmap on Initial Writes

The *mmap on initial writes* value determines whether to use mmap or lseek+write to create the initial CIF header. By default, this value is off. You should not change this value.

You can view the mmap on initial writes setting on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION You should not change the mmap on initial writes value.

Mmap on Cache Updates

The *mmap on cache updates* value determines whether to use memory mapped I/O when sending data from the cache and updating cache files. By default, this value is on.

You can view or modify mmap on cache updates on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The mmap on cache updates setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Use Shared Memory I/O

The *use shared memory I/O* value determines whether shared memory is enabled. The child processes communicate with the cache monitor and cache manager via either an interprocess pipe/socket, or shared memory. The shared memory based I/O is faster. By default, shared memory I/O is enabled.

You can view or modify use shared memory I/O on the Cache Tuning form. You can access this form by choosing Caching|Tune Cache from the Server Manager.

CAUTION The use shared memory I/O setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Notify Num Changes

The *notify num changes* value is used only when shared memory I/O is enabled.

Notify num changes determines the number of changes after which the cache monitor is notified about changes made by that child process. More frequent notifications keep the cache monitor up-to-date on the status and size of the cache, but create more work for the cache monitor. The valid range for notify num changes is 1 to 500, and the default value is 10.

You can view or modify the notify num changes setting on the Cache Tuning form. You can access this form by choosing Caching|Tune Cache from the Server Manager.

CAUTION The notify num changes setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Notify Blk Limit

The *notify blk limit* value is used only when shared memory I/O is enabled.

Notify blk limit determines the number of blocks that cache size usage has changed as a result of a single child process until the cache monitor is notified by that child. Smaller notification limits keep the cache monitor up-to-date on the status and size of the cache, but create more work for the cache monitor. The valid range for notify num changes is 1 to 10000 (0.5K to 5MB), and the default value is 100 (50K).

You can view or modify the notify blk limit setting on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The notify blk limit setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

C-mon Tick Interval

You should not change this value under any circumstances.

Max Mmap Size

The *max mmap size* is the maximum number of kilobytes memory-mapped at once by a single process. Files larger than the max mmap size will be memory-mapped in portions of this size. By default, the max mmap size is 256K.

The max mmap size must be at least 16, and it must be a multiple of page size, which is often four.

You can view or modify the max mmap size setting on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The max mmap size setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Min Sync Interval

The *min sync interval* specifies the amount of time the cache manager waits before traversing the entire cache to find out the actual cache size. The cache monitor attempts to maintain current information about the size of each cache section. However, the information may get skewed by unexpected software shutdowns, system crashes, power failures, etc. Another reason for skew is that a server shutdown will lose unnotified changes to the cache in child processes. Because of these skews, the cache manager periodically traverses the entire cache to determine the actual cache size.

The valid range for min-sync-interval is 1800 seconds(1/2 hour) to 604800 seconds (1 week), and the default is 86400 seconds (24 hours).

You can view or modify the min sync interval on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The min sync interval should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Notify Blk Chunk/Proc

Each server child process notifies the cache monitor about how much new cache space it has taken up (or released) by creating new cache files or updating old ones. To avoid message overflow, these notifications are not sent for every cache operation, but rather each process waits until it has reached a certain threshold before notifying the cache monitor about its activities. The *notify blk chunk/proc* variable controls the threshold which triggers each child process to report their cache usage to the cache monitor. The units are in “blocks.” A block is always 512 bytes (0.5 KB), regardless of the operating system filesystem block size.

The notify blk chunk / proc value is multiplied by the value of the MaxProcs directive. Therefore, the effect of having a higher MaxProcs setting is that child processes will buffer more cache change messages before they notify the cache monitor about them. This effect avoids message overflow on systems with a high load (large MaxProcs), and allows small caches to maintain more accurate size information (small MaxProcs).

The valid range for notify blk chunk / proc is 0 to 2000 (1MB per process), and the default value is 67 (33KB per process).

For example, the default setting 67 (33K) on a system with 64 processes means that each process buffers cache activity notifications until it has reserved 64 * 33K = 2.1MB of the cache. This means that a proxy shutdown can cause an average skew of 67MB in cache size. This condition gets corrected during the next cache size sync (see Min Sync Interval). This behavior is usually acceptable on high-end systems, with greater than 2GB of cache space.

You can view or modify the notify blk chunk / proc setting on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The notify blk chunk / proc setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Fs Full Retry After

When a server child process fails to write to disk and the error code is ENOSPC (No space left on device), the process stops writing data to that cache partition, allowing time for the garbage collector to correct the situation. The *fs full retry after* variable controls how many cache write requests should be skipped before the proxy attempts a write operation again.

The valid range for fs full retry after is 1 (retry immediately) to 1024 (practically don't retry), and the default value is 50.

You can view or modify the fs full retry after setting on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The fs full retry after setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Update After Percent

The cache monitor continuously receives messages from server child processes about how much new cache space they have used up (see variable notify-block-chunk-per-proc). To conserve CPU time, the cache monitor doesn't update its partition table and evaluate garbage collection needs after every such message. Instead, it waits until there is big enough percentage change in size. The *update after percent* value controls that percentage.

The valid range for update after percent is 0 (every time) to 100 (after size doubles), and the default value is 1.

You can view or modify the update after percent setting on the Cache Tuning form. You can access this form by choosing Caching | Tune Cache from the Server Manager.

CAUTION The update after percent setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Sync Dump Ticks

You should not change this value under any circumstances.

Byte-Ranges

The *byte-ranges* value determines whether or not the proxy is allowed to generate byte-range responses from the cache. By default, this feature is disabled. This version of the proxy supports single byte-ranges only. If Adobe Acrobat plugin is not in use, enabling this feature will allow faster truncated document/image retrievals by Navigator clients.

Single Accept

You should not change this value under any circumstances.

Tuning the Garbage Collector

Garbage Collection is a resource-intensive process. Therefore, you may need to tune some garbage collection settings to improve its performance. You can view and modify all of the following garbage collection settings on the GC Tuning form.

Gc URL DB Interval

The *gc URL db interval* controls how often the URL database is cleaned of old URLs and how often it is checked for consistency.

The valid range for the gc URL db interval is 1800 sec (30 minutes) to 604800 (1 week), and the default value is 79200 seconds (22 hours).

You can view or modify the gc URL db interval on the GC Tuning form. You can access this form by choosing Caching|Tune GC from the Server Manager.

CAUTION The gc URL db interval should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Gc Nap Length

The garbage collector takes “naps” every so often so that it does not use all the CPU from other processes. The *gc nap length* is the amount of time that the garbage collector sleeps during one nap.

The frequency of these naps is controlled by two variables: *hard gc nap count* and *soft gc nap count*.

The valid range for gc nap length is 0 (no naps) to 120 (2 minutes), and the default is 1 second.

You can view or modify the gc nap length on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The gc nap length should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Hard Gc Nap Count

There are two types of garbage collection: hard and soft. The hard garbage collector traverses the entire cache section and finds all the files individually, picking the ones to delete, and then generates lists of files in the cache ordered by their relative value. These lists are then used by the soft garbage collector to delete files without traversing the entire cache structure.

Hard garbage collection is more resource intensive than soft garbage collection, but it is required every time soft garbage collection runs out of the lists generated by hard garbage collection.

The *hard gc nap count* value specifies how many naps hard garbage collector takes during the garbage collection cycle of a single cache section. There are 64 subdirectories on each cache section, and naps can be taken between directories only. This means that there can be at most 64 naps. The number of naps should be a power of two.

The length of each nap is controlled by the variable *gc nap length*.

The valid values for hard gc nap count are: 0 (no sleeps), 1, 2, 4, 8, 16, 32, and 64 (max). The default value is 16 (sleep after every 4 directories).

You can view or modify the hard gc nap count on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The hard gc nap count should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Soft Gc Nap Count

The soft garbage collector simply reads a list of cache files from a set of files produced by the hard garbage collector. It still looks up the file using the stat() system call to find out if there have been subsequent accesses to the cache file during the last garbage collection, which suggests that the cache file is being (or may be) actively used, and should not be removed. The *soft gc nap count* value specifies how many naps the soft garbage collector takes during the garbage collection cycle.

The length of each nap is controlled by the variable *gc nap length*.

The valid range for soft gc nap count is 0 (no naps) to 1000 naps, and the default value is 20 naps.

You can view or modify the soft gc nap count on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The soft gc nap count should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Hard Gc Max Entries

Hard gc max entries identifies the number of cache entry slots allocated during one pass for garbage collection. One “pass” in garbage collection is defined by the variable *gc dir chunk* which is the number of subdirectories within a cache section that will be processed in one pass.

This number should be of the same order as the number of files in a typical chunk of directories. As an example, a typical subdirectory in a cache section has 100-200 files. If the gc dir chunk variable is set to 8, then the “hard gc max entries” value should be set to around 800-1600.

A single cache entry is about 32 bytes, so every 1000 entries in this variable means 32KB pool allocation.

A valid range for hard gc max entries is 100(3K) to 5000 (160K), and the default value is 500 (16K).

You can view or modify the hard gc max entries setting on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The hard gc max entries setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Gc Dir Chunk

The *gc dir chunk* variable controls the sample size for the LRU algorithm. Basically, this means it controls how many subdirectories under each cache section are processed by the garbage collector in one pass. Because a larger gc dir chunk value causes the proxy to process more files simultaneously, more memory is required. Larger gc dir chunk values also cause the garbage collector to be slower and more CPU intensive. The smaller the gc dir chunk value is, the lighter-weight garbage collection becomes. However, the sample size for the LRU algorithm becomes smaller.

The gc dir chunk value must be a power of two.

Valid values for gc dir chunk are: 1 (single directory), 2, 4, 8, 16, 32, and 64 (all directories at once). The default value is 8 directories.

You can view or modify the gc dir chunk setting on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The gc dir chunk setting should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Gc Hi Margin Percent

The *gc hi margin percent* variable controls the percentage of the maximum cache size that, when reached, triggers garbage collection.

This value must be higher than the value for gc lo margin percent.

The valid range for gc hi margin percent is 10 to 100 percent (trigger garbage collection when full). The default value is 80 percent (trigger gc when 80% full).

You can view or modify the gc hi margin percent on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The gc hi margin percent should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Gc Lo Margin Percent

The *gc lo margin percent* variable controls the percentage of the maximum cache size that the garbage collector targets.

This value must be lower than the value for gc-hi-margin-percent.

The valid range for gc lo margin percent is 5 to 100 percent. The default value is 70 percent (target at 70% full cache after gc).

You can view or modify the gc lo margin percent on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The gc lo margin percent should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Gc Extra Margin Percent

If the garbage collection is triggered by a reason other than the partition’s size getting close to the maximum allowed size (see gc-hi-margin-percent), the garbage collector will use the percentage set by the *gc extra margin percent* variable to determine the fraction of the cache to remove.

The valid range for gc extra margin percent is 0 to 100 percent, and the default value is 30 percent (remove 30% of existing cache files).

You can view or modify the gc extra margin percent on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The gc extra margin percent should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Gc Leave Fs Full Percent

The *gc leave fs full percent* value determines the percentage of the cache partition size below which garbage collection will not go. This value prevents the garbage collector from removing all files from the cache if some other application is hogging the disk space.

The valid range for gc leave fs full percent is 0 (allow total removal) to 100 percent (remove nothing). The default value is 60 percent (allow the cache size to shrink to 60% of current).

You can view or modify the gc leave fs full percent on the GC Tuning form. You can access this form by choosing Caching | Tune GC from the Server Manager.

CAUTION The gc leave fs full percent should only be changed if you are instructed to do so by iPlanet Technical Support. The default values should not be changed unless there is a problem with the default behavior.

Chapter 16

Configuring the Proxy Manually

This chapter describes the configuration files that iPlanet Web Proxy Server uses. These are the files that you're changing when you use iPlanet Web Proxy Server Manager online forms. You can also configure iPlanet Web Proxy Server manually by editing the files directly.

You might need to configure iPlanet Web Proxy Server manually for various reasons. If you accidentally lock your hosts out of the administrative forms or forget your administrative password, you'll have to change information manually in the proxy's configuration files. Perhaps more importantly, you will probably need to develop an understanding of what your configuration files do for you, so that you can write scripts to automate configuration functions that you might want in addition to those available in the online forms. This is especially useful if, for example, you are using many proxy servers or your URL lists require frequent or high-volume updates.

Before you can edit any of the configuration files, you must have permission to read and write to the files. If you are running a UNIX proxy server, you might need to log in as root or use the proxy's user account.

The files that you use to configure the proxy are in the *server root*/`proxy-`*id*/`config` directory. Here's a brief description of each file:

- `magnus.conf` is the server's main technical configuration file. It controls aspects of the server operation not related to specific resources or documents, such as host name and port.
- `obj.conf` is the server's object configuration file. It controls access to the proxy server and determines how documents are proxied and cached.
- `mime.types` is the file the server uses to convert file name extensions such as `.GIF` into a MIME type like `image/gif`.

- admpw is the administrative password file. Its format is user:password. The password is DES-encrypted, just like /etc/passwd. This file, as opposed to the other configuration files, is located in the *server root*/admin-serv directory.
- socks5.conf is a file that contains the SOCKS server configuration. The SOCKS daemon is a generic firewall daemon that controls point-to-point access through the firewall. If you use SOCKS, the SOCKS server configuration file has to be stored in /etc/socks5.conf. This file is described on page 263.
- bu.conf is an optional file that contains batch update directives. You can use these to update many documents at once. You can time batch updates; for example, you can have them occur during off-peak hours to minimize the effect on the efficiency of the server.
- icp.conf is the Internet Cache Protocol (ICP) configuration file. It identifies the information about the parent and sibling servers in a proxy array that uses ICP.
- parray.pat is the Proxy Array Table file. The PAT file is an ASCII file used in proxy to proxy routing. It contains information about a proxy array; including the members' machine names, IP addresses, ports, load factors, cache sizes, etc. For more information on the syntax of the parray.pat file, see "The parray.pat File" on page 266.
- parent.pat is the Proxy Array Table file that contains information about an upstream proxy array. For more information on the syntax of the parent.pat file, see "The parent.pat File" on page 267.

The magnus.conf File

The technical configuration file, magnus.conf, controls all global server operations. All of the items in the magnus.conf file apply to the entire proxy server, as opposed to affecting only one URL or set of URLs. The obj.conf file handles URLs (also called resources).

Every command line in the file has this format:

```
Directive Value
```

Directive identifies an aspect of server operation. This string is not case-sensitive.

Value is a number or label you give the directive. Its format depends on the directive. Unlike the Directive string, this string is usually case-sensitive.

Directive lines should not contain white spaces at the beginning of the line or more than one space between the directive and value. Comment lines begin with a # character with no leading white space. If you operate on the configuration files with the Server Manager, when it writes the files out again it does not write comment lines.

The directives in `magnus.conf` are explained in detail in Appendix C, "Proxy Configuration Files."

The obj.conf File

The iPlanet Web Proxy Server object configuration file, `obj.conf`, uses objects to control how the server performs access control, routes URLs, and initializes server subsystems.

Configuration objects (also called resources) are settings that tell the proxy how to treat URLs. URLs matching a specified wildcard pattern belong to the same configuration object (or resource). This object grouping can then be used to control, in fine detail, the behavior of the proxy server.

Using this object-grouping scheme, you can specify single resources with their complete URL, whole "directories" with the path followed by /.*, and various other groups such as .*\.html. You can then configure the settings you want to use for that object (for example, caching or denying access based on the server's host name or a string in a URL).

The Structure of obj.conf

The `obj.conf` file must have specific objects in it (the objects are described on page 259). You can add other objects to this file. To specify an object, use this format:

```
<Object ppath=reg-exp>

Directives

...

  <Client dns=shell-exp>

     Directives

     ...

  </Client>

</Object>
```

Although <Client> lines are not required, you can have as many as needed.

If you want to control access at the URL level, use regular expression patterns to control which URLs are grouped in the object. You can then specify one or more directives to control what the proxy server does when it encounters any URL matching the regular expression pattern specified with ppath.

You can also set options for specific client hosts. This is a powerful feature. Unlike other proxy servers that simply control whether a host can or cannot access a URL, you can make the proxy act differently depending on which user or host is requesting the URL.

Directive Syntax

Each directive line (regardless of where it appears in `obj.conf`) has this format:

```
Directive fn=function [parameter1=value1]...[parameterN=valueN]
```

Directive identifies an aspect of server operation. This string is not case-sensitive and must appear at the beginning of a line.

Function is a function and parameters given to the directive. Its format depends on the directive.

Directive lines cannot contain spaces at the beginning of the line or extra spaces between the directive and value. You shouldn't use trailing spaces after the value because they might confuse the server. Long lines can be continued by starting the next line with white space. *White space* is any keystroke that leaves space on the screen, such as space bar, tab, carriage return, line feed, or vertical tab. Comment lines begin with a # character with no leading white space. If you operate on the configuration files with the Server Manager, when it writes the files out again it does not write comment lines.

CAUTION If you are using the Administration forms, you shouldn't use continuation lines in the `obj.conf` file. Instead, put each directive entirely on a single line. If you are absolutely sure you will never use the Administration forms to edit the `obj.conf` file, you can use the \ character.

A Sample Object

The following sample object applies to all HTTP URLs (the pattern is http://.*). When the proxy receives a request for an HTTP document, it scans the URL for the string play (as specified in PathCheck); if it finds that string in the URL, it doesn't retrieve the document from the remote server, and it denies service to the client.

```
<Object ppath="http://.*">
      PathCheck fn=deny-service
             path=".*play.*"
      ObjectType fn=cache-setting
             max-uncheck=14400
             lm-factor=0.1
      Service  fn=proxy-retrieve
</Object>
```

This object also caches all HTTP documents and refreshes the documents if they are older than four hours or if they need refreshing as determined by the date they were last modified. The Service directive tells the proxy to retrieve the HTTP documents by default. The following code is an example of an `obj.conf` file on a UNIX system:

```
# Netscape Communications Corporation - obj.conf
# You can edit this file, but comments and formatting changes
# might be lost when the admin server makes changes.
Init fn="flex-init" format.access="%Ses->client.ip% - %Req->vars.auth-user%
[%SYSDATE%]
\"%Req->reqpb.proxy-request%\" %Req->srvhdrs.status% %Req->vars.p2c-cl%"
access="/usr/ns-proxy/proxy-TEST/logs/access"
Init fn="load-types" mime-types="mime.types"
Init fn="init-proxy" timeout2="15" timeout="1200"
Init fn="init-cache" dir="/usr/ns-proxy/cache" status="on" ndirs="1"
Init fn="init-partition" min-avail="5" max-size="10" dir="/usr/ns-proxy/cache"
status="on" name="NoName"
```

```
Init fn="init-urldb" dir="/usr/ns-proxy/cache/urldb" status="on"
Init conf-file="bu.conf" fn="init-batch-update" dir="/tmp" status="off"
<Object name="default">
NameTrans from="file:" fn="map" to="ftp:"
NameTrans from="/ns-icons" fn="pfx2dir" dir="/usr/ns-proxy/ns-icons" name="file"
PathCheck fn="url-check"
PathCheck fn="check-acl" acl="proxy-TEST_formgen-READ-ACL_allow-1970"
PathCheck fn="check-acl" acl="proxy-TEST_formgen-WRITE-ACL_deny-1970"
Service fn="deny-service"
AddLog fn="flex-log" name="access"
AddLog fn="urldb-record"
</Object>
<Object name="file">
PathCheck fn="UNIX-uri-clean"
PathCheck index-names="index.html" fn="find-index"
ObjectType fn="type-by-extension"
ObjectType fn="force-type" type="text/plain"
Service fn="send-file"
</Object>
<Object ppath="ftp://.*">
ObjectType fn="cache-enable"
ObjectType fn="cache-setting" max-uncheck="21600"
Service fn="proxy-retrieve"
</Object>
<Object ppath="http://.*">
ObjectType fn="cache-enable"
ObjectType lm-factor="0.100" fn="cache-setting" max-uncheck="3600"
Service fn="proxy-retrieve"
</Object>
<Object ppath="https://.*">
Service fn="proxy-retrieve"
</Object>
<Object ppath="gopher://.*">
ObjectType fn="cache-enable"
ObjectType fn="cache-setting" max-uncheck="14400"
Service fn="proxy-retrieve"
</Object>
<Object ppath="connect://.*:443">
Service fn="connect" method="CONNECT"
</Object>
<Object ppath="connect://.*:563">
Service fn="connect" method="CONNECT"
</Object>
```

Required Objects for obj.conf

Certain objects must be in the `obj.conf` file to make the Administration forms work for your proxy. If you are familiar with Netscape server software (a regular HTTP server), you might notice that these functions control local file access and CGI execution.

On a proxy server the local access interface is a simplified version, and it cannot be used for any purpose other than the online forms. Special care is taken inside the proxy software to guarantee that it cannot be used, accidentally or otherwise, as a normal HTTP server. If you don't use the online forms, these objects don't necessarily have to be in `obj.conf`.

The Default Object

The default object contains the required directives. *Named objects* are objects identified by <Object name=...> in the object configuration file. To control the behavior of the entire server, you would modify the setting for the default object. This object must contain all of the name-translation directives for the server, and it should contain any global configuration changes. Here is an example of a default object for a proxy server running on Windows NT:

```
<Object name=default>
NameTrans fn=map from=file: to=ftp:
NameTrans fn=pfx2dir from=ns-icons dir="" name=file
Service fn=deny-service
AddLog fn=-log
</Object>
```

- The first **NameTrans** directive takes care of URLs that use “file:” by changing them to “ftp:” URLs. If you have any mappings to mirror sites, put them after this mapping.
- The next **NameTrans** directive maps the ns-icons URL into its directory. These are the only legal uses for the **pfx2dir** function, which doesn't belong to the actual proxy configuration (you'll get errors if you try to use it anywhere else).
- The **deny-service** function ensures that by default access isn't granted. (Access isn't granted by default even if you forget this function, but the error message is less descriptive and it is classified as a misconfiguration.)
- The **proxy-log** function takes care of proxy access logging, whether it is in flexible, common, extended, or extended-2 log format. This function can have the additional iponly=1 parameter, which inhibits reverse DNS lookups and logs only the IP address of the requesting client.

- The **urldb-record** function logs URLs and has to be called for each object for which you want URL recording.

How the Proxy Server Handles Objects

Netscape servers (the HTTP server and the proxy) respond to an information request by following certain steps. Each step in the process is done once for all objects, then another step is done for all objects, and so on. The process steps that the server performs are:

1. *Authorization translation*. Translate any authorization information given by the client into a user and group. If necessary, decode the message to get the actual request. Also, proxy authorization is available.
2. *Name translation*. Before anything else is done, a URL can be translated into a file-system-dependent name (an administration URL), a redirection URL, or a mirror site URL, or it might be kept intact and retrieved as is (the normal case for proxy).
3. *Path checks*. Perform various tests on the resulting path, largely used to make sure that it's safe for the given client to retrieve the document (only for local access).
4. *Object type determination*. Determine the MIME type information for the given document. MIME types can be registered document types such as `text/html` and `image/gif`, or they can be internal document identification types. Internal types always begin with `magnus-internal/` and are used to select a server function to use to decode the document (only used for local access; the proxy system calls these routines automatically when necessary).
5. *Service selection*. Select the internal server function that should be used to send the result back to the client. This function can be the normal proxy service routine, or local file blast.
6. *Logging selection*. Select a function to record information about transactions as they finish.

These steps map directly to several configuration directives allowed for each object. Another configuration directive, **send-error**, controls how the server responds to the client when it encounters an error.

The directives in `obj.conf` are explained in detail in Appendix C, “Proxy Configuration Files.”

The mime.types File

The `mime.types` file tells the server how to convert files with certain extensions (such as `.gif`) into a MIME type (such as `image/gif`). MIME files are compact files and transfer quickly. Also, MIME is needed by browsers (like Netscape Navigator); without MIME they can’t tell the difference between an HTML page and a graphics file.

The `mime.types` file contains the global file extensions for all proxy servers. The first line in the file identifies the file format and must read:

```
#--Netscape Communications Corporation MIME Information
```

The following code is a sample `mime.types` file:

```
#--Netscape Communications Corporation MIME Information
# Don’t delete the above line. It identifies this file’s type.
#
# This is a simple MIME types file for Netscape Proxy Server. Most
# of the MIME types are already compiled in the proxy. Types that
# are part of the Administration forms (HTML and GIF) must appear
# here, or they won’t be known to the part of the server that
# manages the Administration interface calls.
#
# Icons (internal-gopher-…) are references to Netscape's
# internal icons. If a client doesn't support these icons, the
# proxy will provide them.
type=application/oda                 exts=oda
type=application/pdf                 exts=pdf
type=application/x-mif               exts=mif
type=application/x-dvi               exts=dvi
type=application/x-hdf               exts=hdf
type=application/x-netcdf            exts=nc,cdf
type=application/x-texinfo           exts=texinfo,texiicon=internal-gopher-text
type=application/zip                 exts=zip
type=application/x-tar               exts=tar
type=application/x-macbinary         exts=bin
type=application/x-stuffit           exts=sit
type=image/gif                       exts=gif     icon=internal-gopher-image
type=image/jpeg                      exts=jpeg,jpg,jpeicon=internal-gopher-image
type=image/x-xwindowdump             exts=xwd     icon=internal-gopher-image
type=text/html
exts=htm,html,shtml icon=internal-gopher-text
type=text/plain                      exts=txt     icon=internal-gopher-text
type=text/richtext                   exts=rtx     icon=internal-gopher-text
type=text/tab-separated-values       exts=tsv     icon=internal-gopher-text
```

```
type=text/x-setext                   exts=etx     icon=internal-gopher-text
type=application/x-tar enc=x-gzip    exts=tgz
enc=x-gzip                           exts=gz
enc=x-compress                       exts=z
```

Parameters

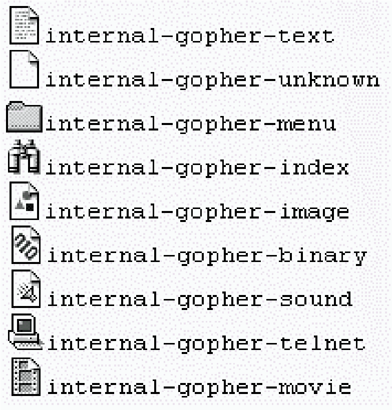
Other non comment lines have this format:

`type=type/subtype exts=`*file extensions* `icon=icon`

where each parameter is as follows.

- **type/subtype** is the MIME type and subtype.
- **exts** are the file extensions associated with this type. When the proxy transfers a file with one of these extensions, it uses the MIME type you specify in type.
- **icon** is the name of the icon the browser displays; the icons are shown in Figure 16-1. Netscape Navigator keeps these images internally. If you use a browser that doesn't have these icons, Netscape Proxy Server delivers them.

Figure 16-1 Internal icons for MIME types



CAUTION If you set the `.pac` MIME type to anything other than `application/x-ns-proxy-autoconfig`, the proxy autoconguration feature will not work.

The admpw File

The admpw file contains the administration password. If you forget your password, there is no way to find out what it was. You must encrypt a new one and replace the old version with it. The file has the format user:password.

If you forget your administration password, you can edit the admpw file and delete the password section (everything after the semicolon). When you go to the administration server, you don't need to enter a new password, but you should immediately go to Access Control in the iPlanet Web Proxy Server Manager and set a new one.

CAUTION Because you can replace the Administration password, it is very important to keep secure the proxy's account and to ensure that only that proxy account and, for UNIX proxy servers, the root account, have full (read / write) access to the server root directory. This way, only someone running as root or with the proxy's user account can enter the *server root*/proxy-*id*/config directory and edit the file.

The socks5.conf File

The SOCKS daemon is a generic firewall daemon that controls point-to-point access through the firewall. By default, the SOCKS daemon features are disabled. The iPlanet Web Proxy Server supports SOCKS versions 4 and 5.

The proxy uses the file socks5.conf to control access to the SOCKS proxy server and its services. Each line defines what the proxy does when it gets a request that matches the line.

When the SOCKS daemon receives a request, it checks the request against the lines in the socks5.conf file. When it finds a line that matches the request, the request is permitted or denied based on the first word in the line (permit or deny). Once it finds a matching line, the daemon ignores the remaining lines in the file. If there are no matching lines, the request is denied. You can also specify actions to take if the client's identd or user ID is incorrect by using #NO_IDENTD: or #BAD_ID as the first word of the line. Each line can be up to 1023 characters long.

Although the SOCKS daemon doesn't know if a host is internal to its network, it does know which host is the requestor and which is the destination (it uses this for access control). This means SOCKS daemon provides access from external hosts into your internal networks in addition to the normal internal-to-external proxy functionality.

NOTE Use caution with the external-to-internal functionality. If you don't need external to internal access, you should specifically deny such connections. For example, if 198.95 is your internal network, use the following as the first lines in `socks5.conf` to protect your internal hosts from external access attempts:

```
auth 198.95. - -
```

```
ban - -
```

These lines will allow anyone on the 198.95 intranet to authenticate using any type of authentication, and will ban all other hosts from the server.

For information on the syntax of socks5.conf, see "The socks5.conf File," on page 468.

The bu.conf File

The optional `bu.conf` file contains batch update directives. You can use these to update many documents at once. You can time these updates to occur during off-peak hours to minimize the effect on the efficiency of the server. The format of this file is described in this section. For more information on batch updating and starting the batch update function, see "init-batch-update (starting batch updates)," on page 429.

Object Boundaries

All of the batch update directives must be in Object boundaries.

The pairs of Object boundaries indicate the individual configurations in the bu.conf file. If you give a unique name to each occurrence, you can specify these boundaries any number of times.

Where you see italicized text in the directive syntax examples, substitute your own information in place of the italicized text.

Syntax

```
<Object name=object_name>

...
```

```
</Object>
```

The directives in bu.conf are explained in detail in Appendix C, “Proxy Configuration Files.”

Examples of bu.conf

Here are some examples of code in a `bu.conf` file for a proxy server running on UNIX:

This example code updates the entire cache every evening:

```
<Object name=cache update>
Source internal
Accept .*
Type ignore
Connections 4
Depth 1
Count 300
Days Sun Mon Tue Wed Thu Fri Sat
Time 20:00 - 3:00
</Object>
```

This example code tells the proxy to mirror a company internal site to a local proxy:

```
<Object name= HumanResourcesMirror>
Source http://hr.mycompany.com
Accept .*
Type inline
Connections 6
Depth 9
Count 1000
Days Mon Tue Wed Thu Fri
Time 4:00 - 6:00
</Object>
```

The icp.conf File

This file is used to configure the Internet Cache Protocol (ICP) feature of your server. There are three functions in the `icp.conf` file, and each can be called as many times as necessary. Each function should be on a separate line. The three functions are **add_parent**, **add_sibling**, and **server**.

For more information on this file and the functions within it, see "The icp.conf File," on page 483.

The parray.pat File

The `parray.pat` (PAT) file describes each member in the proxy array of which the proxy you are administering is a member. The PAT file is an ASCII file used in proxy to proxy routing. It contains proxy array members' machine names, IP addresses, ports, load factors, cache sizes, etc.

Syntax

```
Proxy Array Information/1.0
ArrayEnabled: number
ConfigID: ID number
ArrayName: name
ListTTL: minutes
```

name IPaddress proxyport URLforPAT infostring state time status loadfactor cachesize

Parameters

Proxy Array Information is version information.

ArrayEnabled specifies whether the proxy array is enabled or disabled. Possible values are:

- **0** means the array is disabled.
- **1** means the array is enabled.

ConfigID is the identification number for the current version of the PAT file. The proxy server uses this number to determine whether the PAT file has changed.

ArrayName is the name of the proxy array.

ListTTL specifies how often the proxy should check the PAT file to see if it has changed. This value is specified in minutes.

name is the name of a specific member of the proxy array.

IPaddress is the IP address of the member.

proxyport is the port at which the master proxy accepts HTTP requests.

URLforPAT is the URL of the PAT file that the member will poll the master proxy for.

infostring is version information.

statetime is the amount of time the member has been in its current state.

status specifies whether the member is enabled or disabled.

- **on** means that the member is on.
- **off** means that the member is off. If the member is off, its requests will be routed through another member of the array.

loadfactor is an integer that reflects the number of requests that should be routed through the member.

cachesize is the size of the member’s cache.

Example

```
Proxy Array Information/1.0
ArrayEnabled: 1
ConfigID: 1
ArrayName: parray
ListTTL: 10

proxy1 200.29.186.77 8080 http://pat iPlanetWebProxy/3.6 0 on 100
512
proxy2 187.21.165.22 8080 http://pat iPlanetWebProxy/3.6 0 on 100
512
```

The parent.pat File

The `parent.pat` file is the Proxy Array Table file that contains information about an upstream proxy array. This file has the same syntax as the `parray.pat` file.

Part 2

Programming the Proxy Server

Chapter 17

Creating Server Plug-in Functions

This chapter describes how to create and compile your plug-in functions using the iPlanet Web Proxy Server plug-in application programming interface (API) and how to use the functions you create.

Before creating plug-in functions, you should be familiar with the server configuration files and the built-in functions.

Of the systems the iPlanet server supports, the following systems can load functions into the server at run time and can therefore use plug-in functions:

- Solaris
- HP/UX
- AIX

What Is the Server Plug-in API?

The server plug-in API is a set of functions and header files that help you create functions to use with the directives in server configuration files. The iPlanet Web Proxy Server uses this API to create the functions for the directives used in both `magnus.conf` (the server configuration file) and `obj.conf` (the object configuration file).

The server uses this API, so by becoming familiar with the API, you can learn how the server works. This means you can override the server functionality, add to it, or customize your own functions. For example, you can create functions that use a custom database for access control or functions that create custom log files with special entries.

These steps are a brief overview of the process for creating your own plugin functions:

1. You write code for your functions. Each function you create is written specifically for the directive with which it will be used in the configuration files.
2. For UNIX, you compile your code to create a shared object file (`.so` file).
3. For a UNIX proxy server, tell the server to load your shared object file in the **Init** directives of `obj.conf`.
4. You use your functions in your server configuration file (`obj.conf`).

Before you write your functions, you should understand how the server handles requests.

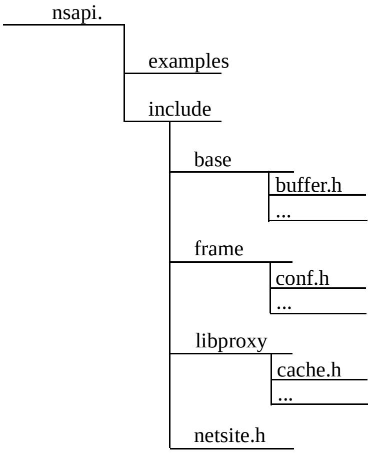
Writing Plug-in Functions

This section describes how to begin writing your plug-in functions. It also describes the header files you need to include in your code. See "Compiling and Linking Your Code" on page 280 for additional information.

The server root directory has a subdirectory called `/nsapi` that contains sample code, the header files, and a makefile. You should familiarize yourself with the code and samples. This documentation is written as a starting point for exploring that code. Figure 17-1 shows the hierarchy of the server plug-in API header files.

- The `nsapi/examples/` directory contains C files with examples for each class of function you can create.
- The `nsapi/include/` directory contains all the header files you need to include when writing your plug-in functions.

Figure 17-1 The hierarchy of server plug-in API header files



The server and its header files are written in ANSI C. On some systems you must have an import list that specifies all global variables and functions you need to access from the server binary.

The Server Plug-in API Header Files

This section describes the header files you can include when writing your plug-in functions. This section is intended as a starting point for learning the functions included in the header files.

Most of the header files are stored in two directories:

- `nsapi/include/base` contains header files that deal with low-level, platform-independent functions such as memory, file, and network access.
- `nsapi/include/frame` contains header files of functions that deal with server- and HTTP-specific functions such as handling access to configuration files and dealing with HTTP.

One header file, `netsite.h`, is stored in the `nsapi/include` directory.

Table 17-1 Header files in the `base` directory

Header File	Description
`buffer.h`	Contains functions that buffer I/O (input/output) for a file or a socket descriptor.
`cinfo.h`	Contains functions for object typing, specifically mapping files to MIME types.
`crit.h`	Contains functions for managing critical sections, an abstraction that facilitates the management of threaded servers.
`daemon.h`	Contains functions called from other header files. It also contains functions that manage group processes that run the server.
`ereport.h`	Contains functions that handle low-level errors.
`file.h`	Contains functions to handle file I/O.
`net.h`	Contains functions for I/O with the client software over the network.
`pblock.h`	Contains functions that manage parameter passing and server internal variables. It also contains functions to get values from a user via the server.
`pool.h`	Contains routines that manage memory pools.
`regexp.h`	Contains functions that support regular expressions.
`sem.h`	Contains semaphores in platform-independent ways (they prevent two processes from doing the same thing).
`session.h`	Contains session data structures for IP addresses, security, and so on.
`shexp.h`	Contains functions to customize wildcard patterns through parsed data.
`shmem.h`	Contains functions that support shared memory.
`systems.h`	Contains functions that handle systems information.
`systhr.h`	Contains functions that support the abstract threading mechanism.
`util.h`	Contains utility functions.

Header File	Description
`conf.h`	Contains functions to access `magnus.conf` (for example, to get port numbers or internal global variables).
`func.h`	Contains data structures. This file is rarely used.
`http.h`	Contains functions for the HTTP protocol. Most of these functions are called from functions in `protocol.h`.
`log.h`	Contains functions for logging errors.
`object.h`	Contains functions for reading `obj.conf`. You'll rarely use these functions.
`objset.h`	Contains functions for reading `obj.conf`. You'll rarely use these functions.
`protocol.h`	Contains functions that perform protocol-specific actions.
`req.h`	Contains request data structures.

Header File	Description
`cache.h` (UNIX Only)	Contains functions that manage proxy caching.
`cif.h` (UNIX Only)	Contains functions for cache information file management.
`cutil.h` (UNIX Only)	Contains cache utility functions.
`fs.h` (UNIX Only)	Contains functions the proxy uses to access the file system.

Header File	Description
`netsite.h`	Contains miscellaneous functions and some vital definitions. Be sure to include this in all your `.c` files, to make sure that the necessary definitions (#defines) are established.

Getting Data From the Server: The Parameter Block

The server stores variables in name-value pairs. The parameter block, or pblock, is a hash table keyed on the name string. The pblock maps these name strings onto their value character strings.

Basically, your plug-in functions use parameter blocks to get, change, add, and remove name-value pairs of data. In order to use the functions to do these actions, you need to know a bit about how the hash table is formed and how the data structures are managed.

The pb_param structure is used to manage the name-value pairs for each client request. The pb_entry structure creates linked lists of pb_param structures. See “The Session Data Structure” on page 395 for more information.

Passing Parameters to Server Application Functions

All server application functions (regardless of class) are described by this prototype:

```
int function(pblock *pb, Session *sn, Request *rq);
```

pb is the parameter block containing the parameters given by the site administrator for this function invocation.

CAUTION The *pb* parameter should be considered read-only, and any data modification should be performed on copies of the data. Doing otherwise is unsafe in threaded server architectures and will yield unpredictable results in multiprocess server architectures.

Parameter-manipulating Functions

When adding, removing, editing, and creating name-value pairs, you use the following functions. This list might seem overwhelming, but you’ll use only a handful of these functions in your plug-in functions.

The `param_create` function creates a parameter with the given name and value. If the name and value aren’t null, they are copied and placed in the new `pb_param` structure.

The `param_free` function frees a given parameter if it’s non-NULL. It is also useful for error checking before using the `pblock_remove` function.

The `pblock_create` function creates a new parameter block with a hash table of a chosen size.

The `pblock_free` function frees a given parameter block and any entries inside it.

The `pblock_find` function finds the name-value entry with the given name in a given parameter block.

The `pblock_findval` function finds the value portion of a name-value entry with a given name in a given parameter block and returns its value.

The `pblock_remove` function behaves like the `pblock_find` function, but when it finds the given parameter block, it removes it.

The `pblock_nninsert` and `pblock_nvinsert` functions both create a new parameter with a given name and value and insert it in a given parameter block. The `pblock_nninsert` function requires that the value be an integer, but the `pblock_nvinsert` function accepts a string.

The `pblock_pinsert` function inserts a parameter in a parameter block.

The `pblock_str2pblock` function scans the given string for parameter pairs in the format name=value or name="value".

The `pblock_pblock2str` function places all of the parameters in the given parameter block in the given string. Each parameter is of the form name="value" and is separated by a space from any adjacent parameter.

Data Structures and Data Access Functions

The data structures are *Session* (see "The Session Data Structure" on page 395) and *Request* (see "The Request Data Structure" on page 397). The data access function is `request_header`.

The Request->vars parameter block contains the server's working variables. The set of active variables is different depending on which step of the request the server is processing.

The Request->reqpb parameter block contains the request parameters that are sent by the client:

- method is the HTTP method used to access the object. Valid HTTP methods are currently GET, HEAD, and POST.
- uri is the URI for which the client asks. The uri is the part of the URL following the host:port combination. This uri is unescaped by the server using URL translations.
- protocol identifies the protocol the client is using.
- clf-request is the full text of the first line of the client's request. This is used for logging purposes.

The Request->headers parameter block contains the client's HTTP headers. HTTP sends any number of headers in this form (RFC 822):

```
Name: value
```

If more than one header has the same name, then they are concatenated with commas:

```
Name: value1, value2
```

The parameter block is keyed on the fully lowercase version of the name string without the colon.

The Request_header Function

The `request_header` function finds the parameter block that contains the client's HTTP headers.

```
#include "frame/req.h"
```

```
int request_header(char *name, char **value, Session *sn, Request *rq);
```

The *name* parameter should be the lowercase header name string for which to look, and *value* is a pointer to your char * that should contain the header. If no header with the given name is sent, *value* is set to NULL.

The Request->srvhdrs parameter block is the set of HTTP headers for the server to send back. This parameter block can be modified by any function.

The last three entries in the Request structure should be considered transparent to application code because they are used by the server's base code.

After the server has a path for the file it intends to return, application functions should use the `request_stat_path` function to obtain stat information about the file. This avoids multiple, unnecessary calls to the `stat` function.

Application Function Status Codes

When your plug-in function is done working with the name-value pairs, it must return a code that tells the server how to proceed with the request.

Reporting Errors to the Server

When problems occur, server application functions should set an HTTP response status code to give the client an idea of what went wrong. The function should also log an error in the error log file.

There are two ways of reporting errors: setting a response status code and reporting an error.

Setting an HTTP Response Status Code

The `protocol_status` function sets the status to the code and reason string. If the reason is NULL, the server attempts to match a string with the given status code (see Table 17-2). If the server can't find a string, it uses "Unknown error."

```
#include "frame/protocol.h"
void protocol_status(Session *sn, Request *rq, int n, char *r);
```

Generally, `protocol_status` will be called with a NULL reason string, and one of the following status codes defined in the `protocol.h` file. If no status is set or the code is set as NULL, the default is PROTOCOL_SERVER_ERROR.)

Table 17-2 Status codes used with protocol_status

Status code	Definition
PROTOCOL_BAD_REQUEST	The request was unintelligible. Used primarily in the framework library.
PROTOCOL_FORBIDDEN	The client is explicitly forbidden to access the object and should be informed of this fact.
PROTOCOL_NOT_FOUND	The server was unable to locate the item requested.
PROTOCOL_NOT_IMPLEMENTED	The client has asked the server to perform an action that it knows it cannot do. Generally, you would use this to indicate your refusal to implement an HTTP feature.
PROTOCOL_NOT_MODIFIED	If the client gave a conditional request, such as an HTTP request with the if-modified-since header, this indicates that the client should use its local copy of the data.
PROTOCOL_OK	Normal status; the request will be fulfilled normally. This should be set only by **Service**-class functions.
PROTOCOL_REDIRECT	The client should be directed to a new URL, which your function should insert into the rq->vars parameter block as url.

Table 17-2 Status codes used with protocol_status

Status code	Definition
PROTOCOL_SERVER_ERROR	Some sort of server-side error has occurred. Possible causes include misconfiguration, resource unavailability, and so on. Any error unrelated to the client generally falls under this rather broad category.
PROTOCOL_UNAUTHORIZED	The client did not give sufficient authorization for the action it was trying to perform. A WWW-authenticate header should be present in the **rq->srvhdrs** parameter block that indicates to the client the level of authorization it needs to perform its action.

Error Reporting

When errors occur, it's customary to report them in the server's error log file. To do this, your plug-in functions should call **log_error**. This logs an error and then returns to tell you if the log records successfully (a return value of 0 means success; -1 means failure).

```
#include "frame/log.h"

int log_error(int degree, char *func, Session *sn, Request *rq,

char *fmt, ...);
```

You can give `log_error` any printf( ) style string to describe the error. If an error occurs after a system call, use the following function to translate an error number to an error string:

```
#include "base/file.h"

char *system_errmsg(SYS_FILE fd );
```

NOTE The *fd* parameter is vestigial and might need to be changed for operating systems other than UNIX and Windows NT. Therefore, it is best to set *fd* to zero.

Compiling and Linking Your Code

You can compile your code with any ANSI C compiler. See the makefile in the `/nsapi/include` directory. The make file assumes the use of gmake.

This section lists the linking options you need to use in order to create a UNIX shared object. The server can be instructed to load by commands in the `magnus.conf` configuration file.

Table 17-3 describes the commands used to link object files into a shared object under the various UNIX platforms. In these examples, the compiled object files `t.o` and `u.o` are linked to form a shared object called `test.so`.

Table 17-3 Options for linking

System	Compile options
Solaris	`ld -G t.o u.o -o test.so`
HP-UX	`ld -b t.o u.o -o test.so`
	When compiling your code, you must also use the `+z` flag to the HP C compiler.
AIX	`cc -bM:SRE -berok t.o u.o -o test.so -bE:ext.exp -lc`
	`The ext.exp file must be a text file with the name of a function that is externally accessible for each line.`

Loading Your Shared Object

After you've compiled your code, you need to tell the server to load the shared object and its functions so that you can begin using your plug-in functions in `obj.conf`.

When the server starts, it uses `obj.conf` to get its configuration information. To tell the server to load your shared object and functions in the shared object, you add this line to `obj.conf`:

```
Init fn=load-modules shlib=[path]filename.so funcs="function1,function1,...,functionN"
```

This initialization function opens the given shared object file and loads the functions *function1*, *function2*, and so on. You then use the functions *function1* and *function2* in the server configuration files (either `magnus.conf` or `obj.conf`). Remember to use the functions only with the directives for which you wrote them, as described in the following section.

Using Your Plug-in Functions

When you have compiled and arranged for the loading of your functions, you need to provide for their execution. All functions are called as follows:

Directive `fn=`*function* `[`*name1*=*value1*`]` ... `[`*nameN*=*valueN*`]`

- *Directive* identifies the class of function that is being called. Functions should not be called from the wrong directive!
- *fn=function* identifies the function to be called using the function's unique character-string name.

These two parameters are mandatory. After this, there may be an arbitrary number of function-specific parameters, each of which is a name-value pair.

You specify your function in the directive for which it was written. For example, the following line uses an **AddLog-class** plug-in function called **myaddlog** that adds an entry to a log file called mylogfile. The plug-in function accepts another parameter that defines how much information to log.

```
AddLog fn=myaddlog name="mylogfile" type="maxinfo"
```

Appendix A

Server Plug-in API Function Definitions

This chapter lists all the public functions and macros of the Server plug-in Applications Programming Interface (server plug-in API) in alphabetical order. Each description identifies the name of the function, its header file, its syntax, its parameters, an example of its use, and a list of related functions. Descriptions of the data structures that are not common to the C programming environment can be found in Appendix B, "Server Data Structures."

cache_digest (declared in libproxy/cache.h)

The **cache_digest** function calculates the MD5 signature of a specified URL and stores the signature in a digest variable.

Syntax

```
#include <libproxy/cache.h>
void cache_digest(char *url, unsigned char digest[16]));
```

Returns

```
void
```

Parameters

**char ***_url_ is a string containing the cache filename of a URL.

**name ***_digest_ is an array to receive first 48 bits of the signature of the URL.

See also

cache_fn_to_dig

cache_filename (declared in libproxy/cutil.h)

The **cache_filename** function returns the cache filename for a given URL, specified by MD5 signature.

Syntax

```
#include <libproxy/cutil.h>
char *cache_filename(unsigned char digest[16]);
```

Returns

A new string containing the cache filename.

Parameters

**char ***_digest_ is an array containing the MD5 signature of a URL.

See also

cache_fn_to_dig

cache_fn_to_dig (declared in libproxy/cutil.h)

The **cache_fn_to_dig** function converts a cache filename of a URL into a partial MD5 digest.

Syntax

```
#include <libproxy/cutil.h>
void *cache_fn_to_dig(char *name, unsigned char digest[16]));
```

Returns

```
void
```

Parameters

**char ***_name_ is a string containing the cache filename of a URL.

**name ***_digest_ is an array to receive first 48 bits of the signature of the URL.

ce_free (declared in libproxy/cache.h)

The **ce_free** function releases memory allocated by the **ce_lookup** function.

Syntax

```
#include <libproxy/cache.h>
void cd_free(CacheEntry *ce);
```

Returns

`void`

Parameters

**CacheEntry ***_ce_ is a cache entry structure to be destroyed.

See also

ce_lookup

ce_lookup (declared in libproxy/cache.h)

The **ce_lookup** cache entry lookup function looks up a cache entry for a specified URL.

Syntax

```
#include <libproxy/cache.h>
CacheEntry *ce_lookup(Session *sn, Request *rq, char *url,
time_t ims_c);
```

Returns

- NULL if caching is not enabled
- A newly allocated **CacheEntry** structure, whether or not a copy existed in the cache. Within that structure, the `ce->state` field reports about the existence:

  CACHE_NO signals that the document is not and will not be cached; other fields in the cache structure may be NULL

  CACHE_CREATE signals that the cache file doesn't exist but may be created once the remote server is contacted. However, during the retrieval it may turn out that the document is not cacheable.

  CACHE_REFRESH signals that the cache file exists, but it needs to be refreshed (an up-to-date check must be made) before it's used; note that the data may still be up-to-date, but the remote server needs to be contacted to find that out. If not, the cache file will be replaced with the new document version sent by the remote origin server.

CACHE_RETURN_FROM_CACHE signals that the cache file exists and is up-to-date based on the configuration and current parameters controlling what is considered fresh.

CACHE_RETURN_ERROR is a signal that happens only if the proxy is set to no-network mode (connect-Modenese), and the document does not exist in the cache.

Parameters

Session ***sn** identifies the Session structure.

Request ***rq** identifies the Request structure.

char ***url** contains the name of the URL for which the cache is being sought.

time-out *misc.* is the if-modified-since time.

See also

ce_free

cif_load (declared in libproxy/cif.h)

The **cif_load** function reads the directory-wide cache information file and stores the data in nodes about files.

Syntax

```
#include <libproxy/cif.h>
int cif_load(CacheEntry *arr, int start, int stop);
```

Returns

- nonzero if the write was successful
- 0 if the write was unsuccessful

Parameters

CacheEntry ***ce** is a cache entry structure to be written to the `.cif` file.

See also

cif_read_entry

cif_clear (declared in libproxy/cif.h)

The **cif_clear** function clears the **CacheEntry** block array so that it can be reused. This function does not write the data back into the cache information file.

Syntax

```
#include <libproxy/cif.h>
void cif_clear(CacheEntry *arr, int size);
```

Returns

```
void
```

Parameters

CacheEntry ***arr** is a cache entry array.

int *size* is the number of items in the array.

See also

cif_load_all_data, cif_load_cif

cif_load_all_data (declared in libproxy/cif.h)

The **cif_load_all_data** function loads the cache information file (CIF) entries from the CIF and then invokes the **stat** function for each file referred to by the CIF entries.

Syntax

```
#include <libproxy/cif.h>
int cif_laod_all_data (char *cif_fn, CacheEntry *arr, int max_entries);
```

Returns

The actual number of entries loaded into the array.

Parameters

**char ***cif_fn* is the absolute pathname for the CIF.

CacheEntry ***arr* is an array of empty **CacheEntry items to be loaded by this function.

int *max_entries* is the maximum number of elements that the array can accept.

See also

cif_clear, cif_load_cif

cif_load_cif (declared in libproxy/cif.h)

The **cif_load_cif** function loads an entire cache information file (CIF) into a specified empty array.

Syntax

```
#include <libproxy/cif.h>
int cif_laod_cif(char *cif_fn, CacheEntry *arr, int max_entries);
```

Returns

The actual number of entries loaded into the array.

Parameters

char **cif_fn* is the absolute pathname for the CIF.

**CacheEntry ****arr* is a cache entry array.

int *max_entries* is the maximum number of elements in the array.

See also

cif_clear

cif_read_entry (declared in libproxy/cif.h)

The **cif_read_entry** function reads the cache information file entry for a specified cache entry. There is rarely any need to use this function directly, because the **ce_lookup** function calls it.

Syntax

```
#include <libproxy/cif.h>
int cif_read_entry(CacheEntry *ce);
```

Returns

- 1 if the read was successful
- 0 if the read was unsuccessful

Parameters

**CacheEntry ****ce* is a cache entry structure to be read from the cache information file.

See also

cif_write_entry

cif_stat_entries (declared in libproxy/cif.h)

The **cif_stat_entries** function gathers information from the file system into the entries of a specified **CacheEntry** array. For each entry, it sets the removed member in the **CacheEntry** if the file no longer exists; otherwise it uses the **stat** function to establish the values for creation time, last access time, size in bytes, and size in blocks and stores these values in the finfo element of the array entry.

Syntax

```
#include <libproxy/cif.h>
int cif_stat_entries(CacheEntry *arr, int cnt);
```

Returns

the number of files that it examined with the **stat** function (*cnt* minus the number of files that were removed).

Parameters

**CacheEntry ****arr* is an existing CacheEntry array.

int *cnt* is the number of entries to obtain.

See also

cif_read_entry

cif_write_entry (declared in libproxy/cif.h)

The **cif_write_entry** function writes a CIF entry for a specified **CacheEntry** structure. The CIF entry is stored in the `.cif` file in the directory in which the cache file is located. Entries are appended to the CIF, so it can contain multiple cache entries, but the last one is the most current, and is the only one that you should use. The Cache Manager periodically rewrites the cache information files, removing old entries.

Syntax

```
#include <libproxy/cif.h>
int cif_write_entry(CacheEntry *ce);
```

Returns

- nonzero if the write was successful
- 0 if the write was unsuccessful

Parameters

**CacheEntry ****ce* is a cache entry structure to be written to the `.cif` file.

See also

cif_read_entry

cinfo_find (declared in base/cinfo.h)

The **cinfo_find** function finds the content information for the URI and returns a pointer to a structure containing the information. Use this function to retrieve information on a URI or a local filename in order to tell the client the type of file it will be receiving from the server.

Syntax

```
#include </base/cinfo.h>
cinfo *cinfo_find(char *uri);
```

Returns

- The pointer to a newly allocated **cinfo** structure if content was found
- NULL if no content was found

The **cinfo** structure that is allocated and returned contains information on the type of data in a file, if and how the data is encoded, and if the data is a textual document; it also contains information on the language of the text. The pointers contained in the returned **cinfo** structure point to data in the **types** database. It is important to note that you should not deallocate these pointers. You can change the data in your **cinfo** structure by first making a copy of the data to which the original structure elements points. However, you must deallocate your **cinfo** structure when you are done using it.

The **cinfo find** function can also be used with local filenames. Simply pass the local filename instead of the URI.

Parameters

**char ****uri* is the name of a URI. The filename specified by *uri* must be the string following the last backslash (/) in the URI. In addition, multiple filename extensions should be separated by CINFO_SEPARATOR (currently defined as a period).

condvar_init (declared in base/crit.h)

The **condvar_init** function is a critical-section function that initializes and returns a new condition variable associated with a specified critical-section variable. You can use the condition variable to manage the prevention of interference between two threads of execution.

Syntax

```
#include <base/crit.h>
CONDVAR condvar_init(CRITICAL id);
```

Returns

A newly allocated condition variable **(CONDVAR)**.

Parameters

CRITICAL *id* is a critical-section variable.

See also

condvar_notify, condvar_terminate, condvar_wait, crit_init.

condvar_notify (declared in base/crit.h)

The **condvar_notify** function is a critical-section function that awakens any threads that are blocked on the given critical-section variable. Use this function to awaken threads of execution of a given critical section. First, use **crit_enter** to gain ownership of the critical section. Then use the returned critical-section variable to call **condvar_notify** to awaken the threads. Finally, when **condvar_notify** returns, call **crit_exit** to surrender ownership of the critical section.

Syntax

```
#include <base/crit.h>
void condvar_notify(CONDVAR cv);
```

Returns

```
void
```

Parameters

CONDVAR *cv* is a condition variable.

See also

condvar_init, condvar_terminate, condvar_wait, crit_enter, crit_exit, crit_init.

condvar_terminate (declared in base/crit.h)

Critical-section function that frees a condition variable. Use this function to free a previously allocated condition variable.

CAUTION Terminating a condition variable that is in use can lead to unpredictable results.

Syntax

```
#include <base/crit.h>
void condvar_terminate(CONDVAR cv);
```

Returns

`void`

Parameters

CONDVAR *cv* is a condition variable.

See also

condvar_init, *condvar_notify*, *condvar_wait*, *crit_init*.

condvar_wait (declared in base/crit.h)

Critical-section function that blocks on a given condition variable. Use this function to wait for a critical section (specified by a condition variable argument) to become available. The calling thread is blocked until another thread calls **condvar_notify** with the same condition variable argument. The caller must have entered the critical section associated with this condition variable before calling **condvar_wait**.

Syntax

```
#include <base/crit.h>
void condvar_wait(CONDVAR cv);
```

Parameters

CONDVAR *cv* is a condition variable.

Returns

`void`

See also

condvar_init, condvar_notify, condvar_terminate, crit_init.

crit_enter (declared in base/crit.h)

Critical-section function that attempts to enter a critical section. Use this function to gain ownership of a critical section. If another thread already owns the section, the calling thread is blocked until the first thread surrenders ownership by calling **crit_exit**.

Syntax

```
#include <base/crit.h>
void crit_enter(CRITICAL crvar);
```

Returns

```
void
```

Parameters

CRITICAL

daemon_atrestart (declared in netsite.h)

The **daemon_atrestart** function lets you register a callback function named by *fn* to be used when the server receives a restart signal. Use this function when you need a callback function to deallocate resources allocated by an initialization function. The **daemon_atrestart** function is a generalization of the **magnus_atrestart** function.

Syntax

```
#include <netsite.h>
void daemon_atrestart(void (*fn)(void *), void *data);
```

Returns

```
void
```

Parameters

void (* *fn*) (void *****) is the callback function.

**void ****data* is the parameter passed to the callback function when the server is restarted.

Example

```
/* Close log file when server is restarted */
daemon_atrestart(brief_terminate, NULL);
return REQPROCEED;
```

See also

http_start_response.

fast_dump_cif (declared in libproxy/cif.h)

The **fast_dump_cif** function dumps all CIF entries from a string array into an existing CIF file.

Syntax

```
#include <libproxy/cif.h>
int fast_dump_cif(char *cif_fn, char **arr, int cnt);
```

Returns

The number of entries dumped into the array (*cnt* minus any NULLs).

Parameters

**char ****cif_fn* is the absolute pathname for the CIF.

char ***arr* is a string array containing all the entries. Each must include a trailing `/n` character, or the file will become corrupted.

int *cnt* is the number of entries in the array. If any entry is NULL, the function ignores the *cnt* value.

See also

cif_write_entry

fast_get_cif_entry_for (declared in libproxy/cif.h)

The **fast_get_cif_entry_for** function gets the CIF entry for a specific cache file. It accepts the entry as a single string, without parsing it.

Syntax

```
#include <libproxy/cif.h>
char *fast_get_cif_entry_for(char *cif_fn, char *name);
```

Returns

A string containing the CIF entry.

Parameters

**char ***_cif_fn_ is the absolute pathname for the CIF.

char *_name_ is the relative name of the cache file (relative to the directory containing the CIF).

See also

_cif_read_entry_

fast_load_cif (declared in libproxy/cif.h)

The **fast_load_cif** function loads all CIF entries as unparsed strings into an array.

Syntax

```
#include <libproxy/cif.h>
int fast_load_cif(char *cif_fn, char **arr, int max, int *multiples_ret);
```

Returns

- the number of entries loaded into the array if the operation was successful
- 0 if the operation was unsuccessful

Parameters

**char ***_cif_fn_ is the absolute pathname for the CIF.

char **_arr_ is a string array where the function will place the entries. Each will include a trailing **/n** character.

int _max_ is the maximum number of entries the array can receive.

int *_multiples_ret_ is an integer to receive the number of multiple entries encountered.

See also

_cif_write_entry_

fast_put_cif_entry (declared in libproxy/cif.h)

The **fast_put_cif_entry** function writes a properly formatted CIF entry to the specified CIF.

Syntax

```
#include <libproxy/cif.h>
int fast_put_cif_entry(char *cif_fn, char *entry);
```

Returns

- nonzero if the write was successful
- 0 if the write was unsuccessful

Parameters

**char ***_cif_fn_ is the absolute pathname for the CIF.

char *_entry_ is the formatted CIF entry, including the trailing \n character.

See also

cif_write_entry

filebuf_buf2sd (declared in base/buffer.h)

The **filebuf_buf2sd** function sends a file buffer to a socket and returns the number of bytes sent.

Use this function to send the contents of a file to a server.

Syntax

```
#include <base/buffer.h>
int filebuf_buf2sd(filebuf *buf, SYS_NETFD sd);
```

Returns

- The number of bytes sent to the socket, if successful
- The constant IO_ERROR if the file buffer could not be sent

Parameters

**filebuf ***_buf_ is the name of the file buffer.

SYS_NETFD *sd* is the platform-independent identifier of the socket.

Example

```
if(filebuf_buf2sd(buf, sn->csd) == IO_ERROR)
   ret = REQ_EXIT;
filebuf_close(buf);
```

See also

filebuf_close, filebuf_open, netbuf_buf2sd

filebuf_close (declared in base/buffer.h)

The **filebuf_close** function deallocates a file buffer and closes its associated files.

Generally, use **filebuf_open** to first open a file buffer and then use **filebuf_getc** to access the information in the file. After you have finished using the file buffer, use **filebuf_close** to close it.

Syntax

```
#include <base/buffer.h>
void filebuf_close(filebuf *buf);
```

Returns

void

Parameters

filebuf * *buf* is the name of the file buffer.

Example

```
if(filebuf_buf2sd(buf, sn->csd) == IO_ERROR)
   ret = REQ_EXIT;
filebuf_close(buf);
```

See also

filebuf_buf2sd, filebuf_getc, filebuf_open, netbuf_close

filebuf_getc (declared in base/buffer.h)

The **filebuf_getc** function retrieves a character from the current cursor position and returns an integer.

Use **filebuf_getc** to sequentially read one character from the file buffer.

Syntax

```
#include <base/buffer.h>
netbuf_getc(netbuf b);
```

Returns

- An integer representation of the character retrieved
- The constant IO_EOF or IO_ERROR upon an end of file or error

Parameters

netbuf *b* is the name of the file buffer.

See also

filebuf_close, netbuf_getc, netbuf_open

filebuf_open (declared in base/buffer.h)

The **filebuf_open** function opens a new file buffer and returns a pointer to the buffer. Use this function to read through a file using a buffer. This function provides more efficient file access because using the function guarantees use of buffered file I/O in environments where it is not supported by the operating system.

Syntax

```
#include <base/buffer.h>
filebuf *filebuf_open(SYS_FILE fd, int sz);
```

Returns

- A pointer to a new buffer structure to hold the data, if one was created
- NULL if no buffer could be opened

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

int *sz* is the size, in characters, to be used for the buffer.

Example

```
buf = filebuf_open(fd, &finfo);
if (!buf){
   system_fclose(fd);
   goto done;
}
```

See also

filebuf_close, filebuf_open_nostat, netbuf_open

filebuf_open_nostat (declared in base/buffer.h)

The **filebuf_open_nostat** function opens a new file buffer and returns a new buffer structure. This function accomplishes the same purpose as the **filebuf_open** function but is more efficient because it does not need to call the **stat** function.

Syntax

```
#include <base/buffer.h>
#include <sys/stat.h>
filebuf* filebuf_open_nostat(SYS_FILE fd, int sz, struct stat *finfo);
```

Returns

- A pointer to a new buffer structure to hold the data, if one was created
- NULL if no buffer could be opened

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

int *sz* is the file descriptor to be opened.

struct stat ****finfo* is the file descriptor to be opened. Before calling the **filebuf_open_nostat function, you must call the **stat** function for the file, so that the parameter returned by the **stat** function (specified by *finfo*) has been established.

Example

```
buf = filebuf_open_nostat(fd, FILE_BUFFERSIZE, &finfo);
if (!buf){
   system_fclose(fd);
   goto done;
}
```

See also

filebuf_close, filebuf_open

FREE (declared in netsite.h)

The **FREE** macro is a platform-independent substitute for the C library routine **free**. It deallocates the space previously allocated by MALLOC or STRDUP to a specified pointer.

Syntax

```
#include <netsite.h>
FREE(ptr);
```

Returns

```
void
```

Parameters

ptr is a (void) pointer to an object. If the pointer is not one created by MALLOC or STRDUP, the behavior is undefined.

Example

```
if(alt) {
   pb_param *pp = pblock_find("ppath", rq->vars);
   /* Trash the old value */
   FREE(pp->value);
   /* Dup it because the library will later free this pblock */
   pp->value = STRDUP(alt);
   return REQ_PROCEED;
}
/* Else do nothing */
return REQ_NOACTION;
```

See also

MALLOC, REALLOC, STRDUP

fs_blks_available (declared in libproxy/fs.h)

The **fs_blks_available** function returns the number of disk blocks available on the disk partition on which a specified directory resides.

Syntax

```
#include <libproxy/fs.h>
long fs_blks_avail(char *root);
```

Returns

The number of available disk blocks

Parameters

**char ***_root_ is the name of the directory.

See also

fs_blk_size

fs_blk_size (declared in libproxy/fs.h)

The **fs_blk_size** function returns the block size of the disk partition on which a specified directory resides.

Syntax

```
#include <libproxy/fs.h>
long fs_blk_size(char *root);
```

Returns

the block size, in bytes

Parameters

**char ***_root_ is the name of the directory.

See also

fs_blks_available

func_exec (declared in frame/func.h)

The **func_exec** function executes the function named by the *fn* entry in a specified parameter block, for a specified Session and a specified Request. If the function name is not found, the func_exec function creates a LOG_MISCONFIG message for the missing function parameter.

You can use this function to execute a server application function (SAF) by identifying it in the parameter block.

Syntax

```
#include <frame/func.h>
int func_exec(pblock *pb, Session *sn, Request *rq);
```

Returns

- The value returned by the executed function.

- The constant REQ_ABORTED if no function was executed

Parameters

pblock * *pb* is the parameter block containing the function.

Session * *sn* identifies the Session structure.

Request * *rq* identifies the Request structure.

The **Session** and **Request** parameters can be the same as the ones passed to your function.

See also

log_error

func_find (declared in frame/func.h)

The **func_find** function returns a pointer to the function specified by name. If no pointer exists, the function returns NULL.

Syntax

```
#include <frame/func.h>
FuncPtr func_find(char *name);
```

Returns

- A pointer to the chosen function, suitable for dereferencing.
- NULL if the function could not be found.

Parameters

char * *name* is the name of the function.

Example

```
/* this block of code does the same thing as func_exec */
   char *afunc = pblock_findval("afunction", pb);
   FuncPtr afnptr = func_find(afunc);
   if(afnptr) return (afnptr)(pb, sn, rq);
```

See also

func_exec

http_dump822 (declared in frame/http.h)

Utility function that prints headers into a buffer and returns it.

The **http_dump822** function prints headers from the parameter block named by *pb* into a buffer named by *t*, with the size and position specified by *tsz* and *pos*, respectively.

Use this function to serialize the headers so that they can be sent, for example, in a mail message.

Syntax

```
#include <frame/http.h>
char *http_dump822(pblock *pb, char *t, int *pos, int tsz);
```

Returns

The buffer, reallocated if necessary, and modifies *pos* to denote a new position in the buffer.

See also

http_handle_session, *http_scan_headers*, *http_start_response*, *protocol_status*

http_hdrs2env (declared in frame/http.h)

Utility function that converts a parameter block entry into an enviroment.

The **http_hdrs2env** function takes the entries in the parameter block named by *pb* and converts them to an environment.

Note that each entry is converted to uppercase text with the prefix HTTP_.

A hyphen (-) or double hyphen (--) in the text is automatically converted into an underscore (_), or double underscore (_ _), respectively.

Use this function to create an environment that a program can later use.

Syntax

```
#include <frame/http.h>
char **http_hdrs2env(pblock *pb);
```

Returns

A pointer to the new environment.

See also

http_handle_session, *http_scan_headers*, *http_start_response*, *protocol_status*

# http_scan_headers (declared in frame/http.h)

Utility function that scans HTTP headers from a network buffer and places them in a parameter block.

Scans HTTP headers from the network buffer named by *buf*, and places them in the parameter block named by *headers*.

The **Session** structure named by *sn* contains a pointer to a netbuf called *inbuf*. If the parameter *buf* is NULL, the function automatically uses *inbuf*.

Folded lines are joined and the linefeeds are removed (but not the whitespace). If there are any repeat headers, they are joined and the two field bodies are separated by a comma and space. For example, multiple mail headers are combined into one header and a comma is used to separate the field bodies.

The parameter *t* defines a string of length REQ_MAX_LINE. This is an optimization for the internal code to reduce usage of runtime stack.

Note that *sn* is an optional parameter that is used for error logs. Use NULL if you wish.

Syntax

```
#include <frame/http.h>
int http_scan_headers(Session *sn, netbuf *buf, char *t,
pblock *headers);
```

Returns

- The constant REQ_PROCEED if the operation succeeded
- The constant REQ_ABORTED if the operation did not succeed

See also

http_handle_session, *http_start_response*, *protocol_status*, *protocol_scan_headers*

http_set_finfo (declared in frame/http.h)

Utility function that retrieves HTTP information about a file being sent to a client.

The **http_set_finfo** function retrieves the length and date from the **stat** structure named by *finfo*, for the Session named by *sn* and the request denoted by *rq*.

Note that the **stat** structure contains the information about the file you are sending back to the client.

Use **http_set_finfo** only after receiving a start_response from a service class server application function (SAF).

Syntax

```
#include <frame/http.h>
int http_set_finfo(Session *sn, Request *rq, struct stat *finfo);
```

Returns

- The constant REQ_PROCEED if the request can proceed normally
- The constant REQ_ABORTED if the function should treat the request normally, but not send any output to the client

See also

http_handle_session, *http_scan_headers*, *http_start_response*, *protocol_status*, *protocol_set_finfo*

http_start_response (declared in frame/http.h)

Utility function that initiates the HTTP response.

The **http_start_response** function initiates the HTTP response for the Session named by *sn* and the request denoted by *rq*. If the protocol version is HTTP / 0.9, the function does nothing. If the protocol version is HTTP / 1.0, the function sends a header.

Note that if the return value is REQ_NOACTION, you should not send the data you were going to send in response to the request. Otherwise, **http_start_response** will return REQ_PROCEED.

Use this function to set up HTTP and prepare the server and the client to receive data.

Syntax

```
#include <frame/http.h>
int http_start_response(Session *sn, Request *rq);
```

Returns

- The constant REQ_PROCEED if the operation succeeded, in which case you can send the data you were preparing to send
- The constant REQ_NOACTION if the operation succeeded, but the client has requested that the server not send the data because the client has it in cache.

- The constant REQ_ABORTED if the operation did not succeed.

See also

http_handle_session, *http_scan_headers*, *protocol_status*, *protocol_start_response*

http_status (declared in frame/http.h)

Utility function that sets session status and reason string.

The **http_status** function sets the session status to indicate whether an error condition occurred.

If the reason string is NULL, the server attempts to find a reason string for the given status code. If it finds none, it returns "Unknown reason."

Use this function to check the status of the Session before calling the function **start_response**.

The following is a list of valid status codes:

```
PROTOCOL_OK
PROTOCOL_NO_RESPONSE
PROTOCOL_REDIRECT
PROTOCOL_NOT_MODIFIED
PROTOCOL_BAD_REQUEST
PROTOCOL_UNAUTHORIZED
PROTOCOL_FORBIDDEN
PROTOCOL_NOT_FOUND
PROTOCOL_PROXY_UNAUTHORIZED
PROTOCOL_SERVER_ERROR
PROTOCOL_NOT_IMPLEMENTED
```

Syntax

```
#include <frame/http.h>
void http_status(Session *sn, Request *rq, int n, char *r);
```

Returns

void, but it sets values in the session / request designated by *sn* / *rq* for the status code and the reason string

See also

http_handle_session, *http_scan_headers*, *http_start_response*, *protocol_status*

http_uri2url (declared in frame/http.h)

Utility function that converts URI to URL.

The **http_uri2url** function takes the given URI *prefix* and *suffix*, and creates a newly-allocated full URL in the form http://(server):(port)(*prefix*)(*suffix*).

If you want to skip either the URI prefix or suffix, use NULL as the value for either parameter. To redirect the client somewhere else, use the function **pblock_nvinsert** to create a new entry in the vars in the pblock in your request structure.

Use **http_uri2url** when you want to convert from URI to URL in order to pass a fully qualified resource locator to a client.

Syntax

```
#include <frame/http.h>
char *http_uri2url(char *prefix, char *suffix);
```

Returns

A new string containing the URL

See also

http_handle_session, *http_scan_headers*, *http_start_response*, *protocol_status*, *protocol_uri2url*

log_error (declared in frame/log.h)

The **log_error** function creates an entry in an error log, recording the date, the severity, and a specified text.

Syntax

```
#include <frame/log.h>
int log_error(int degree, char *func, Session *sn, Request *rq,
char *fmt, ...);
```

Returns

- 0 if the log entry was created.
- -1 if the log entry was not created.

Parameters

int *degree* specifies the severity of the error. It must be one of the following constants:

LOG_WARN — warning
LOG_MISCONFIG — a syntax error or permission violation
LOG_SECURITY—- an authentication failure or 403 error from a host
LOG_FAILURE — an internal problem
LOG_CATASTROPHE — a non-recoverable server error
LOG_INFORM — an informational message

**char ***func* is the name of the function where the error occurred.

**Session ***sn* identifies the Session structure.

**Request ***rq* identifies the Request structure.

char ***fmt* specifies the format for the **printf function that delivers the message.

... represents a sequence of parameters for the **printf** function.

Example

```
if(!groupbuf) {
   log_error(LOG_WARN, "send-file", sn, rq,
      "error opening buffer from %s (%s)"), path,
         system_errmsg(fd));;
   return REQ_ABORTED;
}
```

See also

func_exec

magnus_atrestart (declared in netsite.h)

NOTE	Use the **daemon-atrestart** function in place of the obsolete **magnus_atrestart** function.

The **magnus_atrestart** function lets you register a callback function named by *fn* to be used when the server receives a restart signal. Use this function when you need a callback function to deallocate resources allocated by an initialization function.

Syntax

```
#include <netsite.h>
void magnus_atrestart(void (*fn)(void *), void *data);
```

Returns

```
void
```

Parameters

void *(* fn) (void *)* is the callback function.

**void ****data* is the parameter passed to the callback function when the server is restarted.

Example

```
/* Close log file when server is restarted */
magnus_atrestart(brief_terminate, NULL);
return REQPROCEED;
```

make_log_time (declared in libproxy/util.h)

The **make_log_time** function translates a given time from **time_t** format to a character format suitable for access logs. It can also deliver the current time in the access log format.

Syntax

```
#include <libproxy/util.h>
char *make_log_time(time_t tt);
```

Returns

- the character equivalent of the specified time *tt*, if *tt* is not 0
- the current local time, in character format if *tt* is 0

Parameters

time_t *tt* is a time.

MALLOC (declared in netsite.h)

The **MALLOC** macro is a platform-independent substitute for the C library routine **malloc**. It uses memory pools, creating one for each request, automatically freeing it after the request has been processed. The data in the Request parameter block is allocated by MALLOC, not PERM_MALLOC, which provides allocation that persists beyond the end of the request. If memory pooling has been disabled in the configuration file, PERM_MALLOC and MALLOC both obtain their memory from the system heap.

Syntax

```
#include <netsite.h>
MALLOC(size)
```

Returns

A pointer to space for an object of size *size*.

Parameters

size (an int) is the number of bytes to allocate.

Example

```
/* Initialize hosts array */
    num_hosts = 0;
    hosts = (char **) MALLOC(1 * sizeof(char *));
    hosts[0] = NULL;
```

See also

PERM_MALLOC, REALLOC, FREE, PERM_FREE, STRDUP, PERM_STRDUP

netbuf_buf2sd (declared in base/buffer.h)

The **netbuf_buf2sd** function sends a buffer to a socket. You can use this function to send data from IPC pipes to the client.

Syntax

```
#include <base/buffer.h>
int netbuf_buf2sd(netbuf *buf, SYSNETFD sd, int len);
```

Returns

- The number of bytes transferred to the socket, if successful
- The constant IO_ERROR if unsuccessful

Parameters

netbuf * *buf* is the buffer to send.

SYS_NETFD *sd* is the platform-independent socket identifier.

int *len* is the buffer length.

See also

filebuf_buf2sd, *netbuf_close*, *netbuf_grab*, *netbuf_open*

netbuf_close (declared in base/buffer.h)

The **netbuf_close** function deallocates a network buffer and closes its associated files. Use this function when you need to deallocate the network buffer and close the socket.

You should never close the **netbuf** parameter in a Session structure.

Syntax

```
#include <base/buffer.h>
void netbuf_close(netbuf *buf);
```

Returns

```
void
```

Parameters

netbuf * *buf* is the buffer to close.

See also

filebuf_close, *netbuf_grab*, *netbuf_open*

netbuf_getc (declared in base/buffer.h)

The **netbuf_getc** function retrieves a character from the cursor position of the network buffer specified by *b*.

Syntax

```
#include <base/buffer.h>
netbuf_getc(netbuf b);
```

Returns

- The integer representing the character, if one was retrieved
- The constant IO_EOF or IO_ERROR, for end of file or error

Parameters

netbuf *b* is the buffer from which to retrieve one character.

See also

filebuf_getc, *netbuf_grab*, *netbuf_open*

netbuf_grab (declared in base/buffer.h)

The **netbuf_grab** function assigns a size to the array in the network buffer named by *buf*. The size of the array is specified by *sz*, which is the number of bytes from the buffer's associated object.

The buffer processes the allocation and deallocation of the array.

This function is used by the function **netbuf_buf2sd**.

Syntax

```
#include <base/buffer.h>
int netbuf_grab(netbuf *buf, int sz);
```

Returns

- The number of bytes actually read (from 1through *sz*), if the assignment was successful
- The constant IO_EOF or IO_ERROR, for end of file or error

Parameters

**netbuf ***buf* is the buffer into which to read.

int *sz* is the array size for the buffer to allocate.

See also

netbuf_close, *netbuf_open*

netbuf_open (declared in base/buffer.h)

The **netbuf_open** function opens a new network buffer and returns it. You can use **netbuf_open** to create a **netbuf** structure and start using buffered I/O on a socket.

Syntax

```
#include <base/buffer.h>
netbuf* netbuf_open(SYS_NETFD sd, int sz);
```

Returns

A new **netbuf** structure (network buffer)

Parameters

SYS_NETFD *sd* is the platform-independent socket identifier.

int *sz* is the number of characters to allocate for the network buffer.

See also

filebuf_open, *netbuf_close*, *netbuf_grab*

net_ip2host (base/net.h)

The **net_ip2host** function transforms a textual IP address into a fully qualified domain name and returns it.

Syntax

```
#include <base/net.h>
char *net_ip2host(char *ip, int verify);
```

Returns

- A new string containing the fully qualified domain name, if the transformation was accomplished.
- NULL if the transformation was not accomplished.

Parameters

**char ***ip* is the IP address as a character string in dotted-decimal notation: *nnn.nnn.nnn.nnn*

int *verify*, if nonzero, specifies that the function should verify the fully qualified domain name. Though this requires an extra query, you should use it when determining access control.

See also

net_sendmail

net_read (declared in base/net.h)

The **net_read** function reads bytes from a specified socket into a specified buffer. The function waits to receive data from the socket until either at least one byte is available in the socket or the specified time has elapsed.

Syntax

```
#include <base/net.h
int net_read (SYS_NETFD sd, char *buf, int sz, int timeout);
```

Returns

- The number of bytes read, which will not exceed the maximum size, *sz*.
- A negative value if an error has occurred, in which case *errno* is set to the constant ETIMEDOUT if the operation did not complete before *timeout* seconds elapsed.

Parameters

SYS_NETFD *sd* is the platform-independent socket descriptor.

**char ***buf* is the buffer to receive the bytes.

int *sz* is the maximum number of bytes to read.

int *timeout* is the number of seconds to allow for the read operation before returning. The purpose of *timeout* is not to return because not enough bytes were read in the given time but to limit the amount of time devoted to waiting until some data arrives.

See also

net_socket, *net_write*

net_socket (declared in base/net.h)

The **net_socket** function opens a connection to a socket, creating a new socket descriptor. The socket is not connected to anything, and is not listening to any port. A function must use **net_connect** to make a connection, and **net_accept** to listen.

Syntax

```
#include <base/net.h>
SYS_NETFD net_socket (int domain, int type, int protocol);
```

Returns

The platform-independent socket descriptor (**SYS_NETFD**) associated with the socket.

Parameters

int *domain* must be the constant AF_INET.

int *type* must be the constant SOCK_STREAM.

int *protocol* must be the constant IPPROTO_TCP.

See also

net_read, net_write

net_write (declared in base/net.h)

The **net_write** function writes a specified number of bytes to a specified socket into a specified buffer. It returns the number of bytes written.

Syntax

```
#include <base/net.h>
int net_write (SYS_NETFD sd, char *buf, int sz);
```

Returns

The number of bytes written, which may be less than the requested size if an error occurred.

Parameters

SYS_NETFD *sd* is the platform-independent socket descriptor.

**char ****buf* is the buffer containing the bytes.

int *sz* is the number of bytes to write.

Example

```
/* Start response by giving boundary string */
if(net_write(sn->csd, FIRSTMSG, strlen(FIRSTMSG)) == IO_ERROR)
   return REQ_EXIT;
```

See also

net_socket

param_create (declared in base/pblock.h)

The **param_create** function creates a parameter block structure containing a specified name and value. If the name or value is not NULL, the pair is copied and placed into the new parameter block structure; otherwise the pair is created with name and value both. Use this function to prepare a parameter block structure to be used in calls to parameter block routines such as **pblock_pinsert**.

Syntax

```
#include <base/pblock.h>
pb_param *param_create(char *name, char *value);
```

Returns

A new parameter block structure.

Parameters

**char ****name* is the string containing the name portion of the name-value pair.

**char ****value* is the string containing the value portion of the name-value pair.

Example

```
pblock *pb = pblock_create(4);
pb_param *newpp = param_create("hello","world");
pblock_pinsert(newpp, pb);
```

See also

param_free

param_free (declared in base/pblock.h)

The **param_free** function frees the parameter specified by *pp*. Use the **param_free** function for error checking after removing the function using **pblock_remove**.

Syntax

```
#include <base/pblock.h>
int param_free(pb_param *pp);
```

Returns

- 1 if the parameter was freed
- 0 if the parameter was NULL

Parameters

**pb_param ***pp* is the name portion of a name-value pair stored in a pblock.

Example

```
int check(pblock *pb)
{
    if(param_free(pblock_remove("hello", pb)))
      return 1; /* signal that we removed it */
   else
      return 0; /* We didn't remove it. */
}
```

See also

param_create, pblock_remove

pblock_copy (declared in base/pblock.h)

The **pblock_copy** function copies the contents of one parameter block into another.

Syntax

```
#include <base/pblock.h>
void pblock_copy(pblock *src, pblock *dst);
```

Returns

```
void
```

Parameters

**pblock ***src* is the source parameter block.

**pblock ***dst* is the destination parameter block.

Both entries are newly allocated so that the original parameter block may be freed, or the new parameter block changed, without affecting the other parameter block.

See also

pblock_create, pblock_dup, pblock_free, pblock_find, pblock_remove, pblock_nvinsert

pblock_create (declared in base/pblock.h)

The **pblock_create** function creates a new parameter block. The system maintains an internal hash table for fast name-value pair lookups.

Syntax

```
#include <base/pblock.h>
pblock *pblock_create(int n);
```

Returns

The newly allocated parameter block.

Parameters

int *n* is the size of the hash table (number of name-value pairs) for the parameter block.

See also

pblock_copy, pblock_dup, pblock_find, pblock_free, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_dup (declared in base/pblock.h)

The **pblock_dup** function duplicates a parameter block. It is equivalent to a sequence of **pblock_create** and **pblock_copy**.

Syntax

```
#include <base/pblock.h>
pblock pblock_dup(pblock *src);
```

Returns

The newly allocated parameter block.

Parameters

**pblock ****src* is the source parameter block.

See also

pblock_create, *pblock_free*, *pblock_find*, *pblock_remove*, *pblock_nvinsert*

pblock_find (declared in base/pblock.h)

The **pblock_find** function finds a specified name-value pair entry in a parameter block and retrieves the name and structure of the parameter block. If you want only the value of the parameter block, use only the function **pblock_findval** to get the actual value in the name-value pair.

Note that this function is implemented as a macro.

Syntax

```
#include <base/pblock.h>
pb_param *pblock_find(char *name, pblock *pb);
```

Returns

- A parameter block structure, if one was found
- NULL if no parameter block was found

Parameters

**char ***name* is the name of a name-value pair.

**pblock ***pb* is the parameter block to be searched.

See also

pblock_copy, pblock_findval, pblock_free, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_findlong (declared in libproxy/util.h)

The **pblock_findlong** function finds a specified name-value pair entry in a parameter block, and retrieves the name and structure of the parameter block. Use **pblock_findlong** if you want to retrieve the name, structure, and value of the parameter block. However, if you want only the name and structure of the parameter block, use the **pblock_find** function. Do not use these two functions in conjunction.

Syntax

```
#include <libproxy/util.h>
long pblock_findlong(char *name, pblock *pb);
```

Returns

- A **long** containing the value associated with the name
- -1 if no match was found

Parameters

**char ***_name_ is the name of a name-value pair.

**pblock ***_pb_ is the parameter block to be searched.

See also

pblock_nlinsert

pblock_findval (declared in base/pblock.h)

The **pblock_findval** function finds a specified name-value pair entry in a parameter block. Use **pblock_findval** if you want to retrieve the name, structure, and value of the parameter block. However, if you want just the name and structure of the parameter block, use only the macro **pblock_find**.

Syntax

```
#include <base/pblock.h>
char *pblock_findval(char *name, pblock *pb);
```

Returns

- A string containing the value associated with the name
- NULL if no match was found

Parameters

**char ***_name_ is the name of a name-value pair.

**pblock ***_pb_ is the parameter block to be searched.

Example

See **pblock_nvinsert**.

See also

pblock_copy, pblock_find, pblock_free, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_free (declared in base/pblock.h)

The **pblock_free** function frees a specified parameter block and any entries inside it. If you want to save a variable in the parameter block, remove the variable using the function **pblock_remove** and then save the resulting pointer.

Syntax

```
#include <base/pblock.h>
void pblock_free(pblock *pb);
```

Returns

`void`

Parameters

pblock ***pb** is the parameter block to be freed.

See also

pblock_copy, pblock_create, pblock_find, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_nlinsert (declared in libproxy/util.h)

The **pblock_nlinsert** function creates a new parameter structure with a given name and long numeric value and inserts it into a specified parameter block. The name and value parameters are also newly allocated.

Syntax

```
#include <libproxy/util.h>
pb_param *pblock_nlinsert(char *name, long value, pblock *pb);
```

Returns

The newly allocated parameter block structure

Parameters

char ***name** is the name by which the name-value pair is stored.

long *value* is the long (or integer) value being inserted into the parameter block.

pblock ***pb** is the parameter block into which the insertion occurs.

See also

pblock_findlong

pblock_nninsert (declared in base/pblock.h)

The **pblock_nninsert** function creates a new parameter structure with a given name and a numeric value and inserts it into a specified parameter block. The name and value parameters are also newly allocated.

Syntax

```
#include <base/pblock.h>
pb_param *pblick_nninsert(char *name, int value, pblock *pb);
```

Returns

The new parameter block structure

Parameters

char ***name** is the name by which the name-value pair is stored.

int *value* is the numeric value being inserted into the parameter block.

> The **pblock_nninsert** function requires that the parameter *value* be an integer. If the value you assign is not a number, then instead use the function **pblock_nvinsert** to create the parameter.

pblock ***pb** is the parameter block into which the insertion occurs.

See also

pblock_copy, *pblock_create*, *pblock_find*, *pblock_free*, *pblock_nvinsert*, *pblock_remove*, *pblock_str2pblock*

pblock_nvinsert (declared in base/pblock.h)

The **pblock_nvinsert** function creates a new parameter structure with a given name and character value and inserts it into a specified parameter block. The name and value parameters are also newly allocated.

You could use this function when an error condition is encountered, in order to insert an error into the parameter block argument and to tell initialization routines in the server that an error occurred.

Syntax

```
#include <base/pblock.h>
pb_param *pblock_nvinsert(char *name, char *value, pblock *pb);
```

Returns

The newly allocated parameter block structure

Parameters

**char ***_name_ is the name by which the name-value pair is stored.

**char ***_value_ is the string value being inserted into the parameter block.

**pblock ***_pb_ is the parameter block into which the insertion occurs.

Example

```
int brief_init(pblock *pb, Session *sn, Request *rq)
{
/* find "find" value in the parameter blcock */
    char *fn = pblock_findval("file", pb);
   /* if "file" is not found, insert an "error" value
      asking to supply a filename*/
      if(!fn) {
         pblock_nvinsert("error",
            "brief-init: please supply a filename", pb);
         return REQ_ABORTED;
      }
      /* open a file in write/append mode*/
      logfd = system_fopenWA(fn);
      if(logfd == SYS_ERROR_FD) {
         pblock_nvinsert("error",
         "brief-init: please supply a filename", pb);
      return REQ_ABORTED;
}
```

See also

pblock_copy, *pblock_create*, *pblock_find*, *pblock_free*, *pblock_nninsert*, *pblock_remove*, *pblock_str2pblock*

pblock_pb2env (declared in base/pblock.h)

The **pblock_pb2env** function copies a specified parameter block into a specified environment. The function creates one new environment entry for each name-value pair in the parameter block. Use this function to send pblock entries to a program that you are going to execute.

Syntax

```
#include <base/pblock.h>
char **pblock_pb2env(pblock *pb, char **env);
```

Returns

A pointer to the array of name-value pairs.

Parameters

pblock ***pb** is the parameter block to be copied.

char ****env** is the environment into which the parameter block is to be copied.

See also

pblock_copy, pblock_create, pblock_find, pblock_free, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_pblock2str (declared in base/pblock.h)

The **pblock_pblock2str** function copies all parameters of a specified parameter block into a specified string. The function allocates additional non-heap space for the string if needed.

Use this function to stream the parameter block for archival and other purposes.

Syntax

```
#include <base/pblock.h>
char *pblock_pblock2str(pblock *pb, char *str);
```

Returns

The new version of the *str* parameter. If *str* was NULL, this is a new string; otherwise it is a reallocated string. In either case, it is allocated in the pool of memory established for the current request.

Parameters

pblock ***pb** is the parameter block to be copied.

char ***str** is the string into which the parameter block is to be copied. It must have been allocated by MALLOC or REALLOC, not by PERM_MALLOC or PERM_REALLOC (which allocate from the heap).

Each name-value pair in the string is separated from its neighbor pair by a space and is in the format `name="value"`.

See also

pblock_copy, pblock_create, pblock_find, pblock_free, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_pinsert (declared in base/pblock.h)

This function can be used instead of the function **pblock_nvinsert**. Both functions insert an error into the parameter block argument to tell initialization routines in the server that an error occurred. However, **pblock_pinsert** is more convenient if you want to insert several parameter structures.

Syntax

```
#include <base/pblock.h>
void pblock_pinsert(pb_param *pp, pblock *pb);
```

Returns

```
void
```

Parameters

**pb_param ***_pp_ is the parameter to insert.

**pblock ***_pb_ is the parameter block.

See also

pblock_copy, pblock_create, pblock_find, pblock_free, pblock_nvinsert, pblock_remove, pblock_str2pblock

pblock_remove (declared in base/pblock.h)

The **pblock_remove** function removes a specified name-value entry from a specified parameter block.

Note that this function is implemented as a macro. Furthermore, if you use this macro your code must eventually call **param_free** in order to deallocate the memory used by the parameter structure.

Syntax

```
#include <base/pblock.h>
pb_param *pblock_remove(char *name, pblock *pb);
```

Returns

- The removed parameter block structure, if it was found
- NULL if no parameter block was found

Parameters

**char ***_name_ is the name portion that identifies the name-value pair to be removed.

pblock ***pb*** is the from which the name-value entry is to be removed. See **pblock_free**.

See also

pblock_copy, pblock_create, pblock_find, pblock_free, pblock_nvinsert, pblock_str2pblock

pblock_replace_name (declared in libproxy/util.h)

The **pblock_replace_name** function replaces the name of a name-value pair, retaining the value.

Syntax

```
#include <libproxy/util.h>
void pblock_replace_name(char *oname,char *nname, pblock *pb);
```

Returns

```
void
```

Parameters

char ***oname*** is the old name of a name-value pair.

char ***nname*** is the new name for the name-value pair.

pblock ***pb*** is the parameter block to be searched.

See also

pblock_remove

pblock_str2pblock (declared in base/pblock.h)

The **pblock_str2pblock** function scans a string for parameter pairs, adds the value to a parameter block, and returns the number of parameters added.

Syntax

```
#include <base/pblock.h>
int pblock_str2pblock(char *str, pblock *pb);
```

Returns

- The number of parameters pair added to the parameter block, if any
- -1 if an error occurred

Parameters

**char ****str* is the string to be scanned.

> The name-value pairs in the string can have the format `name=value` or `name="value"`.
>
> All backslashes (\) must be followed by a literal character. If string values are found with no unescaped = signs (no name=), it assumes the names 1, 2, 3, and so on, depending on the string position (zero doesn't count). For example, if **pblock_str2pblock** finds "some" "strings" "together", the function treats the strings as if they appeared in the name-value pairs as 1="some" 2 ="strings" 3="together".

**pblock ****pb* is the parameter block into which all name-value pairs are to be stored.

See also

pblock_copy, pblock_create, pblock_find, pblock_free, pblock_nvinsert, pblock_remove, pblock_pblock2str

PERM_FREE (declared in netsite.h)

The **PERM_FREE** macro is a platform-independent substitute for the C library routine **free**. It deallocates the persistent space previously allocated by PERM_MALLOC or PERM_STRDUP to a specfied pointer. If memory pooling has been disabled in the configuration file, PERM_FREE and FREE both return memory to the system heap.

Syntax

```
#include <netsite.h>
PERM_FREE(ptr);
```

Returns

```
void
```

Parameters

ptr is a (void) pointer to an object. If the pointer is not one created by PERM_MALLOC or PERM_STRDUP, the behavior is undefined.

See also

FREE, MALLOC, REALLOC, STRDUP, PERM_MALLOC, PERM_STRTUP

PERM_MALLOC (declared in netsite.h)

The **PERM_MALLOC** macro is a platform-independent substitute for the C library routine **malloc**. It provides allocation of memory that persists after the request that was being processed has been completed. If memory pooling has been disabled in the configuration file, PERM_MALLOC and MALLOC both obtain their memory from the system heap.

Syntax

```
#include <netsite.h>
PERM_MALLOC(size)
```

Returns

A pointer to space for an object of size *size*.

Parameters

size (an int) is the number of bytes to allocate.

Example

```
/* Initialize hosts array */
    num_hosts = 0;
    hosts = (char **) PERM_MALLOC(1 * sizeof(char *));
    hosts[0] = NULL;
```

See also

MALLOC, REALLOC, FREE, PERM_FREE, STRDUP, PERM_STRDUP

PERM_STRDUP (declared in netsite.h)

The **PERM_STRDUP** macro is a platform-independent substitute for the common UNIX library routine **strdup**. It creates a new copy of a string in memory that persists after the request that was being processed has been completed. If memory pooling has been disabled in the configuration file, PERM_STRDUP and STRDUP both obtain their memory from the system heap.

The **strdup** routine is functionally equivalent to

```
char *newstr = (char *) malloc(strlen(str) + 1);
strcpy(newstr, str);
```

Syntax

```
#include <netsite.h>
PERM_STRDUP(ptr);
```

Returns

A pointer to the new string.

Parameters

ptr is a pointer to a string.

See also

MALLOC, FREE, REALLOC

protocol_dump822 (declared in frame/protocol.h)

The **protocol_dump822** function prints headers from a specified parameter block into a specific buffer, with a specified size and position. Use this function to serialize the headers so that they can be sent, for example, in a mail message.

Syntax

```
#include <frame/protocol.h>
char *protocol_dump822(pblock *pb, char *t, int *pos, int tsz);
```

Returns

The buffer, reallocated if necessary

The function also modifies *pos* to denote a new position in the buffer.

Parameters

**pblock ***pb* is the parameter block structure.

**char ***t* is the name of the buffer.

**int ***pos* is the position within the buffer at which the headers are to be inserted.

**int ***tsz* is the size of the buffer.

See also

protocol_handle_session, protocol_scan_headers, protocol_start_response, protocol_status

protocol_finish_request (declared in frame/protocol.h)

The **protocol_finish_request** function finishes a specified request. For HTTP, the function just closes the socket.

Syntax

```
#include <frame/protocol.h>
void protocol_finish_request(Session *sn, Request *rq);
```

Returns

```
void
```

Parameters

Session * *sn* is the Session that generated the request.

Request * *rq* is the Request to be finished.

See also

protocol_handle_session, *protocol_scan_headers*, *protocol_start_response*, *protocol_status*

protocol_handle_session (declared in frame/protocol.h)

The **protocol_handle_session** function processes each request generated by a specified session.

Syntax

```
#include <frame/protocol.h>
void protocol_handle_session(Session *sn);
```

Parameters

Session * *sn* is the that generated the requests.

See also

protocol_scan_headers, *protocol_start_response*, *protocol_status*

protocol_hdrs2env (declared in frame/protocol.h)

The **protocol_hdrs2env** function converts the entries in a specified parameter block and converts them to an environment. Use this function to create an environment that a program can later use.

Syntax

```
#include <frame/protocol.h>
char **protocol_hdrs2env(pblock *pb);
```

Returns

A pointer to the new environment. Note that each entry is converted to uppercase text with the prefix HTTP_.

A hyphen (-) or double hyphen(--) in the text is automatically converted into an underscore (_), or double underscore (_ _), respectively.

Parameters

**pblock ****pb* is the parameter block.

See also

protocol_handle_session, protocol_scan_headers, protocol_start_response, protocol_status

protocol_parse_request (declared in frame/protocol.h)

Parses the first line of an HTTP request.

Syntax

```
#include <frame/protocol.h>
int protocol_parse_request(char *t, Request *rq, Session *sn);
```

Returns

- The constant `REQ_PROCEED` if the operation succeeded
- The constant `REQ_ABORTED` if the operation did not succeed

Parameters

**char ****t* defines a string of length REQ_MAX_LINE. This is an optimization for the internal code to reduce usage of runtime stack.

**Request ****rq* is the request to be parsed.

**Session ****sn* is the session that generated the request.

See also

protocol_scan_headers, protocol_start_response, protocol_status

protocol_scan_headers (declared in frame/protocol.h)

Scans HTTP headers from a specified network buffer, and places them in a specified parameter block.

Folded lines are joined and the linefeeds are removed (but not the whitespace). If there are any repeat headers, they are joined and the two field bodies are separated by a comma and space. For example, multiple mail headers are combined into one header and a comma is used to separate the field bodies.

Syntax

```
#include <frame/protocol.h>
int protocol_scan_headers(Session *sn, netbuf *buf, char *t,
pblock *headers);
```

Returns

- The constant REQ_PROCEED if the operation succeeded
- The constant REQ_ABORTED if the operation did not succeed

Parameters

**Session ****sn* is the session that generated the request. The structure named by *sn* contains a pointer to a netbuf called *inbuf*. If the parameter *buf* is NULL, the function automatically uses *inbuf*.

Note that *sn* is an optional parameter that is used for error logs. Use NULL if you wish.

**netbuf ****buf* is the network buffer to be scanned for HTTP headers.

**char ****t* defines a string of length REQ_MAX_LINE. This is an optimization for the internal code to reduce usage of runtime stack.

**pblock ****headers* is the parameter block to receive the headers.

See also

protocol_handle_session, protocol_start_response, protocol_status

protocol_set_finfo (declared in frame/protocol.h)

The **protocol_set_finfo** function retrieves the content-length and last-modified date from a specified **stat** structure, for a specified Session and the Request generated by that Session. Use **protocol_set_finfo** only after receiving a **start_response** from a service-class server application function (SAF).

Syntax

```
#include <frame/protocol.h>
int protocol_set_finfo(Session *sn, Request *rq, struct stat
*finfo);
```

Returns

- The constant REQ_PROCEED if the Request can proceed normally
- The constant REQ_ABORTED if the function should treat the Request normally but not send any output to the client

Parameters

Session * *sn* is the Session that generated the Request.

Request * *rq* is the Request.

stat * *finfo* is the stat structure for the file.

> The **stat** structure contains the information about the file you are sending back to the client. The full description of the **stat** structure should be available from the documentation for your system. For the basic elements of the **stat** structure, see "The Stat Data Structure," on page 397.

See also

protocol_handle_session, protocol_scan_headers, protocol_start_response, protocol_status

protocol_start_response (declared in frame/protocol.h)

The **protocol_start_response** function initiates the HTTP response for a specified Session and Request. If the protocol version is HTTP/0.9, the function does nothing, because that version has no concept of status. If the protocol version is HTTP/1.0, the function sends a status line followed by the response headers. Use this function to set up HTTP and prepare the client and server to receive the body (or data) of the response.

Syntax

```
#include <frame/protocol.h>
int protocol_start_response(Session *sn, Request *rq);
```

Returns

- The constant REQ_PROCEED if the operation succeeded, in which case you can send the data you were preparing to send
- The constant REQ_NOACTION if the operation succeeded, but the client has requested that the server not send the data because the client has it in cache
- The constant REQ_ABORTED if the operation did not succeed

Parameters

**Session ***sn* is the Session that generated the Request.

**Request ***rq* is the Request to which a response is being started.

Example

```
/* A noaction response from this function means the request was HEAD
*/
if(protocol_start_response(sn, rq) == REQ_NOACTION) {
   filebuf_close(groupbuf);  /* this also closes fd */
   return REQ_PROCEED;
}
```

See also

protocol_handle_session, *protocol_scan_headers*, *protocol_status*

protocol_status (declared in frame/protocol.h)

The **protocol_status** function sets the Session status to indicate whether an error condition occurred. If the reason string is NULL, the server attempts to find a reason string for the given status code. If it finds none, it returns "Unknown reason." This reason string is sent to the client in the status line. Use this function to set the status of the Session before calling the function **protocol_start_response**.

These are valid status codes:

```
PROTOCOL_OK
PROTOCOL_NO_RESPONSE
PROTOCOL_REDIRECT
PROTOCOL_NOT_MODIFIED
PROTOCOL_BAD_REQUEST
PROTOCOL_UNAUTHORIZED
PROTOCOL_FORBIDDEN
PROTOCOL_NOT_FOUND
PROTOCOL_PROXY_UNAUTHORIZED
PROTOCOL_SERVER_ERROR
PROTOCOL_NOT_IMPLEMENTED
```

Syntax

```
#include <frame/protocol.h>
void protocol_status(Session *sn, Request *rq, int n, char *r);
```

Returns

`void`, but sets values in the Session / Request designated by *sn* / *rq* for the status code and the reason string

Parameters

**Session ****sn* is the Session that generated the Request.

**Request ****rq* is the Request that is being checked on.

int *n* is the value to which to set the status code.

**char ****r* is the reason string.

Example

```
if( (t = pblock_findval("path-info", rq->vars)) ) {
    protocol_status(sn, rq, PROTOCOL_NOT_FOUND, NULL);
    log_error(LOG_WARN, "send-images", sn, rq,
                "%s%s not found", path, t);
    return REQ_ABORTED;
}
```

See also

protocol_handle_session, protocol_scan_headers, protocol_start_response

protocol_uri2url (declared in frame/protocol.h)

The **protocol_uri2url** function takes strings containing the given URI prefix and URI suffix and creates a newly allocated fully qualified URL in the form http://(server):(port)(*prefix*)(*suffix*).

If you want to omit either the URI prefix or suffix, use "" instead of NULL as the value for either parameter. To redirect the client somewhere else, use the function **pblock_nvinsert** to create a new entry in the vars in the pblock in your Request structure.

Syntax

```
#include <frame/protocol.h>
char *protocol_uri2url(char *prefix, char *suffix);
```

Returns

A new string containing the URL

Parameters

**char ****prefix* is the prefix.

**char ****suffix* is the suffix.

See also

protocol_handle_session, protocol_scan_headers, protocol_start_response, protocol_status

protocol_uri2url_dynamic (declared in frame/protocol.h)

The **protocol_uri2url** function takes strings containing the given URI prefix and URI suffix and creates a newly allocated fully qualified URL in the form http://(server):(port)(*prefix*)(*suffix*).

If you want to omit either the URI prefix or suffix, use "" instead of NULL as the value for either parameter. To redirect the client somewhere else, use the function **pblock_nvinsert** to create a new entry in the vars in the pblock in your Request structure.

The **protocol_uri2url_dynamic** function is exactly like the **protocol_uri2url** function, but should be used whenever the **Session** and **Request** structures are available. This ensures that the URL that it constructs refers to the host that the client specified.

Syntax

```
#include <frame/protocol.h>
char *protocol_uri2url(char *prefix, char *suffix, Session *sn,
Request *rq);
```

Returns

A new string containing the URL

Parameters

**char ****prefix* is the prefix.

**char ****suffix* is the suffix.

**Session ****sn* is the Session that generated the Request.

**Request ****rq* is the Request that is being processed.

See also

protocol_handle_session, protocol_scan_headers, protocol_start_response, protocol_status

REALLOC (declared in netsite.h)

The **REALLOC** macro is a platform-independent substitute for the C library routine **realloc**. It changes the size of a specfied object that was originally created by MALLOC or STRDUP. The contents of the object remain unchanged up to the minimum of the old and new sizes. If the new size is larger, the new space is uninitialized.

NOTE Calling REALLOC for a block that was allocated with PERM_MALLOC will not work.

Syntax

```
#include <netsite.h>
REALLOC(ptr, size);
```

Returns

A pointer to the new space if the Request could be satisfied.

Parameters

ptr is a (void) pointer to an object. If the pointer is not one created by MALLOC or STRDUP, the behavior is undefined.

size (an int) is the number of bytes to allocate.

Example

```
while(fgets(buf, MAX_ACF_LINE, f)) {
/* Blast linefeed that stdio leaves on there */
   uf[strlen(buf) - 1] = '\0';
   hosts = (char **) REALLOC(hosts, (num_hosts + 2) * sizeof(char
*));
   hosts[num_hosts++] = STRDUP(buf);
   hosts[num_hosts] = NULL;
}
```

See also

MALLOC, FREE, STRDUP

request_create (declared in frame/req.h)

The **request_create** function is a utility function that creates a new request structure.

Syntax

```
#include <frame/req.h>
Request *request_create(void);
```

Returns

A **Request** structure

Parameters

No parameter is required.

See also

request_free, request_header

request_free (declared in frame/req.h)

The **request_free** function frees a specified request structure.

Syntax

```
#include <frame/req.h>
void request_free(Request *req);
```

Returns

```
void
```

Parameters

**Request ****rq* is the Request structure to be freed.

See also

request_header

request_header (declared in frame/req.h)

The **request_header** function finds the parameter block containing the client's HTTP headers. You can use this function to access a parameter block indirectly, thereby avoiding multiple and unnecessary calls. In addition, **request_header** allows you to access the parameter block headers in your copy of the request structure.

Syntax

```
#include <frame/req.h>
int request_header(char *name, char **value, Session *sn, Request *rq);
```

Returns

A REQ return code, such as `REQ_ABORTED` to signal that an error occurred, or `REQ_PROCEED` to signal that all went well

Parameters

**char ***_name_ is the name of the header.

**char ****_value_ is the address where the function will place the value of the specified header. If none is found, the function stores a NULL.

**Session ***_sn_ is the Session identifier for the server application function call that generated the Request.

**Request ***_rq_ is the Request identifier for a server application function call.

The *sn* and *rq* parameters can also be used to identify a specific Request in asynchronous operations as well as for other internal housekeeping purposes.

Example

See **shexp_cmp**.

See also

request_create, *request_free*

request_stat_path (declared in frame/req.h)

The **request_stat_path** function returns the file information structure for a specified path, if none is specified, the path entry in the vars pblock in a specified Request structure. If the resulting filename points to a file that the server can read, **request_stat_path** returns a new file information structure. This structure contains information on the size of the file, when it was last accessed, and when it was last changed.

You can use **request_stat_path** to retrieve information on the file you are currently accessing (instead of calling **stat** directly), because this function keeps track of other calls.

Syntax

```
#include <frame/req.h>
struct stat *request_stat_path(char *path, Request *rq);
```

Returns

- NULL if the file is not valid or the server cannot read it. In this case, it also leaves an error message describing the problem in the Request structure denoted by *rq*.
- The file information structure for the file named by the *path* parameter.

If you receive a valid file information structure, you should not free this structure.

Parameters

**char ****path* is the string containing the name of the path. If the value of *path* is NULL, the function uses the path entry in the vars pblock in the Request structure denoted by *rq*.

**Request ****rq* is the Request identifier for a server application function call.

Example

```
if( (!(fi = request_stat_path(path, rq)) ) ||
( (fd = system_fopenRO(path)) == IO_ERROR) )
```

See also

request_create, request_free, request_header

request_translate_uri (declared in frame/req.h)

The **request_translate_uri** function performs virtual-to-physical mapping on a specified URI during a specified Session. Use this function when you want to determine which file will be sent back if a given URI is accessed.

Syntax

```
#include <frame/req.h>
char *request_translate_uri(char *uri, Session *sn);
```

Returns

- A path string, if it performed the mapping
- NULL if it could not perform the mapping

Parameters

**char ***uri* is the name of the URI.

**Session ***sn* is the Session identifier for the server application function call.

See also

request_create, request_free, request_header

sem_grab (declared in base/sem.h)

The **sem_grab** function requests exclusive access to a specified semaphore. If exclusive access is unavailable, the caller blocks execution until exclusive access becomes available. Use this function to ensure that only one server processor thread performs an action at a time.

Syntax

```
#include <base/sem.h>
int sem_grab(SEMAPHORE id);
```

Returns

- -1 if an error occurred
- 0 to signal success

Parameters

SEMAPHORE *id* is the unique identification number of the requested semaphore.

See also

sem_init, sem_release, sem_terminate, sem_tgrab

sem_init (declared in base/sem.h)

The **sem_init** function creates a semaphore with a specified name and unique identification number. Use this function to allocate a new semaphore that will be used with the functions **sem_grab** and **sem_release**. Call **sem_init** from an **init** class function to initialize a static or global variable that the other classes will later use.

Syntax

```
#include <base/sem.h>
SEMAPHORE sem_init(char *name, int number);
```

Returns

The constant SEM_ERROR if an error occurred.

Parameters

**SEMAPHORE ****name* is the name for the requested semaphore. The filename of the semaphore should be a file accessible to the process.

int *number* is the unique identification number for the requested semaphore.

See also

sem_grab, sem_release, sem_terminate

sem_release (declared in base/sem.h)

The **sem_release** function releases the process's exclusive control over a specified semaphore. Use this function to release exclusive control over a semaphore created with the function **sem_grab**.

Syntax

```
#include <base/sem.h>
int sem_release(SEMAPHORE id);
```

Returns

- -1 if an error occurred
- 0 of no error occurred

Parameters

SEMAPHORE *id* is the unique identification number of the semaphore.

See also

sem_grab, sem_init, sem_terminate

sem_terminate (declared in base/sem.h)

The **sem_terminate** function deallocates the semaphore specified by *id*. You can use this function to deallocate a semaphore that was previously allocated with the function **sem_init**.

Syntax

```
#include <base/sem.h>
void sem_terminate(SEMAPHORE id);
```

Returns

```
void
```

Parameters

SEMAPHORE *id* is the unique identification number of the semaphore.

See also

sem_grab, sem_init, sem_release

sem_tgrab (declared in base/sem.h)

The **sem_tgrab** function tests and requests exclusive use of a semaphore. Unlike the somewhat similar **sem_grab** function, if exclusive access is unavailable the caller is not blocked but receives a return value of -1. Use this function to ensure that only one server processor thread performs an action at a time.

Syntax

```
#include <base/sem.h>
int sem_grab(SEMAPHORE id);
```

Returns

- -1 if an error occurred or if exclusive access was not available
- 0 exclusive access was granted

Parameters

SEMAPHORE *id* is the unique identification number of the semaphore.

See also

sem_grab, sem_init, sem_release, sem_terminate

session_create (declared in base/session.h)

The **session_create** function creates a new Session structure for the client with a specified socket descriptor and a specified socket address. It returns a pointer to that structure.

Syntax

```
#include <base/session.h>
Session *session_create(SYS_NETFD csd, struct sockaddr_in *sac);
```

Returns

- A pointer to the new Session if one was created
- NULL if no new Session was created

Parameters

SYS_NETFD*csd* is the platform-independent socket descriptor.

**sockaddr_in ****sac* is the socket address.

See also

session_maxdns

session_free (declared in base/session.h)

The **session_free** function frees a specified Session structure. The **session_free** function does not close the client socket descriptor associated with the Session.

Syntax

```
#include <base/session.h>
void session_free(Session *sn);
```

Returns

`void`

Parameters

**Session ****sn* is the Session to be freed.

See also

session_create, session_maxdns

session_maxdns (declared in base/session.h)

The **session_maxdns** function resolves the IP address of the client associated with a specified Session into a host name. It returns a string. You can use **session_maxdns** to change the numeric IP address into something more readable. Use **session_maxdns** instead of the function **session_dns** if you want to be sure that the host name is associated with the IP address of the client.

This function is implemented as a macro.

Syntax

```
#include <base/session.h>
char *session_maxdns(Session *sn);
```

Returns

- A string containing the host name
- NULL if no host name was associated with the IP address

Parameters

Session * *sn* is the Session identifier for the server application function call.

shexp_casecmp (declared in base/shexp.h)

The **shexp_casecmp** function validates a specified shell expression and compares it with a specified string. It returns one of three possible values representing match, no match, and invalid comparison. In contrast with the **shexp_cmp** function, the comparison is not case-sensitive.

Use this function if you have a shell expression like *.netscape.com and you want to make sure that a string matches it, such as foo.netscape.com.

Syntax

```
#include <base/shexp.h>
int shexp_casecmp(char *str, char *exp);
```

Returns

- 0 if a match was found
- 1 if no match was found
- -1 if the comparison resulted in an invalid expression

Parameters

**char ***_str_ is the string to be compared.

**char ***_exp_ is the shell expression (possibly containing wildcard characters) against which to compare.

See also

shexp_cmp, shexp_match

shexp_cmp (declared in base/shexp.h)

The **shexp_cmp** function validates a specified shell expression and compares it with a specified string. It returns one of three possible values representing match, no match, and invalid comparison. In contrast with the **shexp_casecmp** function, the comparison is case-sensitive.

Use this function if you have a shell expression like *.netscape.com and you want to make sure that a string matches it, such as foo.netscape.com.

Syntax

```
#include <base/shexp.h>
int shexp_cmp(char *str, char *exp);
```

Returns

- 0 if a match was found
- 1 if no match was found
- -1 if the comparison resulted in an invalid expression

Parameters

**char ***_str_ is the string to be compared.

**char ***_exp_ is the shell expression (possibly containing wildcard characters) against which to compare.

Example

```
#include "base/util.h"        /* is_mozilla */
#include "frame/protocol.h"   /* protocol_status */
#include "base/shexp.h"       /* shexp_cmp */
int https_redirect(pblock *pb, Session *sn, Request *rq)
{
   /* Server Variable */
   char *ppath = pblock_findval("ppath", rq->vars);
```

```
    /* Parameters */
    char *from = pblock_findval("from", pb);
    char *url = pblock_findval("url", pb);
    char *alt = pblock_findval("alt", pb);
    /*
    /* Check usage */
    if((!from) || (!url)) {
    log_error(LOG_MISCONFIG, "https-redirect", sn, rq,
        "missing parameter (need from, url)");
        return REQ_ABORTED;
    }
    /* Use wildcard match to see if this path is one to redirect */
    if(shexp_cmp(ppath, from) != 0)
        return REQ_NOACTION;   /* no match */
        /* The only way to check for SSL capability is to check UA */
    if(request_header("user-agent", &ua, sn, rq) == REQ_ABORTED)
    return REQ_ABORTED;
    /* The is_mozilla fn checks for Mozilla version 0.96 or greater
*/
    f(util_is_mozilla(ua, "0", "96")) {
        /* Set the return code to 302 Redirect */
        protocol_status(sn, rq, PROTOCOL_REDIRECT, NULL);
            /* The error handling fns use this to set Location: */
        pblock_nvinsert("url", url, rq->vars);
        return REQ_ABORTED;
    }
    /* No match. Old client. */
    /* If there is an alternate document specified, use it. */
    if(alt) {
        pb_param *pp = pblock_find("ppath", rq->vars);
        /* Trash the old value */
        FREE(pp->value);
        /* Dup it because the library will later free this pblock */
        pp->value = STRDUP(alt);
        return REQ_PROCEED;
    }
    /* Else do nothing */
    return REQ_NOACTION;
}
```

See also

shexp_casecmp, shexp_match

shexp_match (declared in base/shexp.h)

The **shexp_match** function compares a specified prevalidated shell expression against a specified string. It returns one of three possible values representing match, no match, and invalid comparison. In contrast with the **shexp_casecmp** function, the comparison is case-sensitive.

The **shexp_match** function doesn't perform validation of the shell expression; instead the function assumes that you have already called **shexp_valid**.

Use this function if you have a shell expression like *.netscape.com and you want to make sure that a string matches it, such as foo.netscape.com.

Syntax

```
#include <base/shexp.h>
int shexp_match(char *str, char *exp);
```

Returns

- 0 if a match was found
- 1 if no match was found
- -1 if the comparison resulted in an invalid expression

Parameters

char * *str* is the string to be compared.

char * *exp* is the prevalidated shell expression (possibly containing wildcard characters) against which to compare.

See also

shexp_casecmp, shexp_cmp

shexp_valid (declared in base/shexp.h)

The **shexp_valid** function validates a specified shell expression named by *exp*. Use this function to validate a shell expression before using the function **shexp_match** to compare the expression with a string.

Syntax

```
#include <base/shexp.h>
int shexp_valid(char *exp);
```

Returns

- The constant NON_SXP if *exp* is a standard string
- The constant INVALID_SXP if *exp* is a shell expression but invalid
- The constant VALID_SXP if *exp* is a valid shell expression

Parameters

**char ***exp* is the prevalidated shell expression (possibly containing wildcard characters) to be used later in a shexp_match comparison.

See also

shexp_casecmp, shexp_match, shexp_cmp

shmem_alloc (declared in base/shmem.h)

The **shmem_alloc** function allocates a region of shared memory of the given size, using the given name to avoid conflicts between multiple regions in the program. The size of the region will not be automatically increased if its boundaries are overrun; use the **shmem_realloc** function for that.

This function must be called before any daemon workers are spawned in order for the handle to the shared region to be inherited by the children.

Because of the requirement that the region must be inherited by the children, the region cannot be reallocated with a larger size when necessary.

Syntax

```
#include <base/shmem.h>
shmem_s *shmem_alloc(char *name, int size, int expose);
```

Returns

A pointer to a new shared memory region.

Parameters

char **name* is the name for the region of shared memory being created. The value of *name* must be unique to the program that calls the **shmem_alloc** function or conflicts will occur.

int *size* is the number of characters of memory to be allocated for the shared memory.

int *expose* is either zero or nonzero. If nonzero, then on systems that support it, the file that is used to create the shared memory becomes visible to other processes running on the system.

See also

shmem_free

shmem_free (declared in base/shmem.h)

The **shmem_free** function deallocates (frees) the specified region of memory.

Syntax

```
#include <base/shmem.h>
void *shmem_free(shmem_s *region);
```

Returns

```
void
```

Parameters

**shmem_s ****region* is a shared memory region to be released.

See also

shmem_allocate

STRDUP (declared in netsite.h)

The **STRDUP** macro is a platform-independent substitute for the common UNIX library routine **strdup**. It creates a new copy of a string.

The **strdup** routine is functionally equivalent to this:

```
char *newstr = (char *) MALLOC(strlen(str) + 1);
strcpy(newstr, str);
```

Syntax

```
#include <netsite.h>
STRDUP(ptr);
```

Returns

A pointer to the new string.

Parameters

ptr is a pointer to a string.

Example

```
while(fgets(buf, MAX_ACF_LINE, f)) {
/* Blast linefeed that stdio leaves on there */
   uf[strlen(buf) - 1] = '\0';
   hosts = (char **) REALLOC(hosts, (num_hosts + 2) * sizeof(char
*));
   hosts[num_hosts++] = STRDUP(buf);
   hosts[num_hosts] = NULL;
}
```

See also

MALLOC, FREE, REALLOC

systhread_attach (declared in base/systhr.h)

The **systhread_attach** function makes an existing thread a platform-independent thread.

Syntax

```
#include <base/systhr.h>
SYS_THREAD systhread_attach(void);
```

Returns

A **SYS_THREAD** pointer to the platform-independent thread.

Parameters

void

See also

systhread_current, systhread_getdata, systhread_init, systhread_newkey, systhread_setdata, systhread_sleep,systhread_start, systhread_terminate, systhread_timerset

systhread_current (declared in base/systhr.h)

The **systhread_current** function returns a pointer to the current thread.

Syntax

```
#include <base/systhr.h>
SYS_THREAD systhread_current(void);
```

Returns

A **SYS_THREAD** pointer to the current thread

Parameters

void

See also

systhread_getdata, systhread_newkey, systhread_setdata, systhread_sleep,systhread_start, systhread_terminate, systhread_ timerset

systhread_getdata (declared in base/systhr.h)

The **systhread_getdata** function gets data that is associated with a specified key in the current thread

Syntax

```
#include <base/systhr.h>
void *systhread_getdata(int key);
```

Returns

- A pointer to the data that was earlier used with the **systhread_setkey** function from the current thread, using the same value of *key*.
- NULL if the call did not succeed, for example if the **systhread_setkey** function was never called with the specified key during this session.

Parameters

int *key* is the value associated with the stored data by a **systhread_setdata** function. Keys are assigned by the **systhread_newkey** function.

See also

systhread_current, systhread_newkey, systhread_setdata, systhread_sleep,systhread_start, systhread_terminate, systhread_ timerset

systhread_init (declared in base/systhr.h)

The **systhread_init** function initializes the threading system.

Syntax

```
#include <base/systhr.h>
void systhread_init(char *name);
```

Returns

void

Parameters

**char ***name* is a name to be assigned to the program for debugging purposes.

See also

systhread_attach, systhread_current, systhread_getdata, systhread_newkey, systhread_setdata, systhread_sleep,systhread_start, systhread_terminate, systhread_ timerset

systhread_newkey (declared in base/systhr.h)

The **systhread_newkey** function allocates a new integer key (identifier) for thread-private data. Use this key to identify a variable that you want to localize to the current thread, then use the **systhread_setdata** function to associate a value with the key.

Syntax

```
#include <base/systhr.h>
int systhread_newkey(void);
```

Returns

An integer key.

Parameters

`void`

See also

systhread_current, systhread_getdata, systhread_setdata, systhread_sleep, systhread_start, systhread_terminate, systhread_ timerset

systhread_setdata (declared in base/systhr.h)

The **systhread_setdata** function associates data with a specified key number for the current thread. Keys are assigned by the **systhread_newkey** function.

Syntax

```
#include <base/systhr.h>
void systhread_start(int key, void *data);
```

Returns

```
void
```

Parameters

int *key* is the priority of the thread.

void ***data*** is the pointer to the string of data to be associated with the value of *key*.

See also

systhread_current, systhread_getdata, systhread_newkey, systhread_sleep, systhread_start, systhread_terminate, systhread_ timerset

systhread_sleep (declared in base/systhr.h)

The **systhread_sleep** function puts the calling thread to sleep for a given time.

Syntax

```
#include <base/systhr.h>
void systhread_sleep(int milliseconds);
```

Returns

```
void
```

Parameters

int *milliseconds* is the number of milliseconds the thread is to sleep.

See also

systhread_current, systhread_getdata, systhread_newkey, systhread_setdata, systhread_start, systhread_terminate, systhread_ timerset

systhread_start (declared in base/systhr.h)

The **systhread_start** function creates a thread with the given priority, allocates a stack of a specified number of bytes, and calls a specified function with a specified argument.

Syntax

```
#include <base/systhr.h>
SYS_THREAD systhread_start(int prio, int stksz, void (*fn)(void *),
void *arg);
```

Returns

- A new **SYS_THREAD** pointer if the call succeeded
- The constant SYS_THREAD_ERROR if the call did not succeed.

Parameters

int *prio* is the priority of the thread. Priorities are system-dependent.

int *stksz* is the stack size in bytes. If *stksz* is zero, the function allocates a default size.

void (**fn*)(void *) is the function to call.

void **arg* is the argument for the *fn* function.

See also

systhread_current, systhread_getdata, systhread_newkey, systhread_setdata, systhread_sleep, systhread_terminate, systhread_ timerset

systhread_terminate (declared in base/systhr.h)

The **systhread_terminate** function terminates a specified thread.

Syntax

```
#include <base/systhr.h>
void systhread_terminate(SYS_THREAD thr);
```

Returns

void

Parameters

SYS_THREAD *thr* is the thread to terminate.

See also

systhread_current, systhread_getdata, systhread_newkey, systhread_setdata, systhread_sleep, systhread_start, systhread_ timerset

systhread_timerset (declared in base/systhr.h)

The **systhread_timerset** function starts or resets the interrupt timer interval for a thread system.

NOTE Because most systems don't allow the timer interval to be changed, this should be considered a suggestion, rather than a command.

Syntax

```
#include <base/systhr.h>
void systhread_timerset(int usec);
```

Returns

void

Parameters

int *usec* is the time, in microseconds

See also

systhread_current, systhread_getdata, systhread_newkey, systhread_setdata, systhread_sleep,systhread_start, systhread_terminate

system_errmsg (declared in base/file.h)

The **system_errmsg** function returns the last error that occurred from the most recent system call. This function is implemented as a macro that returns an entry from the global array **sys_errlist**. Use this macro to help with I/O error diagnostics.

Syntax

```
#include <base/file.h>
char *system_errmsg(int para1);
```

Returns

A string containing the text of the latest error message that resulted from a system call.

Parameters

int *para1* is reserved and should always have the value zero.

See also

system_fclose, system_fread, system_fopenRO, system_fwrite

system_fclose (declared in base/file.h)

The **system_fclose** function closes a specified file descriptor. The **system_fclose** function must be called for every file descriptor opened by any of the **system_fopen** functions.

Syntax

```
#include <base/file.h>
int system_fclose(SYS_FILE fd);
```

Returns

- 0 if the close succeeded
- The constant IO_ERROR if the close failed

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

Example

```
static SYS_FILE logfd = SYS_ERROR_FD;
// this function closes global logfile
void brief_terminate()
{
   system_fclose(logfd);
   logfd = SYS_ERROR_FD;
}
```

See also

system_errmsg, system_fread, system_fopenRO, system_fwrite

system_flock (declared in base/file.h)

The **system_flock** function locks the specified file against interference from other processes. Use **system_flock** if you do not want other processes using the file you currently have open. Overusing file locking can cause performance degradation and possibly lead to deadlocks.

Syntax

```
#include <base/file.h>
int system_flock(SYS_FILE fd);
```

Returns

- The constant IO_OK if the lock succeeded
- The constant IO_ERROR if the lock failed

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

See also

system_fclose, system_fread, system_fopenRO, system_fwrite, system_ulock

system_fopenRO (declared in base/file.h)

The **system_fopenRO** function opens the file identified by *path* in read-only mode and returns a valid file descriptor. Use this function to open files that will not be modified by your program. In addition, you can use **system_fopenRO** to open a new file buffer structure using **filebuf_open**.

Syntax

```
#include <base/file.h>
SYS_FILE system_fopenRO(char *path);
```

Returns

- The system-independent file descriptor (**SYS_FILE)** if the open succeeded
- 0 if the open failed

Parameters

**char ***path* is the filename.

See also

system_fclose, system_fread, system_fopenRW, system_fopenWA, system_fwrite, system_ulock

system_fopenRW (declared in base/file.h)

The **system_fopenRW** function opens the file identified by *path* in read-write mode and returns a valid file descriptor. If the file already exists, **system_fopenRW** does not truncate it. Use this function to open files that will be read from and written to by your program.

Syntax

```
#include <base/file.h>
SYS_FILE system_fopenRW(char *path);
```

Returns

- The system-independent file descriptor (**SYS_FILE**) if the open succeeded
- 0 if the open failed

Parameters

**char ****path* is the filename.

Example

```
/* If any errors, just skip it. */
if(stat(pathname, &finfo) == -1)
   break;

fd = system_fopenRO(pathname);
if(fd == SYS_ERROR_FD)
   break;
```

See also

system_fclose, system_fread, system_fopenRO, system_fopenWA, system_fwrite, system_ulock

system_fopenWA (declared in base/file.h)

The **system_fopenWA** function opens the file identified by *path* in append mode and returns a valid file descriptor. Use this function to open those files to which your program will append data.

Syntax

```
#include <base/file.h>
SYS_FILE system_fopenWA(char *path);
```

Returns

- The system-independent file descriptor (**SYS_FILE)** if the open succeeded
- 0 if the open failed

Parameters

**char ***_path_ is the filename.

See also

system_fclose, system_fread, system_fopenRO, system_fopenRW, system_fwrite, system_ulock

system_fread (declared in base/file.h)

The **system_fread** function reads a specified number of bytes from a specified file into a specified buffer. It returns the number of bytes read. Before **system_fread** can be used, you must open the file using any of the **system_fopen** functions, except **system_fopenWA**.

Syntax

```
#include <base/file.h>
int system_fread(SYS_FILE fd, char *buf, int sz);
```

Returns

The number of bytes read, which may be less than the requested size if an error occurred or the end of the file was reached before that number of characters was obtained.

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

**char ***_buf_ is the buffer to receive the bytes.

int *sz* is the number of bytes to read.

See also

system_fclose, system_fopenRO, system_fopenRW, system_fopenWA, system_fwrite, system_ulock

system_fwrite (declared in base/file.h)

The **system_fwrite** function writes a specified number of bytes from a specified buffer into a specified file. Before **system_fwrite** can be used, you must open the file using any of the **system_fopen** functions, except **system_fopenRO**.

Syntax

```
#include <base/file.h>
int system_fwrite(SYS_FILE fd, char *buf, int sz);
```

Returns

- The constant IO_OK if the write succeeded
- The constant IO_ERROR if the write failed

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

**char ****buf* is the buffer containing the bytes to be written.

int *sz* is the number of bytes to write to the file.

See also

system_fclose, system_fopenRO, system_fopenRW, system_fopenWA, system_fwrite_atomic, system_ulock

system_fwrite_atomic (declared in base/file.h)

The **system_fwrite_atomic** function writes a specified number of bytes from a specified buffer into a specified file. The function also locks the file prior to performing the write and then unlocks it when done, thereby avoiding interference between simultaneous write actions. Before **system_fwrite_atomic** can be used, you must open the file using any of the **system_fopen** functions.

Syntax

```
#include <base/file.h>
int system_fwrite_atomic(SYS_FILE fd, char *buf, int sz);
```

Returns

- The constant IO_OK if the write/lock succeeded
- The constant IO_ERROR if the write/lock failed

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

char ***buf*** is the buffer containing the bytes to be written.

int *sz* is the number of bytes to write to the file.

Example

```
logmsg = (char *)
   MALLOC(strlen(ip) + 1 + strlen(method) + 1 + strlen(uri) + 1 +
1);
len = util_sprintf(logmsg, "%s %s %s\n", ip, method, uri);
/* The atomic version uses locking to prevent interference */
system_fwrite_atomic(logfd, logmsg, len);
FREE(logmsg);
```

See also

system_fclose, system_fopenRO, system_fopenRW, system_fopenWA, system_fwrite, system_ulock

system_gmtime (declared in base/file.h)

The **system_gmtime** function is a thread-safe version of the standard **gmltime** function.

Syntax

```
#include <base/file.h>
struct tm *system_gmtime(const time_t *tp, const struct tm *res);
```

Returns

a pointer to a calendar time (**tm**) structure containing the GMT time. Depending on your system, the pointer may point to the data item represented by the second parameter, or it may point to a statically allocated item. For portability, do not assume either situation.

Parameters

time_t ***tp*** is an arithmetic time.

tm ***res*** is a pointer to a calendar time (**tm**) structure.

Example

```
time_t tp;
struct tm res, *resp;
...
tp = time(NULL);
resp = system_gmtime(&tp, &res);
```

See also

system_localtime

system_localtime (declared in base/file.h)

The **system_localtime** function is a thread-safe version of the standard **localtime** function.

Note that this function is implemented as a macro.

Syntax

```
#include <base/file.h>
struct tm *system_localtime(const time_t *tp, const struct tm *res);
```

Returns

a pointer to a calendar time (**tm**) structure containing the local time. Depending on your system, the pointer may point to the data item represented by the second parameter, or it may point to a statically allocated item. For portability, do not assume either situation.

Parameters

**time_t ***tp* is an arithmetic time.

tm ***res* is a pointer to a calendar time (tm**) structure.

Example

```
time_t tp;
struct tm res, *resp;
...
tp = time(NULL);
resp = system_localtime(&tp, &res);
```

See also

system_gmtime

system_ulock (declared in base/file.h)

The **system_ulock** function unlocks the specified file that has been locked by the function **system_lock**. For more information about locking, see **system_flock**.

Syntax

```
#include <base/file.h>
int system_ulock(SYS_FILE fd);
```

Returns

- The constant IO_OK if the unlock succeeded
- The constant IO_ERROR if the unlock failed

Parameters

SYS_FILE *fd* is the platform-independent file descriptor.

See also

system_fclose, system_flock, system_fopenRO, system_fopenRW, system_fopenWA, system_fwrite

system_unix2local (declared in base/file.h)

The **system_unix2local** function converts a specified UNIX-style pathname to a local pathname named by *lp*. Use this function when you have a filename in the UNIX format (such as one containing forward slashes) and you are running Windows NT. You can use **system_unix2local** to convert the UNIX filename into the format that Windows NT accepts. In the UNIX environment, this function does nothing.

Syntax

```
#include <base/file.h>
char *system_unix2local(char *path, char *lp);
```

Returns

A pointer to the local path string

Parameters

char ***path** is the UNIX-style pathname to be converted.

char ***lp** is the local pathname.

You must allocate the parameter *lp*, and it must contain enough space to hold the local pathname.

See also

system_fclose, system_flock, system_fopenRO, system_fopenRW, system_fopenWA, system_fwrite

util_can_exec (declared in base/util.h)

The **util_can_exec** function checks that a specified file can be executed, returning either a 1 (executable) or a 0. The function checks to see if the file can be executed by the user with the given user and group ID.

The **util_can_exec** function is available only under UNIX.

Use this function before executing a program using the **exec** system call.

Syntax

```
#include <base/util.h>
int util_can_exec(struct stat *finfo, uid_t uid, gid_t gid);
```

Returns

- 1 if the file is executable
- 0 if the file is not executable

Parameters

**stat ***finfo* is the stat structure associated with a file.

uid_t *uid* is the UNIX user ID.

gid_t *gid* is the UNIX group ID. Together with *uid*, this determines the permissions of the UNIX user.

See also

util_env_create, util_getline, util_host name

util_chdir2path (declared in base/util.h)

The **util_chdir2path** function changes the current directory to a specified directory, which should point to a file.

When running under Windows NT, use a semaphore to ensure that more than one thread does not call this function at the same time.

Use **util_chdir2path** when you want to make file access a little quicker, because you do not need to use a full path with this function.

Syntax

```
#include <base/util.h>
int util_chdir2path(char *path);
```

Returns

- 0 if the directory was changed
- **-**1 if the directory could not be changed

Parameters

**char ****path* is the name of a directory.

The parameter must be a writable string because it isn't permanently modified.

See also

util_env_create, util_getline, util_host name

util_does_process_exist (declared in libproxy/util.h)

The **util_does_process_exist** function verifies that a given process ID is that of an executing process.

Syntax

```
#include <libproxy/til.h>
int util_does_process_exist (int pid)
```

Returns

- nonzero if the *pid* represents an executing process
- 0 if the *pid* does not represent an executing process

Parameters

int *pid* is the process ID to be tested.

See also

util_url_fix_host name, util_uri_check

util_env_create (declared in base/util.h)

The **util_env_create** function creates and allocates the environment specified by *env*, returning a pointer to the environment. If the parameter *env* is NULL, the function allocates a new environment. Use **util_env_create** to create an environment when executing a new program.

Syntax

```
#include <base/util.h>
char **util_env_create(char **env, int n, int *pos);
```

Returns

A pointer to an environment.

Parameters

**char ****_env_ is the existing environment or NULL.

int *n* is the maximum number of environment entries that you want in the environment.

**int ***_pos_ is an integer that keeps track of the number of entries used in the environment.

See also

util_env_replace, util_env_str, util_env_free, util_env_find

util_env_find (declared in base/util.h)

The **util_env_find** function locates the string denoted by a name in a specified enviroment and returns the associated value. Use this function to find an entry in an environment.

Syntax

```
#include <base/util.h>
char *util_env_find(char **env, char *name);
```

Returns

- The value of the string, if one was found
- NULL if the string was not found

Parameters

**char ****_env_ is the environment.

**char ***_name_ is the name of a name-value pair.

See also

util_env_replace, util_env_str, util_env_free, util_env_create

util_env_free (declared in base/util.h)

The **util_env_free** function frees a specified environment. Use this function to deallocate an environment that you created using the function **util_env_create**.

Syntax

```
#include <base/util.h>
void util_env_free(char **env);
```

Returns

```
void
```

Parameters

**char ****_env_ is the environment to be freed.

See also

util_env_replace, util_env_str, util_env_find, util_env_create

util_env_replace (declared in base/util.h)

The **util_env_replace** function replaces the occurrence of the variable denoted by a name in a specified environment with a specified value. Use this function to change the value of a setting in an environment.

Syntax

```
#include <base/util.h>
void util_env_replace(char **env, char *name, char *value);
```

Returns

```
void
```

Parameters

**char ****_env_ is the environment.

**char ***_name_ is the name of a name-value pair.

**char ***_value_ is the new value to be stored.

See also

util_env_str, util_env_free, util_env_find, util_env_create

util_env_str (declared in base/util.h)

The **util_env_str** function creates an environment entry and returns it. This function does not check for nonalphanumeric symbols in the name (such as the equal sign "="). You can use this function to create a new environment entry.

Syntax

```
#include <base/util.h>
char *util_env_str(char *name, char *value);
```

Returns

A newly allocated string containing the name-value pair

Parameters

**char ***name* is the name of a name-value pair.

**char ***value* is the new value to be stored.

See also

util_env_replace, util_env_free, util_env_find, util_env_create

util_get_current_gmt (declared in libproxy/util.h)

The **util_get_current_gmt** function obtains the current time, represented in terms of GMT (Greenwich Mean Time).

Syntax

```
#include <libproxy/util.h>
time_t util_get_current_gmt(void);
```

Returns

the current GMT

Parameters

No parameter is required.

See also

util_make_local

util_get_int_from_aux_file (declared in libproxy/cutil.h)

The **util_get_int_from_aux_file** function obtains an integer from a specified file.

Syntax

```
#include <libproxy/cutil.h>
int util_get_int_from_file(char *root, char *name);
```

Returns

an integer from the file.

Parameters

**char ***root* is the name of the directory containing the file to be read.

**char ***name* is the name of the file to be read.

See also

util_get_long_from_aux_file, util_get_string_from_aux_file, util_get_int_from_file, util_get_long_from_file, util_get_string_from_file, util_put_int_to_file, util_put_long_to_file, uutil_put_string_to_aux_file, util_put_string_to_file

util_get_long_from_aux_file (declared in libproxy/cutil.h)

The **util_get_long_from_file** function obtains a long from a specified file.

Syntax

```
#include <libproxy/cutil.h>
long util_get_long_from_file(char *root,char *name);
```

Returns

a long integer from the file.

Parameters

**char ***root* is the name of the directory containing the file to be read.

**char ***name* is the name of the file to be read.

See also

util_get_int_from_aux_file, util_get_string_from_aux_file, util_get_int_from_file, util_get_long_from_file, util_get_string_from_file, util_put_int_to_file, util_put_long_to_file, uutil_put_string_to_aux_file, util_put_string_to_file

util_get_string_from_aux_file (declared in libproxy/cutil.h)

The **util_get_string_from_aux_file** function obtains a single line of text from a specified file and returns it as a string.

Syntax

```
#include <libproxy/cutil.h>
char *util_get_string_from_file(char *root, char *name, char *buf,
int maxsize);
```

Returns

a string containing the next line from the file.

Parameters

**char ***_root_ is the name of the directory containing the file to be read.

**char ***_name_ is the name of the file to be read.

**char ***_buf_ is the string to use as the file buffer.

int *maxsize* is the maximum size for the file buffer.

See also

util_get_int_from_aux_file, util_get_long_from_aux_file, util_get_int_from_file, util_get_long_from_file, util_get_string_from_file, util_put_int_to_file, util_put_long_to_file, uutil_put_string_to_aux_file, util_put_string_to_file

util_get_int_from_file (declared in libproxy/cutil.h)

The **util_get_int_from_file** function obtains an integer from a specified file.

Syntax

```
#include <libproxy/cutil.h>
int util_get_int_from_file(char *filename);
```

Returns

- an integer from the file.
- -1 if no value was obtained from the file.

Parameters

**char ****filename* is the name of the file to be read.

See also

util_get_long_from_file, util_get_string_from_file, util_put_int_to_file, util_put_long_to_file, util_put_string_to_file

util_getline (declared in base/util.h)

The **util_getline** function scans the specified buffer to find an LF- or CRLF-terminated string. The function stores the string in another specified buffer, and NULL-terminates it. Finally, the function returns a value that signals whether the operation stored anything in the buffer or encountered an error and whether it reached the end of the file.

Use this function to scan lines out of a text file, such as a configuration file.

Syntax

```
#include <base/util.h>
int util_getline(filebuf *buf, int lineno, int maxlen, char *l);
```

Returns

- 0 if the scan was successful, with the scanned line (less its terminator) in *l*
- 1 if the scan reached an end of file, with the scanned line (less its terminator) in *l*
- -1 if the scan resulted in an error, with the error description in *l*

Parameters

**filebuf ****buf* is the buffer to be scanned.

int *lineno* is used for error diagnostics to include the line number in the error message. The caller is responsible for making sure the line number is accurate.

int *maxlen* is the maximum number of characters that can be written into *l*.

**char ****l* is the buffer into which to store the string. The user is responsible for allocating and deallocating *l*.

See also

util_can_exec, util_env_create, util_host name

util_get_long_from_file (declared in libproxy/cutil.h)

The **util_get_long_from_file** function obtains a long integer from a specified file.

Syntax

```
#include <libproxy/cutil.h>
long util_get_long_from_file(char *filename);
```

Returns

- a long integer from the file.
- -1 if no value was obtained from the file.

Parameters

**char ****file* is the name of the file to be read.

See also

util_get_int_from_file, util_get_string_from_file, util_put_int_to_file, util_put_long_to_file, util_put_string_to_file

util_get_string_from_file (declared in libproxy/cutil.h)

The **util_get_string_from_file** function obtains a single line of text from a specified file and returns it as a string.

Syntax

```
#include <libproxy/cutil.h>
char *util_get_string_from_file(char *filename, char *buf, int maxsize);
```

Returns

- a string containing the next line from the file.
- NULL if no string was obtained.

Parameters

**char ****file* is the name of the file to be read.

**char ****buf* is the string to use as the file buffer.

int *maxsize* is the maximum size for the file buffer.

See also

util_get_int_from_file, util_get_long_from_file, util_put_int_to_file, util_put_long_to_file, util_put_string_to_file

util_grab_lock (declared in libproxy/cutil.h)

The **util_grab_lock** function is for use on systems that do not provide reliable **flock( )** or **lockf( )** functions. It creates a lock file by using the O_EXCL flag for the **open( )** system call to make the open fail if there is already a lock file created by another process.

NOTE This mechanism is resource intensive and should not be used for locking at a rate of more than once a second.

Syntax

```
#include <libproxy/cutil.h>
int util_grab_lock(char *lck, int key, int retries, int break_after_retry);
```

Returns

- nonzero if the operation succeeded
- 0 if the operation failed

Parameters

**char ***lck* is the absolute path to the lock file.

int *key* is the process ID (pid) to use as the key.

int *retries* is the maximum number of retries before abandoning the lock attempt.

int *break_after_retry* signals (by a nonzero value) that if all retries have failed, remove the old lock file and retry again.

See also

util_release_lock

util_host name (declared in base/util.h)

The **util_host name** function retrieves the local host name and returns it as a string. If the function cannot find a fully qualified domain name, it returns NULL. You can reallocate or free this string. Use this function to determine the name of the system you are on.

Syntax

```
#include <base/util.h>}
char *util_host name(void);
```

Returns

- If a fully qualified domain name was found, a string containing that name
- NULL if the fully qualified domain name was not found

Parameters

No parameter is required.

See also

util_can_exec, util_env_create, util_getline

util_is_mozilla (declared in base/util.h)

The **util_is_mozilla** function checks whether a specified user-agent is a Netscape browser of at least a specified revision level, returning a 1 if it is and 0 otherwise. The function uses strings to specify the revision level to avoid ambiguities like 1.56 > 1.5.

Syntax

```
#include <base/util.h>
int util_is_mozilla(char *ua, char *major, char *minor);
```

Returns

- 1 if the user-agent is a Netscape browser
- 0 if the user-agent is not a Netscape browser

Parameters

**char ***_ua_ is the user-agent.

**char ***_major_ is the major release number (to the left of the decimal point).

**char ***_minor_ is the minor release number (to the right of the decimal point).

Example

See the example under **shexp_cmp**

See also

util_is_url, util_later_than

util_is_url (declared in base/util.h)

The **util_is_url** function checks whether a string is a URL, returning 1 if it is and 0 otherwise.

Syntax

```
#include <base/util.h>
int util_is_url(char *url);
```

Returns

- 1 if the string specified by *url* is a URL
- 0 if the string specified by *url* is not a URL

Parameters

**char ****url* is the string to be examined.

See also

util_is_mozilla, util_later_than

util_itoa (declared in base/util.h)

The **util_itoa** function converts a specified integer to a string and returns the length of the string. Use this function to create a textual representation of a number.

Syntax

```
#include <base/util.h>
int util_itoa(int i, char *a);
```

Returns

The length of the string created in *a*

Parameters

int *i* is the integer to be converted.

**char ****a* is the ASCII string that represents the value. The user is responsible for the allocation and deallocation of *a*, which should be at least 32 bytes long.

See also

util_sh_escape

util_later_than (declared in base/util.h)

The **util_later_than** function compares the date specified in a time structure against a date specified in a string. If the date in the string is later than or equal to the one in the time structure, the function returns 1. Use this function to handle RFC 822, 850, and ctime formats.

Syntax

```
#include <base/util.h>
int util_later_than(struct tm *lms, char *ims);
```

Returns

- 1 if the date represented by *ims* is the same as or later than that represented by the *lms*
- 0 if the date represented by *ims* is earlier than that represented by the *lms*

Parameters

**tm ****lms* is the time structure containing a date.

**char ****ims* is the string containing a date.

See also

util_is_mozilla, util_is_url, util_itoa

util_make_filename (declared in libproxy/cutil.h)

The **util_make_filename** function concatenates a directory name and a filename into a newly created string. This can be handy when you are dealing with a number of files that all go to the same directory.

Syntax

```
#include <libproxy/cutil.h>
char *util_make_filename(char *root, char *name);
```

Returns

A new string containing the directory name concatenated with the filename.

Parameters

char **root* is a string containing the directory name.

char **name* is a string containing the filename.

See also

util_make_lockname

util_make_gmt (declared in libproxy/util.h)

The **util_make_gmt** function converts a given local time to GMT (Greenwich Mean Time), or obtains the current GMT.

Syntax

```
#include <libproxy/util.h>
time_t util_make_gmt(time_t t);
```

Returns

- the GMT equivalent to the local time *t*, if t is not 0
- the current GMT if *t* is 0

Parameters

time_t *t* is a time.

See also

util_make_local, *util_get_current_gmt*

util_make_local (declared in libproxy/util.h)

The **util_make_local** function converts a given GMT to local time.

Syntax

```
#include <libproxy/util.h>
time_t util_make_local(time_t t);
```

Returns

the local equivalent to the GMT *t*

Parameters

time_t *t* is a time.

See also

util_make_gmt, util_get_current_gmt

util_make_lockname (declared in libproxy/cutil.h)

The **util_make_lockname** function concatenates a filename and a lock suffix into a newly created string.

Syntax

```
#include <libproxy/cutil.h>
char *util_make_lockname(char *fn);
```

Returns

A new string containing the filename concatenated with the lock suffix.

Parameters

**char ***fn* is a string containing the filename.

See also

util_make_filename

util_make_printable (declared in libproxy/cutil.h)

The **util_make_printable** function copies the contents of a binary buffer into a printable buffer. A binary buffer may contain any characters, while a printable buffer must contain only printable characters.

Syntax

```
#include <libproxy/cutil.h>
void util_make_printable(unsigned char *b, int blen, char *p,
int plen);
```

Returns

```
void
```

Parameters

**char ***_b_ is a binary buffer.

int _blen_ is the length of buffer _b_.

**char ***_p_ is a printable buffer of length _plen_+1 into which the contents of buffer _b_ are copied. The only characters copies are letters, digits, and punctuation characters. No control characters, and not even spaces, are copies.

int _plen_ is the length of buffer _b_ up to, but not including, a terminating NULL.

util_move_dir (declared in libproxy/util.h)

The **util_move_dir** function moves a directory, preserving permissions, creation times, and last-access times. It attempts to do this by renaming, but if that fails (for example, if the source and destination are on two different file systems), it copies the directory.

Syntax

```
#include <libproxy/util.h>
int util_move_dir (char *src, char *dst);
```

Returns

- 0 if the move failed
- nonzero if the move succeeded

Parameters

**char ***_src_ is the fully qualified name of the source directory.

**char ***_dst_ is the fully qualified name of the destination directory.

See also

_util_move_file_

util_move_file (declared in libproxy/util.h)

The **util_move_dir** function moves a file, preserving permissions, creation time, and last-access time. It attempts to do this by renaming, but if that fails (for example, if the source and destination are on two different file systems), it copies the file.

Syntax

```
#include <libproxy/util.h>
int util_move_file (char *src, char *dst);
```

Returns

- 0 if the move failed
- nonzero if the move succeeded

Parameters

**char ***_src_ is the fully qualified name of the source file.

**char ***_dst_ is the fully qualified name of the destination file.

See also

util_move_dir

util_parse_http_time (declared in libproxy/util.h)

The **util_parse_http_time** function converts a given HTTP time string to **time_t** format.

Syntax

```
#include <libproxy/util.h>
time_t util_parse_http_time(char *date_string);
```

Returns

the **time_t** equivalent to the GMT *t*

Parameters

time_t *t* is a time.

See also

util_make_gmt, util_get_current_gmt

util_put_int_to_file (declared in libproxy/cutil.h)

The **util_put_int_to_file** function writes a single line containing an integer to a specified file.

Syntax

```
#include <libproxy/cutil.h>
int util_put_int_to_file(char *filename, int i);
```

Returns

- nonzero if the operation succeeded
- 0 if the operation failed

Parameters

char ***file*** is the name of the file to be written.

int *i* is the integer to write.

See also

util_get_int_from_file, util_get_long_from_file, util_put_long_to_file, util_put_string_to_file

util_put_long_to_file (declared in libproxy/cutil.h)

The **util_put_long_to_file** function writes a single line containing a long integer to a specified file.

Syntax

```
#include <libproxy/cutil.h>
ing util_put_long_to_file(char *filename, long l);
```

Returns

- nonzero if the operation succeeded
- 0 if the operation failed

Parameters

char ***file*** is the name of the file to be written.

long *l* is the long integer to write.

See also

util_get_int_from_file, util_get_long_from_file, util_put_int_to_file, util_put_string_to_file

util_put_string_to_aux_file (declared in libproxy/cutil.h)

The **util_put_string_to_aux_file** function writes a single line containing a string to a file specified by directory name and file name.

Syntax

```
#include <libproxy/cutil.h>
int util_put_string_to_aux_file(char *root, char *name, char *str);
```

Returns

- non-zero if the operation succeeded
- 0 if the operation failed

Parameters

**char ****root* is the name of the directory where the file is to be written.

**char ****name* is the name of the file is to be written.

**char ****str* is the string to write.

See also

util_get_int_from_file, util_get_long_from_file, util_put_int_to_file, util_put_long_to_file, util_put_string_to_file

util_put_string_to_file (declared in libproxy/cutil.h)

The **util_put_string_to_file** function writes a single line containing a string to a specified file.

Syntax

```
#include <libproxy/cutil.h>
int util_put_string_to_file(char *filename, char *str);
```

Returns

- nonzero if the operation succeeded
- 0 if the operation failed

Parameters

char ***file*** is the name of the file to be read.

char ***str*** is the string to write.

See also

util_get_int_from_file, util_get_long_from_file, util_put_int_to_file, util_put_long_to_file

util_release_lock (declared in libproxy/cutil.h)

The **util_release_lock** function releases a lock established by the **util_grab_lock** function.

Syntax

```
#include <libproxy/cutil.h>
int util_release_lock(char *lck, int key, int force);
```

Returns

- nonzero if the operation succeeded
- 0 if the operation failed

Parameters

char ***lck*** is the absolute path to the lock file to be removed.

int *key* is the process ID (pid) to use as the key.

int *force* indicates (by a value other than zero) that the lock is to be removed even if the pid in that file does not match the pid of the process making the call to **util_release_lock**.

See also

util_grab_lock

util_sect_id (declared in libproxy/cutil.h)

The **util_sect_id** function creates a section ID from the section dim and an index.

Syntax

```
#include <libproxy/cutil.h>
void util_sect_id(int dim, int idx, char *buf);
```

Returns

- nonzero if the operation succeeded
- 0 if the operation failed

Parameters

int *dim* is the section dim.

int *idx* is the index.

**char ****buf* is the buffer to receive the section ID.

util_sh_escape (declared in base/util.h)

The **util_sh_escape** function parses a specified string and places a backslash (\) in front of any shell-special characters, returning the resultant string. Use this function to ensure that strings from clients won't cause a shell to do anything unexpected.

Syntax

```
#include <base/util.h>
char *util_sh_escape(char *s);
```

Returns

A newly allocated string

Parameters

**char ****s* is the string to be parsed.

See also

util_uri_escape

util_snprintf (declared in base/util.h)

The **util_snprintf** function formats a specified string, using a specified format, into a specified buffer using the **printf**-style syntax and performs bounds checking. It returns the number of characters in the formatted buffer.

For more information, see the documentation on the **printf** function for the run-time library of your compiler.

Syntax

```
#include <base/util.h>
int util_snprintf(char *s, int n, char *fmt, ...);
```

Returns

The number of characters formatted into the buffer.

Parameters

**char ***_s_ is the buffer to receive the formatted string.

int _n_ is the maximum number of bytes allowed to be copied.

**char ***_fmt_ is the format string. The function handles only %d and %s strings; it does not handle any width or precision strings.

... represents a sequence of parameters for the **printf** function.

Example

Similar to the example for **util_sprintf**.

See also

util_sprintf, util_vsnprintf, util_vsprintf

util_sprintf (declared in base/util.h)

The **util_sprintf** function formats a specified string, using a specified format, into a specified buffer using the **printf**-style syntax without bounds checking. It returns the number of characters in the formatted buffer.

Because **util_sprintf** doesn't perform bounds checking, use this function only if you are certain that the string fits the buffer. Otherwise, use the function **util_snprintf**. For more information, see the documentation on the **printf** function for the run-time library of your compiler.

Syntax

```
#include <base/util.h>
int util_sprintf(char *s, char *fmt, ...);
```

Returns

The number of characters printed.

Parameters

**char ***_s_ is the buffer to receive the formatted string.

**char ***fmt* is the format string. The function handles only %d and %s strings; it does not handle any width or precision strings.

... represents a sequence of parameters for the **printf** function.

Example

```
int brief_log(pblock *pb, Session *sn, Request *rq)
{
char *method = pblock_findval("method", rq->reqpb);
   char *uri = pblock_findval("uri", rq->reqpb);
   char *ip = pblock_findval("ip", sn->client);
   /* Temp vars */
   char *logmsg;
   int len;
   logmsg = (char *)
      MALLOC(strlen(ip) + 1 + strlen(method) + 1
            + strlen(uri) + 1 + 1);
   len = util_sprintf(logmsg, "%s %s %s\n", ip, method, uri);
   /* The atomic version uses locking to prevent interference */
   system_fwrite_atomic(logfd, logmsg, len);
   FREE(logmsg);
   return REQ_PROCEED;
}
```

See also

util_snprintf, *util_vsnprintf*, *util_vsprintf*

util_strcasecmp (declared in base/systems.h)

The **util_strcasecmp** function performs a comparison of two alphanumeric strings and returns a -1, 0, or 1 to signal which is larger or that they are identical. The function's comparison is not case-sensitive.

Syntax

```
#include <base/systems.h>
int util_strcasecmp(const char *s1, const char *s2);
```

Returns

- 1 if *s1* is greater than *s2*
- 0 if *s1* is equal to *s2*
- -1 if *s1* is less than *s2*

Parameters

**char ****s1* is the first string.

**char ****s2* is the second string.

See also

util_strncasecmp

util_strncasecmp (declared in base/systems.h)

The **util_strncasecmp** function performs a comparison of the first *n* characters in the alphanumeric strings and returns a -1, 0, or 1 to signal which is larger or that they are identical. The function's comparison is not case-sensitive.

Syntax

```
#include <base/systems.h>
int util_strncasecmp(const char *s1, const char *s2, int n);
```

Returns

- 1 if *s1* is greater than *s2*
- 0 if *s1* is equal to *s2*
- -1 if *s1* is less than *s2*

Parameters

**char ****s1* is the first string.

**char ****s2* is the second string.

int *n* is the number of initial characters to compare.

See also

util_strcasecmp

util_uri_check (declared in libproxy/util.h)

The **util_uri_check** function checks that a URI has a format conforming to the standard.

At present, the only URI it checks for is a URL. The standard format for a URL is

protocol`://`*user*`:`*password@host*`:`*port*`/`*url-path* where *user*:*password*, :*password*. :*port*, or /*url-path* can be omitted.

Syntax

```
#include <libproxy/util.h>
int util_uri_check (char *uri);
```

Returns

- 0 if the URI does not have the proper form.
- nonzero if the URI has the proper form.

Parameters

**char ***_uri_ is the URI to be tested.

util_uri_escape (declared in base/util.h)

The **util_uri_escape** function converts any special characters in a specified string into the URI format, and returns the escaped string. Use **util_uri_escape** before sending the URI back to the client.

Syntax

```
#include <base/util.h>
char *util_uri_escape(char *d, char *s);
```

Returns

The string (possibly newly allocated) with escaped characters replaced.

Parameters

**char ***_d_ is a string. If *d* is not NULL, the function copies the formatted string into *d* and returns it. If *d* is NULL, the function allocates a properly sized string and copies the formatted special characters into the new string, then returns it.

The **util_uri_escape** function does not check bounds for the parameter *d*. Therefore, *d* should be at least three times as large as *s*.

**char ***_s_ is the string containing the unescaped form of the URI.

See also

util_uri_is_evil, util_uri_parse, util_uri_unescape

util_uri_is_evil (declared in base/util.h)

The **util_uri_is_evil** function checks a specified URI and returns 1 if it contains ../ or //. Use this function to make sure that a URI given by a client won't do anything unexpected.

Syntax

```
#include <base/util.h>
int util_uri_is_evil(char *t);
```

Returns

- 1 if the URI contains ../ or //
- 0 if the URI does not contain ../ or //

Parameters

char ***t*** is the URI to be checked.

See also

util_uri_escape, *util_uri_parse*

util_uri_parse (declared in base/util.h)

The **util_uri_parse** function removes /../, /./, and // in a specified URI. You can use this function to convert a URI's bad sequences into valid ones. First use the function **util_uri_is_evil** to determine whether the function has a bad sequence.

Syntax

```
#include <base/util.h>
void util_uri_parse(char *uri);
```

Returns

`void`

Parameters

char ***uri*** is the URI to be converted.

See also

util_uri_is_evil, *util_uri_unescape*

util_uri_unescape (declared in base/util.h)

The **util_uri_unescape** function converts the encoded characters of a specified URI into special characters in place.

Syntax

```
#include <base/util.h>
void util_uri_unescape(char *uri);
```

Returns

```
void
```

Parameters

**char ***uri* is the URI to be converted.

See also

util_uri_escape, util_uri_is_evil, util_uri_parse

util_url_cmp (declared in libproxy/util.h)

The **util_url_cmp** function compares two URLs. It is analogous to the **strcmp( )** library function of C.

Syntax

```
#include <libproxy/util.h>
int util_url_cmp (char *s1, char *s2);
```

Returns

- **-**1 if the first URL, *s1*, is less than the second, *s2*
- 0 if they are identical
- 1 if the first URL, *s1*, is greater than the second, *s2*

Parameters

**char ***s1* is the first URL to be tested.

**char ***s2* is the second URL to be tested.

See also

util_url_fix_host name, util_uri_check

util_url_fix_host name (declared in libproxy/util.h)

The **util_url_fix_host name** function converts the host name in a URL to lowercase and removes redundant port numbers.

Syntax

```
#include <libproxy/util.h>
void util_url_fix_host name(char *url);
```

Returns

`void` (but changes the value of its parameter string)

The protocol specifier and the host name in the parameter string are changed to lowercase. The function also removes redundant port numbers, such as 80 for HTTP, 70 for gopher, and 21 for FTP.

Parameters

**char ***url* is the URL to be converted.

See also

util_url_cmp, util_uri_check.

util_url_has_FQDN (declared in libproxy/util.h)

The **util_url_has_FQDN** function returns a value to indicate whether a specified URL references a fully qualified domain name.

Syntax

```
#include <libproxy/util.h>
int util_url_has_FQDN(char *url);
```

Returns

- `1` if the URL has a fully qualified domain name
- 0 if the URL does not have a fully qualified domain name

Parameters

**char ***url* is the URL to be examined.

util_vsnprintf (declared in base/util.h)

The **util_vsnprintf** function formats a specified string, using a specified format, into a specified buffer using the **vprintf**-style syntax and performs bounds checking. It returns the number of characters in the formatted buffer.

For more information, see the documentation on the **printf** function for the run-time library of your compiler.

Use this function if you want a **vsprintf** syntax that takes a standard arg format. For more information, see the documentation on the **vsprintf** function for the run-time library of your compiler.

Syntax

```
#include <base/util.h>
int util_vsnprintf(char *s, int n, register char *fmt, va_list args);
```

Returns

The number of characters printed

Parameters

**char ****s* is the buffer to receive the formatted string.

int *n* is the maximum number of bytes allowed to be copied.

**register char ****fmt* is the format string. The function handles only %d and %s strings; it does not handle any width or precision strings.

va_list *args* is an STD arg variable obtained from a previous call to **va_start**.

See also

util_sprintf, util_vsprintf

util_vsprintf (declared in base/util.h)

The **util_vsprintf** function formats a specified string, using a specified format, into a specified buffer using the **vprintf**-style syntax without bounds checking. It returns the number of characters in the formatted buffer.

For more information, see the documentation on the **printf** function for the run-time library of your compiler.

Use this function if you want a **vsprintf** syntax that takes a standard arg format. For more information, see the documentation on the **vsprintf** function for the runtime library of your compiler.

Syntax

```
#include <base/util.h>
int util_vsprintf(char *s, register char *fmt, va_list args);
```

Returns

The number of characters printed

Parameters

**char ***_s_ is the buffer to receive the formatted string.

**register char ***_fmt_ is the format string. The function handles only %d and %s strings; it does not handle any width or precision strings.

va_list _args_ is an STD arg variable obtained from a previous call to **va_start**.

See also

util_snprintf, util_vsnprintf

Appendix B

Server Data Structures

The server plug-in API uses many data structures. All their definitions are gathered here for your convenience.

The Session Data Structure

A *session* is the time between the opening and the closing of the connection between the client and the server. The Session data structure holds variables that apply session wide, regardless of the requests being sent, as shown in this code. It is defined in the base/session.h file.

```
typedef struct {
/* Information about the remote client */
   pblock *client;

   /* The socket descriptor to the remote client */
   SYS_NETFD csd;
   /* The input buffer for that socket descriptor */
   netbuf *inbuf;

/* Raw socket information about the remote */
/* client (for internal use) */
   struct in_addr iaddr;
} Session;
```

The Parameter Block (pblock) Data Structure

The parameter block is the hash table that holds pb_entry structures. Its contents are transparent to most code. It is defined in the base/pblock.h file.

```
#include "base/pblock.h"
```

```
typedef struct {
   int hsize;
   struct pb_entry **ht;
} pblock;
```

The Pb_entry Data Structure

The pb_entry data structure is a single element in the parameter block. It is defined in the `base/pblock.h` file.

```
struct pb_entry {
   pb_param *param;
   struct pb_entry *next;
};
```

The Pb_param Data Structure

The pb_param data structure represents a name-value pair, as stored in a pb_entry. It is defined in the `base/pblock.h` file.

```
typedef struct {
   char *name,*value;
} pb_param;
```

The Client Parameter Block

The Session->client parameter block structure, defined in the `base/session.h` file, contains two entries:

- The IP entry is the IP address of the client machine.
- The DNS entry is the DNS name of the remote machine. This member must be accessed through the **session_dns** function call:

```
/*
* session_dns returns the DNS host name of the client for this
* session and inserts it into the client pblock. Returns NULL if
* unavailable.
*/

char *session_dns(Session *sn);
```

The Request Data Structure

Under HTTP protocol, there is only one request per session. The Request structure contains the variables that apply to the request in that session (for example, the variables include the client's HTTP headers). It is declared in the `frame/req.h` file.

```
typedef struct {
   /* Server working variables */
   pblock *vars;

   /* The method, URI, and protocol revision of this request */
   block *reqpb;
   /* Protocol specific headers */
   int loadhdrs;
   pblock *headers;

   /* Server's response headers */
   pblock *srvhdrs;

   /* The object set constructed to fulfill this request */
   httpd_objset *os;

   /* The stat last returned by request_stat_path */
   char *statpath;
   struct stat *finfo;
} Request;
```

The Stat Data Structure

When the program calls the **stat( )** function for a given file, the system returns a structure that provides information about the file. The specific details of the structure must be obtained from your own implementation, but the basic outline of the structure is as follows:

```
struct stat {
   dev_t     st_dev;/* device of inode */
   inot_tst_ino;/* inode number */
   shortst_mode;/* mode bits */
   shortst_nlink;/* number of links to file /*
   shortst_uid;/* owner's user id */
   shortst_gid;/* owner's group id */
   dev_tst_rdev;/* for special files */
   off_tst_size;/* file size in characters */
   time_tst_atime;/* time last accessed */
   time_tst_mtime;/* time last modified */
   time_tst_ctime;/* time inode last changed*/
}.
```

The elements that are most significant for server plug-in API activities are st_size, st_atime, st_mtime, and st_ctime.

The Shared Memory Structure, Shmem_s

```
typedef struct {
void *data;   /* the data */
   int size;     /* the maximum length of the data */
   char *name;   /* internal use: filename to unlink if exposed */
   SYS_FILE fd;  /* internal use: file descriptor for region */
} shmem_s;
```

The Netbuf Data Structure

The netbuf data structure is a platform-independent network-buffering structure that maintains such members as buffer address, position in buffer, current file size, maximum file size, and so on. Details of its structure vary between implementations. It is defined in `buffer.h`.

The Filebuffer Data Structure

The filebuffer data structure is a platform-independent file-buffering structure that maintains such members as buffer address, file position, current file size, and so on. Details of its structure vary between implementations. It is defined in `buffer.h`.

The Cinfo Data Structure

The cinfo data structure records the content information for a file. It is defined in `cinfo.h`.

```
typedef struct {
   char *type;/* Identifies what kind of data is in the file*/
   char *encoding;/* Identifies any compression or other content*/-
             /* independent transformation that's been applied*/
             /* to the file, such as uuencode)*/
   char *language;/* Identifies the language a text document is in.
*/
} cinfo;
```

The SYS_NETFD Data Structure

The SYS_NETFD data structure is a platform-independent socket descriptor. Details of its structure vary between implementations.

The SYS_FILE Data Structure

The SYS_FILE data structure is a platform-independent file descriptor. Details of its structure vary between implementations.

The SEMAPHORE Data Structure

The SEMAPHORE data structure is a platform-independent implementation of semaphores. Details of its structure vary between implementations. It is defined in `sem.h`.

The Sockaddr_in Data Structure

The socaddr_in data structure is a platform-dependent socket address. For UNIX proxies, go to `netinet/in.h`.

The CONDVAR Data Structure

The CONDVAR data structure is a platform-independent implementation of a condition variable. Details of its structure may vary between implementations. It is defined in `crit.h`.

The CRITICAL Data Structure

The CRITICAL data structure is a platform-independent implementation of a critical-section variable. Details of its structure may vary between implementations. It is defined in crit.h.

The SYS_THREAD Data Structure

The SYS_THREAD data structure is a platform-independent implementation of a system-thread variable. Details of its structure may vary between implementations. It is defined in systhr.h.

The CacheEntry Data Structure

The CacheEntry data structure holds all the information about one cache entry. It is created by the **ce_lookup** function and destroyed by the **ce_free** function. It is defined in the libproxy/cache.h file.

```
typedef struct _CacheEntry {
   CacheState  state;/* state of the cache file; DO NOT refer to any
              * of the other fields in this C struct if state
              * is other than
              *     CACHE_REFRESH or
              *     CACHE_RETURN_FROM_CACHE
              */
SYS_FILE    fd_in;/* do not use: open cache file for reading */
int fd_out;/* do not use: open (locked) cache file for writing */
struct stat finfo;/* stat info for the cache file */
unsigned char  digest[CACHE_DIGEST_LEN];/* MD5 for the URL */
char * url_dig;  /* URL used to for digest; field #8 in CIF */
char * url_cif;  /* URL read from CIF file */
char * filname;  /* Relative cache file name */
char * dirname;  /* Absolute cache directory name */
char * absname;  /* Absolute cache file path */
char * lckname;  /* Absolute locked cache file path */
char * cifname;  /* Absolute CIF path */
int    sect_idx; /* Cache section index */
int    part_idx; /* Cache partition index */
CSect *section;  /* Cache section that this file belongs to */
CPart *partition;/* Cache partition that this file belongs to */
int    xfer_time;/* secs *//* Field #2 in CIF */
time_t last_modified;/* GMT *//* Field #3 in CIF */
time_t expires;  /* GMT */ /* Field #4 in CIF */
time_t last_checked;/* GMT *//* Field #5 in CIF */
long   content_length;     /* Field #6 in CIF */
char * content_type;       /* Field #7 in CIF */
int    is_auth;  /* Authenticated data -- always do recheck */
int    auth_sent;/* Client did send the Authorization header */
longmin_size;    /* Min size for a cache file (in KB) */
longmax_size;    /* Max size for a cache file (in KB) */
time_t last_accessed;/* GMT for proxy, local for gc */
time_t created;  /* localtime (only used by gc, st_mtime) */
int    removed;  /* gc only; file was removed from disk */
long   bytes;    /* from stat(), using this we get hdr len */
long   bytes_written;/* Number of bytes written to disk */
long   bytes_in_media;/* real fs size taken up */
long   blks;     /* size in 512 byte blocks */
int    category; /* Value category; bigger is better */
int    cif_entry_ok;/* CIF entry found and ok */
time_t ims_c;    /* GMT; Client -> proxy if-modified-since */
time_t start_time;/* Transfer start time */
int    inhibit_caching;/* Bad expires/other reason not to cache */
int corrupt_cache_file;/* Cache file gone corrupt => remove */
int write_aborted;/* True if the cache file write was aborted */
int batch_update;/* We’re doing batch update (no real user) */
char * cache_exclude;/* Hdrs not to write to cache (RE) */
char * cache_replace;/* Hdrs to replace with fresh ones from 304
response (RE) */
char * cache_nomerge;/* Hdrs not to merge with the cached ones (RE)
*/
Session * sn;
Request * rq;
} CacheEntry;
```

The CacheState Data Structure

The CacheState data structure is actually an enumerated list of constants. Aways use their names because values are subject to implementation change.

```
typedef enum {
CACHE_EXISTS_NOT = 0,/* Internal flag -- do not use! */
CACHE_EXISTS,    /* Internal flag -- do not use! */
CACHE_NO,        /* No caching: don’t read, don’t write cache */
CACHE_CREATE,    /* Create cache; don’t read */
CACHE_REFRESH,   /* Refresh cache; read if not modified */
CACHE_RETURN_FROM_CACHE,/* Return directly, no check */
CACHE_RETURN_ERROR/* With connect-mode=never when not in cache */
} CacheState;
```

The ConnectMode Data Structure

The ConnectMode data structure is actually an enumerated list of constants. Aways use their names because values are subject to implementation change.

```
typedef enum {
CM_NORMAL = 0,/* normal -- retrieve/refresh when necessary */
CM_FAST_DEMO,/* fast -- retrieve only if not in cache already */
CM_NEVER  /* never -- never connect to network */
} ConnectMode;
```

Appendix C

Proxy Configuration Files

This appendix describes the directives and functions in the configuration files that iPlanet Web Proxy Server uses. You can configure iPlanet Web Proxy Server manually by editing the files directly.

The files that you use to configure the proxy are in a directory called `proxy-`*id*`/config` in your server root directory. Here's a brief description of each file described in this appendix:

- `magnus.conf` is the server's main technical configuration file. It controls aspects of the server operation not related to specific resources or documents, such as host name and port.
- `obj.conf` is the server's object configuration file. It controls access to the proxy server, and determines how documents are proxied and cached.
- `socks5.conf` is a file that contains the SOCKS server configuration. The SOCKS daemon is a generic firewall daemon that controls point-to-point access through the firewall.
- `bu.conf` is an optional file that contains batch update directives. You can use these to update many documents at once. You can time batch updates; for example, you can have them occur during off-peak hours to minimize the effect on the efficiency of the server.
- `icp.conf` is the Internet Cache Protocol (ICP) configuration file. It identifies the information about the parent and sibling servers in a proxy array that uses ICP.

Other files that affect the proxy are explained elsewhere in this book:

- `mime.types` is the file the server uses to convert filename extensions such as `.GIF` into a MIME type like `image/gif`. This file is described in Chapter 16, "Configuring the Proxy Manually."

- `admpw` is the administrative password file. Its format is user:password. The password is DES-encrypted just like `/etc/passwd`. This file is described in Chapter 16, “Configuring the Proxy Manually.” The `admpw` file is located in the `admin-serv` directory.
- `autoconfig` is a file in Netscape Navigator 2.0 JavaScript software that enables you to specify when to use the proxy. This file is described in Chapter 11, “Using the Client Autoconfiguration File.”
- `parray.pat` is the Proxy Array Table file. The PAT file is an ASCII file used in proxy to proxy routing. It contains information about a proxy array; including the members’ machine names, IP addresses, ports, load factors, cache sizes, etc. For more information on the syntax of the `parray.pat` file, see “The parray.pat File,” on page 266.
- `parent.pat` is the Proxy Array Table file that contains information about an upstream proxy array. For more information on the syntax of the `parent.pat` file, see “The parent.pat File,” on page 267.

The magnus.conf File

For each directive, this section provides the directive’s characteristics, including the directive name, description, format for the value string, default value if the directive is omitted, and how many times the directive can be in the file. The directives are:

- **Certfile** specifies the location of the certificate file.
- **Ciphers** specifies which ciphers are enabled for your server.
- **DNS** specifies if the server does DNS lookups on clients who access the server.
- **ErrorLog** specifies the directory where the server logs its errors.
- **Keyfile** specifies the location of the key file.
- **LDAPConnPool** specifies the number of persistent connections to the LDAP directory.
- **LoadObjects** specifies a startup object configuration file.
- **MaxProcs** sets the maximum number of active processes.
- **PidLog** specifies a file to record the proxy’s main process ID (*pid*).
- **Port** defines the TCP port to which the server listens.

- **ProcessLife** specifies the number of requests each child process serves during its lifetime.
- **RootObject** defines the default server object.
- **Security** specifies whether SSL is enabled or disabled.
- **ServerName** defines the proxy host name.
- **SSLClientAuth** requires that client authentication be done for every request.
- **SSL2** specifies whether SSL 2.0 is enabled or disabled.
- **SSL3** specifies whether SSL 3.0 is enabled or disabled.
- **SSL3Ciphers** specifies which encryption schemes are enabled.
- **User** specifies the proxy's UNIX user account.

Certfile

The **Certfile** directive specifies where the certificate file is located.

Syntax

`certfile [`*filename*`]`

`certfile` is the server's certificate file, specified as a relative path from the server root or as an absolute path.

Ciphers

The **Ciphers** directive specifies the ciphers enabled for your server. For a discussion of the pros and cons of these ciphers, see Chapter 14, "Understanding Encryption and SSL."

Syntax

```
Ciphers +rc4 +rc4export -rc2 -rc2export +idea +des +desede3
```

A + means the cipher is active, and a **-** means the cipher is inactive.

Valid ciphers are `rc4`, `rc4export`, `rc2`, `rc2export`, `idea`, `des`, `desede3`. Any cipher with `export` as port of its name is not stronger than 40 bits.

DNS

The **DNS** directive specifies whether the server performs DNS lookups on clients accessing the server. When a client connects to your server, the server knows the client's IP address but not its host name (for example, it knows the client as 198.95.251.30, rather than its host name www.a.com). The server will resolve the client's IP address into a host name for operations like access control, CGI, error reporting, and access logging.

If your server responds to many requests per day, you might want (or need) to stop host name resolution; doing so can reduce the load on the DNS or NIS server.

Syntax

```
DNS [on|off]
```

Default

DNS host name resolution is on as a default.

ErrorLog

The **ErrorLog** directive specifies the directory where the server logs its errors. You can also use the syslog facility. If errors are reported to a file (instead of syslog), then the file and directory in which the log is kept must be writable by whatever user account the server runs as.

Syntax

`ErrorLog` *logfile*

logfile can be a full path and filename or the keyword SYSLOG (which must be in all capital letters).

Keyfile

The **Keyfile** directive tells the server where the key file is located.

Syntax

`keyfile [`*filename*`]`

`keyfile` is the server's key file, specified as a relative path from the server root or as an absolute path.

LDAPConnPool

The LDAPConnPool Directive specifies the number of persistent connections to maintain to the LDAP directory server. Creating these connections and binding to the directory on each one is an expensive operation. This setting establishes a reasonably sized pool of connections that will be shared among the proxy's request handler threads. Increase this value to allow more connections and to improve proxy performance if your directory server is not overloaded. Decrease the value if your directory server is very busy.

The default value for LDAPConnPool is 5.

LoadObjects

The **LoadObjects** directive specifies one or more object configuration files to use on startup; these files tell the server the kinds of URLs to proxy and cache. If any **User** directive is in the `magnus.conf` file, it must appear before the **LoadObjects** directive.

Although you can have more than one object configuration file, the proxy's administration forms work with only one file and assume that it is in the server root in `proxy-`*`id`*`/config/obj.conf`. If you use the online forms (or plan to), don't put the `obj.conf` file in any other directory and don't rename it.

Syntax

`LoadObjects` *filename*

filename is either the full pathname or a relative pathname. Relative pathnames are resolved from the directory specified with the -d command line flag. If no -d flag is given, the server looks in the current directory.

MaxProcs

The **MaxProcs** directive sets the maximum number of processes the server can have active. The **MaxProcs** number is the number of processes that the proxy is to constantly keep active.

When you change **MaxProcs**, you must do a hard restart for the change to take effect.

Choose a number that is appropriate for the volume of access you expect for the proxy server. If this number is too small, clients will experience delays. If the number is too large, you might waste resources that other programs could use.

Syntax

`MaxProcs` *number*

number is a whole number between 1 and the size of your system's process table.

Default

```
MaxProcs 50
```

PidLog

The **PidLog** directive specifies a file in which to record the process ID (pid) of the base proxy server process. Some of the server support programs (including the forms-based iPlanet Web Proxy Server Manager) assume that the pid log is in the server root, in `logs/pid`.

To shut down your server, kill the base server process listed in the pid log file by using a -TERM signal. To tell your server to reread its configuration files and reopen its log files, use kill with the -HUP signal.

If the pid log file isn't writable by the user account that the server uses, the server does not log its process ID anywhere.

Syntax

`PidLog` *file*

file is the full pathname and filename where the process ID is stored.

Port

The **Port** directive controls to which TCP port the server listens. If you choose a port number less than 1024, the server must be started as root or superuser. The port you choose also affects how the proxy users configure their browsers (they must specify the port number when accessing the proxy server). There should be only one **Port** directive in `magnus.conf`.

There are no official port numbers for proxy servers, but two commonly used numbers are 8080 and 8000. If you use iPlanet Web Proxy Server's SOCKS daemon feature, the proxy should use the standard SOCKS port (1080).

Syntax

`Port` *number*

number is a whole number between 0 and 65535.

Default

```
Port 8080
```

ProcessLife

The **ProcessLife** directive specifies the number of requests that each of the proxy's child processes serves before the processes exit and get respawned. When the processes are stopped and restarted, the memory they use is freed and then reused.

Syntax

`ProcessLife` *number*

number is how many requests each child process serves before the processes exit and get respawned.

Default

```
ProcessLife 1024
```

By stopping and restarting a process, the proxy ensures that memory isn't wasted by "lost" processes. For example, on rare occasions, when a connection is terminated by the client or the remote server while the proxy is processing the request, the request fails and some part of allocated memory isn't freed. **ProcessLife** ensures that the memory is eventually freed.

RootObject

The **RootObject** directive tells the server which object loaded from an object file is the server default. The default object is expected to have all of the name translation directives for the server; any server behavior that is configured in the default object affects the entire server.

If you specify an object that doesn't exist, the server doesn't report an error until a client tries to retrieve a document.

Syntax

`RootObject` *name*

name is the name of an object defined in one of the object files loaded with a **LoadObjects** directive.

Default

There is no default if you do not specify a root object name; even if it is the "default" named object, you must specify "default." The administration forms assume you will use the default named object. Don't deviate from this convention if you plan to use the online forms.

Security

The **Security** directive tells the server whether encryption (Secure Sockets Layer version 2 or version 3 or both) is enabled or disabled.

If **Security** is set to on, and both SSL2 and SSL3 are enabled, then the server tries SSL3 encryption first. If that fails, the server tries SSL2 encryption.

Syntax

```
Security on|off
```

Default

By default, security is off.

ServerName

The **ServerName** directive tells the server what to put in the host name section of any URLs it sends back to the client. This affects redirections that you have set up using the online forms. Combined with the port number, this name is what all clients use to access the server.

You can have only one **ServerName** directive in `magnus.conf`.

Syntax

`ServerName` *host*

host is a fully qualified domain name such as myhost.netscape.com.

Default

If **ServerName** isn't in `magnus.conf`, the proxy server attempts to derive a host name through system calls. If they don't return a qualified domain name (for example, they get myhost instead of myhost.netscape.com), the proxy server won't start, and you'll get a message telling you to set this value manually, meaning put a **ServerName** directive in your `magnus.conf` file.

SSLClientAuth

The **SSLClientAuth** directive causes SSL3 client authentication on all requests.

Syntax

```
SSL3ClientAuth on|off
```

on directs that SSL3 client authentication be performed on every request, independent of ACL-based access control.

SSL2

The **SSL2** directive tells the server whether Secure Sockets Layer, version 2 encryption is enabled or disabled. The **Security** directive dominates the **SSL2** directive; if SSL2 encryption is enabled but the **Security** directive is set to off, then it is as though SSL2 were disabled.

Syntax

```
SSL2 on|off
```

Default

By default, security is off.

SSL3

The **SSL3** directive tells the server whether Secure Sockets Layer, version 3 security is enabled or disabled. The **Security** directive dominates the **SSL3** directive; if SSL3 security is enabled but the **Security** directive is set to off, then it is as though SSL3 were disabled.

Syntax

```
SSL3 on|off
```

Default

By default, security is off.

SSL3Ciphers

The **SSL3Ciphers** directive specifies the SSL3 ciphers enabled for your server.

Syntax

```
SSL3Ciphers +rc4 +rc4export -rc2 -rc2export +idea +des +desede3
```

A + means the cipher is active, and a **-** means the cipher is inactive.

Valid ciphers are `rsa_rc4_128_md5`, `rsa3des_sha`, `rsa_des_sha`, `rsa_rc4_40_md5`, `rsa_rc2_40_md5`, and `rsa_null_md5`. Any cipher with `40` as part of its name is 40 bits.

User

The **User** directive specifies the UNIX user account for the proxy server. If the proxy is started by the superuser or root user, the server binds to the port you specify and then switches its user ID to the user account specified with the **User** directive. This directive is ignored if the server isn't started as the superuser. The **User** directive must occur before any **LoadObjects** directive.

The user account you specify should have write permission to the proxy server's root and cache directories. The user account doesn't need any special privileges. Although you can use the nobody user for UNIX proxy servers, it isn't recommended, and it will not work on some systems, such as HP-UX.

Syntax

`User` *login*

login is the eight-character (or fewer) login name for the user account.

Default

If there is no **User** directive, the server runs with the user account with which it was started. If the server was started as root or superuser, you'll see a warning message after startup.

The obj.conf File

This section defines the obj.conf directives and describes their characteristics, including the directive name and description, format for the function string, default value if the directive is omitted, and how many instances of the directive can be in the file. The directives are:

- **AddLog** adds log entries to any log files.
- **AuthTrans** protects server resources from specific users.

- **Connect** provides a hook for you to call a custom connection function.
- **DNS** calls a custom DNS function you specify.
- **Error** sends customized error messages to clients.
- **Filter** provides content-filtering hooks for functions such as virus detection.
- **Init** (a special directive) initializes server subsystems.
- **NameTrans** maps URLs to mirror sites and the local file system.
- **ObjectType** tags additional information to requests.
- **PathCheck** checks URLs after **NameTrans**.
- **Service** sends data and completes the requests.

AddLog

After the request is finished and the proxy server has stopped sending to and receiving from the client, the proxy server logs the transaction. The proxy server records information about every time a client tries to gain access to the content server through the proxy, and it records information about the client making the request.

If an object has more than one **AddLog** directive, all are used.

The **AddLog** directive works with the log file function **init-clf** in the **Init** directive. For example, you could create three separate log files using the **init-clf** function:

```
Init fn=init-clf
   log1=/usr/ns-home/logs/log-one
   log2=/usr/ns-home/logs/log-two
   log3=/usr/ns-home/logs/log-three
```

Later, you can refer to symbolic names **log1**, **log2,** and **log3** from the **AddLog fn=proxy-log** function:

```
AddLog fn=proxy-log name=log1
```

In some other **<Object>** you could have this command to record those accesses in another log file:

```
AddLog fn=proxy-log name=log2
```

To log only the IP address of the client, and not the DNS name, the **AddLog fn-flex-log** function takes one more optional parameter: **iponly=1**. This optional parameter saves CPU cycles because the DNS name of the client host doesn't have to be resolved by contacting the DNS server:

```
AddLog fn=flex-log name=log3 iponly=1
```

If the **name** parameter is left out, it defaults to **global**. The following are equivalent:

```
AddLog fn=flex-log
```

```
AddLog fn=flex-log name=global
```

This function initializes the URL database; it specifies whether to cache URLs and, if so, the directory where they will be contained.

To log URLs, the **AddLog fn=urldb-record** function has to be called for each object ("**<Object>**") for which URL recording is desired. Also, the **init-urldb** function of the **Init** directive status has to be on. If the **init-urldb** status is off, URLs will not be recorded even if the **AddLog fn=urldb-record** function is called. URLs for documents that don't get cached will not be recorded in the URL database.

NOTE Logging of URLs requires special settings. For information about using the **AddLog** function to start logging of URLs, see "init-urldb (setting up the URL database)" on page 440.

flex-log (starting proxy logging)

The **flex-log** function is an **AddLog** function that records request-specific data in the flexible, common, extended (used by most HTTP servers), or extended-2 log format. There are a number of free statistics generators for the common format, but the extended format gives more detailed information about the bytes transferred and the time elapsed. The extended-2 format provides as much information as the extended format, with additional kinds of information: the route through which the document was received as well as the finish status for the remote connection, the client connection, and the cache.

The log format is specified by the **init-proxy** function call, described in "init-proxy (starting the network software for proxy)," on page 435.

Syntax

```
AddLog fn=proxyflex-log
    name=name  iponly
```

Parameters

name (optional) gives the name of a log file, which must have been given as a parameter to the init-clf function of the Init directive. If no name is given, global is the default.

iponly (optional) instructs the server not to look up the host name of the remote client but to record the IP address instead. The value of iponly can be anything, as long as it exists; the online forms set `iponly="1"`.

Example

```
# Log all accesses to the central log file
AddLog fn=flex-log
# Log non-local accesses to another log file
<Client ip=*~198.93.9[2345].*>
AddLog fn=flex-log name=nonlocal
</Client>
```

AuthTrans

AuthTrans is the Authorization Translation directive. Server resources can be protected so that accessing them requires the client to provide certain information about the person using the client program. This authorization information is “encoded” to prevent clients from authorizing themselves as different users.

The server analyzes the authorization of client users in two steps. First, it translates authorization information sent by the client, and then it requires that such authorization information be present. This is done in the hope that multiple translation schemes can be easily incorporated, as well as providing the flexibility to have resources that record authorization information but do not require it.

If there is more than one **AuthTrans** directive in an object, all functions will be applied. The **AuthTrans** directive has a function called **proxy-auth**.

proxy-auth (translating proxy authorization)

The **proxy-auth** function of the **AuthTrans** directive translates authorization information provided through the basic proxy authorization scheme. This scheme is similar to the HTTP authorization scheme but doesn’t interfere with it, so using proxy authorization doesn’t block the ability to authenticate to the remote server.

This function is usually used with the **PathCheck fn=require-proxy-auth** function.

Syntax

```
AuthTrans fn=proxy-auth auth-type=basic
    dbm=full path name
```

```
AuthTrans fn=proxy-auth auth-type=basic
    userfile=full path name
    grpfile=full path name
```

Parameters

auth-type specifies the type of authorization to be used. The type should be "basic" unless you are running a UNIX proxy and are going to use your own function to perform authentication.

dbm specifies the full path and base filename of the user database in the server's native format. The native format is a system DBM file, which is a hashed file format allowing instantaneous access to billions of users. If you use this parameter, don't use the userfile parameter.

userfile specifies the full pathname of the user database in the NCSA-style httpd user file format. This format consists of name:password lines where password is encrypted. If you use this parameter, don't use dbm.

grpfile (optional) specifies the NCSA-style httpd group file to be used. Each line of a group file consists of group:user1 user2...userN**,** where each user is separated by spaces.

Example

A UNIX example:

```
AuthTrans fn=proxy-auth auth-type=basic
    dbm=/usr/ns-home/proxy-EXAMPLE/userdb/rs
```

A Windows NT example:

```
AuthTrans fn=proxy-auth auth-type=basic
    userfile=\netscape\server\proxy-EXAMPLE\.htpasswd
    grpfile=\netscape\server\proxy-EXAMPLE\.grpfile
```

It is possible to have authentication be performed by a user-provided function by passing the user-fn parameter to the **proxy-auth** function.

Syntax

```
AuthTrans fn=proxy-auth auth-type=basic
    user-fn=your function
    userdb=full path name
```

Parameters

user-fn specifies the name of the user-provided function that will be used to perform authentication in place of the built-in authentication. If authentication succeeds, the function should return REQ-PROCEED and if authentication fails, it should return REQ-NOACTION.

userdb specifies the full path and base filename of the user database in the server's native format. The native format is a system DBM file, which is a hashed file format allowing instantaneous access to billions of users.

Connect

The **Connect** directive calls the connect function you specify.

Syntax

`Connect fn=`*your-connect-function*

Only the first applicable **Connect** function is called, starting from the most restrictive object. Occasionally it is desirable to call multiple functions (until a connection is established). The function returns REQ_NOACTION if the next function should be called. If it fails to connect, the return value is REQ_ABORT. If it connects successfully, the connected socket descriptor will be returned.

The **Connect** function must have this prototype:

```
int your_connect_function(pblock *pb, Session *sn, Request *rq);
```

Connect gets its destination host name and port number from:

```
rq->host (char *)
rq->port (int)
```

The host can be in a numeric IP address format.

To use the NSAPI custom DNS class functions to resolve the host name, make a call to this function:

```
struct hostent *servact_gethostbyname(char *host name, Session *sn,
Request *rq);
```

Example

This example uses the native connect mechanism to establish the connection:

```
#include "base/session.h"
#include "frame/req.h"
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
int my_connect_func(pblock *pb, Session *sn, Request *rq)
{

   struct sockaddr_in sa;
   int sd;
   memset(&sa, 0, sizeof(sa));
   sa.sin_family = AF_INET;
   sa.sin_port   = htons(rq->port);
   /* host name resolution */
   if (isdigit(*rq->host))
      sa.sin_addr.s_addr = inet_addr(rq->host);
   else
   {
      struct hostent *hp = servact_gethostbyname(rq->host, sn, rq);
      if (!hp)
         return REQ_ABORTED; /* can’t resolv */
      memcpy(&sa.sin_addr, hp->h_addr, hp->h_lenght);
   }
   /* create the socket and connect */
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
   if (sd == -1)
      return REQ_ABORTED; /* can’t create socket */
   if (connect(sd, (struct sockaddr *)&sa, sizeof(sa)) == -1) {
      close(sd);
         return REQ_ABORTED; /* can’t connect */
   }
   return sd;                /* ok */
}
```

DNS

The **DNS** directive calls either the **dns-config** built-in function or a DNS function that you specify.

dns-config (suggest treating certain host names as remote)

Syntax

```
DNS fn=dns-config local-domain-levels=<n>
```

local-domain-levels specifies the number of levels of subdomains that the local network has. The default is 1.

iPlanet Web Proxy Server optimizes DNS lookups by reducing the times of trying to resolve hosts that are apparently fully qualified domain names but which DNS would otherwise by default still try to resolve relative to the local domain.

For example, suppose you're in the netscape.com domain, and you try to access the host www.xyzzy.com. At first, DNS will try to resolve:

```
www.xyzzy.com.netscape.com
```

and only after that the real fully-qualified domain name:

```
www.xyzzy.com
```

If the local domain has subdomains, such as `corp.netscape.com`, it would do the two additional lookups:

```
www.xyzzy.com.corp.netscape.com
www.xyzzy.com.netscape.com
```

To avoid these extra DNS lookups, you can suggest to the proxy that it treat host names that are apparently not local as remote, and it should tell DNS immediately not to try to resolve the name relative to the current domain.

If the local network has no subdomains, you set the value to 0. This means that only if the host name has no domain part at all (no dots in the host name) will it be resolved relative to the local domain. Otherwise, DNS should always resolve it as an absolute, fully qualified domain name.

If the local network has one level of subdomains, you set the value to 1. This means that host names that include two or more dots will be treated as fully qualified domain names, and so on.

An example of one level of subdomains would be the netscape.com domain, with subdomains:

```
corp.netscape.com
engr.netscape.com
mktg.netscape.com
```

This means that hosts without a dot, such as step would be resolved with respect to the current domain, such as engr.netscape.com, and so the **dns-config** function would try this:

```
step.engr.netscape.com
```

If you are on corp.netscape.com but the destination host `step` is on the `engr` subdomain, you could say just:

```
step.engr
```

instead of having to specify the fully qualified domain name:

```
step.engr.netscape.com
```

your-dns-function (a plug-in dns function you create)

This is a DNS-class function that you define.

Syntax

DNS fn=*your-dns-function*

Only the first applicable DNS function is called, starting from the most restrictive object. In the rare case that it is desirable to call multiple DNS functions, the function can return REQ_NOACTION.

The DNS function must have this prototype:

```
int your_dns_function(pblock *pb, Session *sn, Request *rq);
```

The DNS function looks for its parameter host name from:

```
rq->host (char *)
```

and it should place the resolved result into:

```
rq->hp (struct hostent *)
```

The struct hostent * will not be freed by the caller but will be treated as a pointer to a static area, as with the gethostbyname call. It is a good idea to keep a pointer in a static variable in the custom DNS function and on the next call either use the same struct hostent or free it before allocating a new one.

The DNS function returns REQ_PROCEED if it is successful, and REQ_NOACTION if the next DNS function (or gethostbyname, if no other applicable DNS class functions exist) should be called instead. Any other return value is treated as failure to resolve the host name.

Example

This example uses the normal gethostbyname call to resolve the host name:

```
#include "base/session.h"
#include "frame/req.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
int my_dns_func(pblock *pb, Session *sn, Request *rq)
{
    rq->hp = gethostbyname(rq->host);
        if (rq->hp)
            return REQ_PROCEED;
        else
            return REQ_ABORTED;
}
```

Error

At any time during a request, conditions can occur that cause the server to stop fulfilling a request and to return an error to the client. When this happens, the server can send a short HTML page to the client, very generically describing the error.

To help you make error handling more user friendly, the proxy server lets you intercept certain errors and send a file with your customized error message in place of the server's default error message. You can create an HTML file containing the error message you want to send and associate that message with an error.

Syntax

`Error fn=send-error code=`*code* `path=`*path*

code is the error code for the default error message, as listed below.

path is the full path to the HTML file containing the message you want to send.

The following are errors returned by the server. Each error has a three-digit HTTP code that designates it, followed by a short description of the error. The description might help you write your custom error message, in some of cases below:

- **401 Unauthorized** (for administration forms only). The server requires HTTP user authorization to allow access to the administration forms, and either the client provided none or its HTTP authorization was insufficient. You can customize this error message only when you use Sun ONE Web Proxy Server as a reverse proxy.

- **403 Forbidden**. The server tried to access a file or directory and found that the user it was running as didn't have sufficient permission to access the file. You can customize this error message.

- **404 Not Found**. The client asked for a file system path that doesn't exist or the server was configured to tell the client that it doesn't exist. Since this message is not generated by proxy server, it cannot be customized.

- **407 Proxy Authorization Required**. The proxy requires proxy authorization, and either the client didn't provide any or it was insufficient. Also, the client software might not support proxy authorization. Netscape Navigator version 1.1 or newer supports this authorization. This error message can be customized.
- **500 Server Error**. Server errors mean that an error has occurred in the server that prevents it from finishing the request. Server errors mainly happen because of misconfiguration or machine resources such as swap space being exhausted. Since this message is not generated by proxy server, it cannot be customized.

Example

```
Error fn=send-error Code=403
path=/usr/ns-home/proxy-EXAMPLE/errors/403.html
```

Filter

The **Filter** directive runs an external command and then pipes the data through the external command before processing that data in the proxy. This is accomplished using the **pre-filter** function. The format of the **Filter** directive is as follows:

Syntax

`Filter fn="pre-filter" path="`*/your/filter/prog*`"`

The **Filter** directive performs these tasks:

5. It runs the program **/your/filter/prog** as a separate process.
6. It establishes pipes between the proxy and the external program.
7. It writes the response data from the remote server to the stdin of the external program.
8. It reads the stdout of the program as if it were the response generated by the server.

This is equivalent to this command:

```
Filter fn="pre-filter"
   path="/your/filter/prog"
   headers="stdin"
```

Init

Init is a special directive that initializes certain proxy subsystems such as the networking library, caching module, and access logging. The functions referenced with the **Init** directive load data for specific subsystems once on server startup and keep that data internally until the server is shut down.

Init lines can contain spaces at the beginning of the line and between the directive and value, but you shouldn't have spaces after the value because they might confuse the server. Long lines (although probably not necessary) can be continued with a backslash (\) continuation character before the line feed.

CAUTION If you are using iPlanet Web Proxy Server Manager online forms, you shouldn't use continuation lines in the `obj.conf` file. Instead, put each Init configuration entirely on a single line. If you are absolutely sure you will never use the online forms to edit the `obj.conf` file, you can use the \ character.

Syntax

```
Init fn=function-name [parm1=value1]...[parmN=valueN]
```

function-name identifies the server initialization function to call. These functions shouldn't be called more than once.

parm=value pairs are values for function-specific parameters. The number of parameters depends on the function you use. The order of the parameters doesn't matter. The functions of the **Init** directive listed here are described in detail in the following sections.

- **flex-init** initializes the flex-log flexible access logging feature
- **icp-init** initializes the ICP feature.
- **init-batch-update** initializes the batch update feature.
- **init-cache** enables and initializes caching.
- **init-clf** initializes the Common Log File subsystem.
- **init-dns-cache** enables and initializes dns caching.
- **init-partition** initializes the partitions.
- **init-proxy** initializes the networking code used by the proxy.
- **init-proxy-auth** tells proxy how to authenticate itself.

- **init-proxy-certs** loads a default certificate database on startup.
- **init-sockd** enables and initializes the SOCKD feature.
- **init-urldb** initializes the URL database that you specify.
- **load-modules** tells the server to load functions from a shared object file.
- **load-types** maps file extensions to MIME types.

Init function order in obj.conf

The **Init** functions are a series of steps that the server has to follow in order for the proxy to run. Each function depends on the results of the one before it:

1. Start the proxy.
2. Start the proxy's cache.
3. Initialize the partitions inside the cache.
4. Initialize the batch update process (which might be updating what's inside the partitions).

NOTE In `obj.conf`, the order of certain **init-** functions is crucial. These functions must occur in the order shown here:

```
Init fn=init-proxy ...
Init fn=init-cache ...
Init fn=init-partition ...
Init fn=init-batch-update ...
```

Calling Init functions

Some functions of the **Init** directive are crucial to proxy functioning and must be called once and only once. Others are optional but must be called no more than once, and some are optional and can be called many times. They are shown in Table C-1.

Table C-1 Calling Functions of the Init Directive

Function	Crucial, call just once	Optional, call just once	Optional, call many times
flex-init		X	X
icp-init		X	

Table C-1 Calling Functions of the Init Directive

Function	Crucial, call just once	Optional, call just once	Optional, call many times
init-batch-update		X	
init-cache		X	
init-clf	X		
init-dns-cache		X	
init-partition			X
init-proxy	X		
init-proxy-auth		X	
init-proxy-certs		X	
init-sockd		X	
load-modules			X
load-types	X		
pa-init-parent-array		X	
pa-init-proxy-array		X	

flex-init (starting the flex-log access logs)

The **flex-init** function initializes the flexible logging system. It opens the log file whose name is passed as a parameter and establishes a record format that is passed as another parameter. The log file stays open until the server is shut down, or for UNIX proxies, until the base server process is sent the **-**HUP signal (at which time all logs are closed and reopened).

You use **flex-init** to specify a log filename (such as `loghttp=/var/ns-server/loghttp`); then you use that name with the **flex-log** function in the `obj.conf` file to add a log entry to the file (such as `AddLog fn=flex-log name=loghttp`).

NOTE You can use **AddLog** to store transactions in more than one log file.

If you move, remove, or change the log file without shutting down or restarting the server, client accesses might not be recorded. To save or back up a log file on a UNIX proxy, you need to rename the file and then send the -HUP signal to restart the server. The server uses the inode number, but when you do a soft restart, the server first looks for the filename, and if it doesn't find the log file, it creates a new one (the renamed original log file is left for you to use).

Parameters

The **flex-init** function recognizes two possible parameters: one that names the log file and one that specifies the components of a record in that file.

The **flex-init** function recognizes anything contained between percent signs (%) as the name portion of a name-value pair stored in a parameter block in your program. (The one exception to this rule is the %SYSDATE% component, which delivers the current system date.)

Any additional text is treated as literal text, so you can add to the line to make it more readable. Typical components of the formatting parameter are listed in Table C-1. Certain components might contain spaces, so they should be bounded by escaped quotes (/").

Table C-2 Options for flex-logging

Flex-log option	Component	Escaped
Client host name	%Ses->client.ip%	
Authenticate user name	%Req->vars.auth-user%	
System date	%SYSDATE%	
Full request	/"%Req->reqpb.proxy-request%/"	Yes
Status	%Req->srvhdrs.clf-status%	
Content length	%Req->vars.p2c-cl%	
Referer	/"%Req->headers.referer%/"	Yes
User-agent	/"%Req->headers.user-agent%/"	Yes
Method	%Req->reqpb.method%	
URI	%Req->reqpb.uri%	
Query string of the URI	%Req->reqpb.query%	
Protocol	/"%Req->reqpb.protocol%/"	Yes
Accept header	%Req->headers.accept%	
Date header	/"%Req->headers.date%/"	Yes

Table C-2 Options for flex-logging

Flex-log option	Component	Escaped
“If Modified Since” header	%Req->headers.if-modified-since%	
Authorization	%Req->headers.authorization%	
Cache finish status	%Req->vars.cch-status%	
Remote server finish status	%Req->vars.svr-status%	
Client connection finish status	%Req->vars.cli-status%	
Status code from server	%Req->vars.remote-status%	
Route to proxy	%Req->vars.actual-route%	
Transfer time in seconds	%Req->vars.xfer-time%	
Transfer time in milliseconds	%Req->vars.xfer-time-total%	
DNS time	%Req->vars.xfer-time-dns%	
Connect wait time	%Req->vars.xfer-time-cwait%	
Initial wait time	%Req->vars.xfer-time-iwait%	
Full wait time	%Req->vars.xfer-time-fwait%	
Header-length from server response	%Req->vars.r2p-hl%	
Request header size from proxy to server	%Req->vars.p2r-hl%	
Response header size sent to client	%Req->vars.p2c-hl%	
Request header size received from client	%Req->vars.c2p-hl%	
Content-length from proxy to server request	%Req->vars.p2r-cl%	
Content-length received from client	%Req->headers.content-length%	
Content-length from server response	%Req->vars.r2p-cl%	
Unverified user from client	%Req->vars.unverified-user%	

Example

This example for a UNIX proxy server initializes flexible logging in to the file `/usr/ns-home/proxy-NOTES/logs/access`:

```
Init fn=flex-init access="/usr/ns-home/https-NOTES/logs/access" format.access=
"%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
/"%Req->reqpb.proxy-request%/"
%Req->srvhdrs.clf-status% %Req->vars.p2c-c1%"
```

This example for a Windows NT proxy server initializes flexible logging in to the file `netscape\server\proxy-NOTES\logs\access:`

```
Init fn=flex-init access="\" format.access=
"%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]
/"%Req->reqpb.proxy-request%/"
%Req->srvhdrs.clf-status% %Req->vars-p2c-c1%"
```

This will log the following items:

1. IP or host name, followed by the three characters " - "
2. the user name, followed by the two characters " ["
3. the system date, followed by the two characters "] "
4. the full request, followed by a single space
5. the full status, followed by a single space
6. the content length

This is the default format, which corresponds to the Common Log Format (CLF).

The first six elements of any log should always be in *exactly* this format, because a number of log analyzers expect that as output.

icp-init (initializes ICP)

The **icp-init** function enables and initializes ICP. ICP (Internet Cache Protocol) is an object location protocol that enables caches to communicate with one another. Caches can use ICP to send queries and replies about the existence of cached URLs and about the best locations from which to retrieve those URLs.

Syntax

```
Init fn="icp.init"
   config_file="file name"
   status="on|off"
```

Parameters

config_file is the name of the ICP configuration file.

status specifies whether ICP is enabled or disabled. Possible values are:

- **on** means that ICP is enabled.
- **off** means that ICP is disabled.

Example

```
Init fn="icp.init"
   config_file="icp.conf"
   status="on"
```

init-batch-update (starting batch updates)

The **init-batch-update** function starts and specifies a configuration file for batch updating. Batch updating (or similarly, autoloading) is the process of loading frequently requested objects into the proxy cache in anticipation of those requests.

Batch updating is useful for a number of tasks. A proxy administrator might want to perform up-to-date checks during low-usage hours on all the cached objects to avoid doing these checks when usage is heavier. If a site has heavy daytime usage but little in the evening, the batch update could run in the evening. The process can converse with remote servers and update any objects that have been modified.

At larger sites with a network of servers and proxies, you, as the administrator, might want to use autoloading to "inhale" (pre-load into the cache) a given area of the web. You provide an initial URL, and the batch process does a recursive ("worm") descent across links in the document. Because this function can be a burden on remote servers, be careful when using it. Measures are taken to keep the process from performing recursion indefinitely, and the parameters in `bu.conf` give you some control of this process. You could also use this functionality to update proxies to compensate for any unexpected changes in a company-wide directory index.

Syntax

```
Init fn=init-batch-update
   status=on|off
   conf-file="absolute filename"
```

Parameters

status enables or disables batch updating.

- **on** means batch updating will be started, and the update function expects to find a configuration file (otherwise it will abort).

- **off** means no batch updating or autoloading activity will occur.

conf-file is the pathname to the batch update (autoload) configuration file.

Example

A UNIX example:

```
Init fn=init-batch-update
   status=on
   conf-file="/usr/ns-home/java-proxy/config/bu.conf"
```

A Windows NT example:

```
Init fn=init-batch-update
   status=on
   conf-file=""
```

init-cache (starting the caching system)

The **init-cache** function enables and initializes the cache and starts the caching system. Calling this function is crucial if you want to use caching; otherwise it is optional and must be called only once.

Syntax

For UNIX version:

```
Init fn=init-cache
   status=on|off
   dir=directory
   ndirs=n
   cmon=on|off
```

Parameters

status enables or disables caching.

- on enables caching
- `off` turns caching off

dir specifies a working directory for the proxy's caching module. This is *not* necessarily the cache directory. To specify directories where cached data is actually stored, use the **init-partition** function.

cmon enables or disables monitoring of the cache size.

- on enables monitoring and is the default
- `off` disables monitoring

Setting **cmon** to **off** disables the internal Cache Monitor and Cache Manager processes that the ns-proxy process automatically initiates when caching is enabled. The Cache Monitor process monitors the cache size and when necessary sends a message to the Cache Manager process to trigger garbage collection (cleaning up the cache to allow more space for new cache files). Disabling these internal cache management processes lets you use an external custom program to perform cache management tasks.

Further, setting cmon to off makes it possible for two separate proxy pools to share the same cache. In this situation, only one of the proxies can have cache management enabled, and the other must have it disabled.

ndirs specifies cache capacity as the number of cache sections you want set up for the proxy. The number you specify must be a power of 2, from 1 to 256. This number is related to cache capacity as shown in Table C-3. To determine the number to use, see the capacity entry in Table C-3 for the *optimum* designed capacity of your machine.

Table C-3 shows the optimum designed capacity, the minimum suggested capacity, and the maximum suggested capacity for each valid number of sections. The optimum capacity for a specific ndir matches the minimum capacity for the next larger ndir and also matches the maximum capacity for the next smaller ndir.

For a small cache size you can set a small cache capacity, although an oversized cache structure is not a problem. If the cache is actually larger than the maximum capacity for the number of sections you specify, its performance could be diminished because of an undersized cache structure, that is, there may not be enough sections.

Table C-3 Cache capacity and the number of sections

Optimum	Minimum	Maximum	Number of sections
125MB	0MB	250MB	1
250MB	125MB	500MB	2
500MB	250MB	1GB	4
1GB	500MB	2GB	8
2GB	1GB	4GB	16
4GB	2GB	8GB	32
8GB	4GB	16GB	64
16GB	8GB	32GB	128
32GB	16GB	64GB	256

Example

```
Init fn=init-cache
   status=on
   dir=/usr/ns-home/cache
   ndirs=8
```

init-dns-cache (starting dns caching)

The init-dns-cache function specifies (when DNS lookups are enabled through a Server Manager setting) that you want to cache the DNS entries. If you cache DNS entries, then when the server gets a client's host name information, it can store the data it receives. Then, if the server needs information about the client in the future, the information is available to the server without querying for the information again.

You can specify the size of the DNS cache and the time it takes before a cache entry becomes invalid. The DNS cache can contain 32 to 32768 entries; the default value is 1024 entries. Values for the time it takes for a cache entry to expire can range from 1 second to 1 year (specified in seconds); the default value is 1200 seconds (20 minutes).

Syntax

```
Init fn=init-dns-cache
   cache-size=entries
   expire=seconds
   visible=yes|no
```

Parameters

The init-dns-cache function takes two arguments, cache-size and expire.

cache-size specifies how many entries are contained in the cache. Acceptable values for cache-size are 32 to 32768; the default value is 1024.

expire specifies how long it takes for a cache entry to expire. Acceptable values (specified in seconds) for expire are 1 second to 1 year; the default is 1200 seconds (20 minutes).

visible specifies whether the DNS cache file is visible. Possible values include:

- **yes** means the DNS cache file is visible.
- **no** means the DNS cache file is invisible. This is the default.

Example

```
Init fn="init-dns-cache"
   cache-size="2140"
   expire="600"
   visible=yes
```

init-clf (starting the Common Log File subsystem)

The **init-clf** function initializes the Common Log File subsystem. Use this function to specify which log files the proxy uses to record transactions. Then, use the **AddLog** directive to specify the log file where the proxy stores the transaction record. For more information on the **AddLog** directive, see page 413.

NOTE You can use the **AddLog** directive to store transactions in more than one log file.

The **init-clf** function opens the log files that you specify. The log files stay open until the server is shut down or, for a UNIX proxy server, until the base server process is sent the -HUP signal (at which time the logs are closed and re-opened). Initializing this function is required if you are using the log features.

CAUTION If you move, remove, or change the log file without shutting down or restarting the server, client accesses might not be recorded. To save or back up a log file on a UNIX proxy, you need to rename the file and then send the -HUP signal to restart the server. The proxy uses the *inode* number, but when you do a soft restart, the proxy first looks for the filename, and if it doesn't find the log file, it creates a new one (the renamed original log file remains for you to use).

Syntax

```
Init fn=init-clf
   global=main log filename
```

Parameters

global tells the proxy to use this log file by default. This function doesn't have any predefined parameters, just name and pathname pairs in the format `<name>=<pathname>` where:

- **name** is a symbolic name that you give to a log file pathname.
- **pathname** is the directory and filename for the log file.

Example

```
Init fn=init-clf
   global=\netscape\server\proxy-TEST\logs\access
```

To open three separate log files, you could use **name** and **pathname** pairs like this:

```
Init fn=init-clf
   log1=/usr/ns-home/logs/log-one
   log2=/usr/ns-home/logs/log-two
   log3=/usr/ns-home/logs/log-three
```

Later, you can refer to symbolic names log1, log2, and log3 from the **AddLog fn=proxy-log** function. For details, see “AddLog” on page 413.

init-partition (specifying cache partitions)

The **init-partition** function specifies the status, location, and size of cache partitions.

Syntax

```
Init fn=init-partition
   status=on|off
   dir=directory
   max-size=megabytes
   min-avail=megabytes
```

Parameters

status enables or disables the cache partition.

- **on** means that the cache partition is active (in normal use, data is written to and read from that partition).
- **off** means that the cache partition is not in use (cache data mapping to that partition will not be written to or looked up from that partition).

dir is the directory to use as the cache. The cache structure (that is, sections) must exist under that directory (which can be created and maintained through the online forms).

max-size is the optional number for the maximum size, in megabytes, to allow for the cache partition to grow. You can choose not to set size limits by leaving this parameter out.

min-avail is the minimum amount of available space, in megabytes, on the physical partition. This is the actual disk on which the cache partition (defined by the dir parameter) resides. If less space is ever available, the proxy stops caching to that cache partition, even if it hasn't reached the maximum size (max-size). It continues to write to other partitions that are not full. It also notifies the Cache Monitor that the partition is filling up, and the Cache Manager consequently starts to clean up the cache. You can choose not to set the minimum by leaving this parameter out.

Example

```
Init fn=init-partition
    status=on
    dir=/usr/ns-home/cache
    max-size=2000
    min-avail=5
```

init-proxy (starting the network software for proxy)

The **init-proxy** function initializes the networking software used by the proxy. Calling this function in `obj.conf` is crucial (even though it is called automatically, you should call it manually as a safety measure).

Syntax

For UNIX version:

```
Init fn=init-proxy
    timeout=<seconds>
    timeout-2=seconds
    read-timeout=seconds
    keep-alive-timeout=seconds
    stall-timeout-override=seconds
    netlib-timeout=seconds
    sig="Some readable name"
    anon-pw="e-mail address"
    socks-ns="IP address"
```

Parameters

timeout is the number of seconds of delay allowed between consecutive network packets received from the remote server. If the delay exceeds the timeout, the connection is dropped. The default is 120 seconds (2 minutes). This is *not* the maximum time allowed for an entire transaction, but the delay between the packets. For example, the entire transaction can last 15 minutes, as long as at least one packet of data is received before each timeout period.

timeout-2 tells the proxy how much time it has to continue writing a cache file after a client has aborted the transaction. In other words, if the proxy server has almost finished caching a document and the client aborts the connection, the proxy can continue caching the document until it reaches the timeout after interrupt value.

To determine the best timeout-2 (timeout after interrupt) value for your proxy server, use sitemon. If sitemon reports that the proxy server is using too much of its process pool, you should reduce your timeout after interrupt accordingly. The highest recommended timeout after interrupt value is 5 minutes.

read-timeout is the number of seconds the proxy server will wait for an incoming request. The default value for this timeout is 60 seconds. Setting the timeout a few seconds shorter will allow proxy resources to be released faster if the connection stays idle (e.g., if you telnet to the proxy port and let it just sit there).

keep-alive-timeout is the number of seconds the proxy server will wait for the next request from a keep-alive connection. The default value for this timeout is 5 seconds. Shorter timeouts let the proxy release resources that are tied up by idle keep-alive connections. Longer timeouts will make keep-alive more effective, but can tie up valuable proxy resources, and thus, bog down the server. You should not use the keep-alive feature because it can easily cause problems when the entire process pool is tied up by idle keep-alive connections. This problem can occur quickly - especially with clients that open multiple simultaneous connections, such as Navigator, which by default, opens 4 connections.

stall-timeout-override should not be changed.

netlib-timeout is the absolute maximum number of seconds that the proxy will wait for any HTTP, FTP, Gopher or HTTPS retrieval. The default value for this timeout is 10800 seconds (3 hours).

sig is the signature (trailer) that the proxy appends to its error messages. By default, it contains the proxy host name and the port number. If your site does not want to send that information out or perhaps gives more descriptive names to proxies, you can use **sig** to do that.

anon-pw is the email address to send to anonymous FTP servers as the password. This information can be used by FTP sites to later send notifications to people who downloaded files from their FTP site. Using this option overrides the default that the proxy will derive from its current execution environment (which could be, for example, “nobody@your.site”).

socks-ns is equivalent to using the SOCKS_NS environment variable in the proxy’s execution environment. This tells the proxy the DNS server IP address instead of having the proxy try to derive the default from its environment. This option is useful when the internal DNS server is not capable of resolving external DNS names. It is often necessary when routing the proxy through a SOCKS server.

Example

```
Init fn=init-proxy
   log-format=extended-2
   timeout=120
   sig="Main proxy gateway"
   anon-pw="webmaster@your.site"
   socks-ns="123.1.2.3"
```

init-proxy-auth (specifying the authentication strategy)

The **init-proxy-auth** function tells the proxy server whether it should require authentication from clients as a proxy, or reverse proxy (web server). If the `obj.conf` file does not call this function, the server will automatically act as a proxy requiring authentication.

Syntax

```
Init fn=init-proxy-auth
   pac-auth=on|off
```

Parameters

pac-auth specifies whether local files (PAC files, local icons, etc.) are password-protected.

- **on** means that local files are password-protected and require authentication. This setting has no effect if access control is not enabled for your proxy server. If you set **pac-auth** to yes, and proxy authentication is enabled, users will be prompted for their password twice.
- **off** means that local files do not require authentication.

Example

```
Init fn=init-proxy-auth
   pac-auth=yes
```

init-proxy-certs (loading the default certificate database)

The **init-proxy-certs** function points the proxy to the default certificate database and loads that file at startup.

Syntax

When you make changes to the certificate database and Initialize Certificates Only is disabled, this line is automatically inserted into the `obj.conf` file upon restart:

```
Init fn=init-proxy-certs
    certfile=absolute filename
```

If the proxy is set up to authenticate itself to remote servers by using its certificate, and Initialize Certificates Only is enabled, this line is automatically inserted into the `obj.conf` file upon restart:

```
Init fn=init-proxy-certs
    security=on|off
    keyfile=absolute filename
```

Parameters

certfile is the absolute filename of the default certificate database.

security enables or disables encryption for the proxy.

- **on** means that encryption is enabled.
- **off** means that encryption is disabled.

keyfile is the absolute filename of the private keyfile for the proxy.

Example

```
Init fn=init-proxy-certs
    certfile="/usr/ns-home/proxy-java-proxy/config/ServerCert"

Init fn=init-proxy-certs
    security=on
    keyfile="/usr/ns-home/proxy-java-proxy/config/ServerKey.db"
```

init-sockd (starting the SOCKD feature)

The **init-sockd** function enables and initializes iPlanet Web Proxy Server's SOCKD feature. It makes the proxy behave like a SOCKS daemon in addition to its normal tasks.

The SOCKD configuration is done through the normal SOCKD configuration file, `/etc/socks5.conf`. The same care must be taken as when configuring any other SOCKS daemon.

Syntax

```
Init fn=init-sockd
    status=on|off
    sockd-conf=absolute filename
    ident-check=none|loose|strict
    log-type=common-separate|syslog|syslogseparate
    log-name=absolute filename
```

Parameters

status enables or disables the **sockd** function.

- **on** means the SOCKD feature is enabled; that is, the proxy will act as a SOCKS server.
- **off** means the SOCKD feature is disabled; same as if the function is left out. SOCKS requests will not be answered by the proxy (an error will be returned to the client).

sockd-conf (optional) specifies the absolute pathname for the SOCKD configuration file to use. The default is `/etc/sockd.conf`. The format follows the standard SOCKD configuration file format.

ident-check (optional) specifies the type of remote identity checking. The possible values are:

- **strict** means an identity check is required, and if the remote host is not running **identd**, ("Identity Daemon") the request is denied. The strict identity check is like the "loose" identity check, except that if the remote identity query fails, access will be denied.
- **loose** means an identity check is required, but if the remote host is not running **identd**, access is granted anyway. Remote identity will be queried from the remote **identd**, and the given user name will be verified against the one returned by the remote **identd**. If there's a mismatch, access will be denied. If the remote identity query fails (the remote server's not running **identd**), access will be granted.
- **none** means no remote identity check is performed. The remote user name will not be queried from the remote **identd**. The user name given in the SOCKS request will not be verified.

log-type (optional) specifies where the proxy SOCKD module stores the access information. The directive can be one of the following:

- **common-separate** uses the common log format, but puts the entries in a separate log file (the log filename is given in the **log-name** parameter).

- **syslog** uses the traditional SOCKD format and reports to the syslog facility.
- **syslog-separate** uses the SOCKD syslog format, but instead of syslog, it uses a separate log file as specified by the **log-name** parameter.

log-name specifies the absolute pathname for a separate log file used for logging SOCKS accesses. This name is required when the **log-type** is either common-separate or syslog-separate.

Example

```
Init fn=init-sockd
    status=on
    sockd-conf=/etc/sockd.conf
    ident-check=strict
    log-type=syslog
    log-name=/d/proxy/logs/sockd.log
```

init-urldb (setting up the URL database)

The **init-urldb** function initializes the URL database and it specifies whether to cache URLs. It also identifies the directory where URLs will be contained if they are cached.

Syntax

```
Init fn=init-urldb
    status=on|off
    dir=absolute filename
```

Parameters

status is whether the URL database is enabled or disabled.

- **on** means that URL recording to the URL database is enabled. This database is the one that can be browsed from the administration interface to find out what's actually cached.

  To log the URLs, the **AddLog fn=urldb-record** function has to be called for each object ("<Object>") for which URL recording is desired. URLs for documents that don't get cached will never be recorded in this database.
- **off** means that URL recording is disabled. URLs will not be recorded even if the **AddLog fn=urldb-record** function is called.

dir is the name of the directory containing the URL database. Don't use `/tmp` because it is cleared at boot time.

Example

```
Init fn=init-urldb
   status=on
   dir=/usr/ns-home/cache/urldb
```

load-modules (loading shared object modules)

You can use the **load-modules** function to tell the server to load the functions you need from the shared object. Calling the **load-modules** function is crucial to proxy function.

UNIX allows shared libraries, which are archives of multiple functions packed into a single file (with a `.so` suffix). If you want to link in functions from shared libraries you have created, use this function to pass required information to the server. To do this, you have to tell the main executable where the shared library file resides and the names of the functions to be loaded (which are indexed by name in the `.so` file).

Binaries referring to functions in the shared libraries you specify dynamically load the individual functions at runtime (without loading the entire library).

To register SAF classes with the server you could use this:

```
Init fn=load-modules shlib=/your/lib.so funcs=alpha,beta,alpha-beta
```

where alpha, beta, and alpha-beta represent the functions alpha(), beta(), and alpha_beta() from the shared library `/your/lib.so`. Note the correlation between hyphens and underscores (where the configuration files use hyphens, C code uses underscores).

You can call those functions as you normally would; for example, to call the C function **alpha_beta()** you would use:

```
   Connect fn=alpha-beta
```

Syntax

```
Init fn=load-modules
   shlib=[path]filename.so
   funcs="function1, function2, ..., functionN"
```

Parameters

shlib is the full path and filename of the shared object library containing the functions of interest.

funcs is a list of functions in the shared library to be dynamically loaded.

Example

A UNIX example:

```
Init fn=load-modules
   shlib=/u/myfolder/func.so
   funcs="func1, func2"
```

A Windows NT example:

```
Init fn=load-modules
   shlib=funcs="func1, func2"
```

load-types (loading MIME-type mappings)

The **load-types** function scans a file that tells it how to map filename extensions to MIME types. MIME types are essential for network navigation software like Netscape Navigator to tell the difference between file types. For example, they are used to tell an HTML file from a GIF file. See “The mime.types File,” on page 261 for more information.

Calling this function is crucial if you use iPlanet Web Proxy Server Manager online forms or the FTP proxying capability.

Syntax

```
Init fn=load-types
   mime-types="mime.types"
```

This function loads the MIME type file `mime.types` from the configuration directory (the same directory as `magnus.conf` and `obj.conf`). This function call is mandatory and in practice is always as shown in the syntax.

Parameters

mime-types specifies either the full path to the global MIME types file or a filename relative to the server configuration directory. The proxy server comes with a default file called `mime.types`.

local-types is an optional parameter to a file with the same format as the global MIME types file, but it is used to maintain types that are applicable only to your server.

Example

```
Init fn=load-types mime-types=mime.types
```

```
Init fn=load-types mime-types=/tp/mime.types \
   local-types=local.types
```

pa-init-parent-array (initializing a parent array member)

The **pa-init-parent-array** function initializes a parent array member and specifies information about the PAT file for the parent array of which it is a member.

NOTE The **load modules** directive should come before the **pa-init-proxy-array** function in the `obj.conf` file.

Syntax

```
Init fn=pa-init-parent-array
    set-status-fn=pa-set-member-status
    poll="yes|no"
    file="absolute filename"
    pollhost="host name"
    pollport="port number"
    pollhdrs="absolute filename"
    pollurl="url"
    status="on|off"
```

Parameters

set-status-fn specifies the function that sets the status for the member.

poll tells the array member whether or not it needs to poll for a PAT file.

- **yes** means that the member should poll for the PAT file. A member should only poll for a PAT file if it is not the master proxy. The master proxy has a local copy of the PAT file, and therefore, does not need to poll for it.
- **no** means that the member should not poll for the PAT file. A member should not poll for the PAT file if it is the master proxy.

file is the full pathname of the PAT file.

pollhost is the host name of the proxy to be polled for the PAT file. This parameter only needs to be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

pollport is the port number on the pollhost that should be contacted when polling for the PAT file. This parameter only needs to be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

pollhdrs is the full pathname of the file that contains any special headers that must be sent with the HTTP request for the PAT file. This parameter is optional and should only be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

pollurl is the URL of the PAT file to be polled for. This parameter only needs to be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

status specifies whether the parent array member is on or off.

- **on** means that the member is on.
- **off** means that the member is off.

Example

The following example tells the member not to poll for the PAT file. This example would apply to a master proxy.

```
Init fn=pa-init-parent-array
   poll="no"
   file="c:/netscape/server/bin/proxy/pa1.pat"
```

The following example specifies that the member should poll for a PAT file. This member is not the master proxy.

```
Init fn=pa-init-parent-array
   poll="yes"
   file="c:/netscape/server/bin/proxy/pa2.pat"
   pollhost="proxy1"
   pollport="8080"
   pollhdrs="c:/netscape/server/proxy-name/parray/pa2.hdr"
   status="on"
   set-status-fn=set-member-status
   pollurl="/pat"
```

pa-init-proxy-array (initializing a proxy array member)

The **pa-init-proxy-array** function initializes a proxy array member and specifies information about the PAT file for the array of which it is a member.

NOTE The **load modules** directive should come before the **pa-init-proxy-array** function in the `obj.conf` file.

Syntax

```
Init fn=pa-init-proxy-array
   set-status-fn=pa-set-member-status
   poll="yes|no"
   file="absolute filename"
   pollhost="host name"
```

```
    pollport="port number"
    pollhdrs="absolute filename"
    pollurl="url"
    status="on|off"
```

Parameters

set-status-fn specifies the function that sets the status for the member.

poll tells the array member whether or not it needs to poll for a PAT file.

- **yes** means that the member should poll for the PAT file. A member should only poll for a PAT file if it is not the master proxy. The master proxy has a local copy of the PAT file, and therefore, does not need to poll for it.
- **no** means that the member should not poll for the PAT file. A member should not poll for the PAT file if it is the master proxy.

file is the full pathname of the PAT file.

pollhost is the host name of the proxy to be polled for the PAT file. This parameter only needs to be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

pollport is the port number on the pollhost that should be contacted when polling for the PAT file. This parameter only needs to be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

pollhdrs is the full pathname of the file that contains any special headers that must be sent with the HTTP request for the PAT file. This parameter is optional and should only be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

pollurl is the URL of the PAT file to be polled for. This parameter only needs to be specified if the **poll** parameter is set to yes, meaning that the member is not the master proxy.

status specifies whether the parent array member is on or off.

- **on** means that the member is on.
- **off** means that the member is off.

Example

The following example tells the member not to poll for the PAT file. This example would apply to a master proxy.

```
Init fn=pa-init-proxy-array
    poll="no"
    file="c:/netscape/server/bin/proxy/pa1.pat"
```

The following example specifies that the member should poll for a PAT file. This member is not the master proxy.

```
Init fn=pa-init-proxy-array
   poll="yes"
   file="c:/netscape/server/bin/proxy/pa2.pat"
   pollhost="proxy1"
   pollport="8080"
   pollhdrs="c:/netscape/server/proxy-name/parray/pa2.hdr"
   status="on"
   set-status-fn=set-member-status
   pollurl="/pat"
```

tune-proxy (tuning server performance)

The **tune-proxy** function allows you to tune the performance of your proxy server.You should not change the default settings unless directed to do so by iPlanet Technical Support.

Syntax

```
Init fn=tune-proxy
   byte-ranges=on|off
   single-accept= (do not change)
   ftp-listing-width=number
```

Parameters

byte-ranges determines whether or not the proxy is allowed to generate byte-range responses from the cache. By default, this feature is disabled. This version of the proxy server supports single byte-ranges only.

NOTE Applications which require multiple byte ranges, such as Adobe Acrobat, may see problems when this feature is enabled. If you are not using such applications, enabling this feature will allow faster truncated document/image retrievals by Navigator clients.
Possible values are:

- **on** means the proxy server is allowed to generate byte-range responses from the cache.
- **off** means the proxy server is not allowed to generate byte-range responses from the cache.

single-accept should not be changed.

ftp-listing-width is the maximum number of characters allowed for filenames in the FTP listing. The default is 80.

Example

```
Init fn=tune-proxy
   byte-ranges=off
   single-accept=(do not change)
   ftp-listing-width=80
```

tune-cache (tuning cache performance)

The **tune-cache** function allows you to tune the performance of your proxy server's cache. You should not change the default settings unless directed to do so by iPlanet Technical Support.

Syntax

```
Init fn=tune-cache
   cch-status=DO-NOT-CACHE|NON-CACHEABLE|NOT-IN-CACHE
   mmap-wr=on|off
   mmap-rdwr=on|off
   shmem-io=on|off
   notify-num-changes=number
   notify-blk-limit=number
   cmon-tick-interval=(do not change)
   mmap-max=kilobytes
   min-sync-interval=seconds
   notify-block-chunk-per-proc=blocks
   cache-fs-full-retry-after=number
   update-after-percent=percentage
   sync-dump-ticks=(do not change)
```

cch-status determines whether to use additional cache status values in the access log. Possible values are:

- DO-NOT-CACHE - pre-determined non-cacheable (by configuration)
- NON-CACHEABLE - post-determined non-cacheable (by response fields)
- NOT-IN-CACHE - with disconnected operation, cache miss

mmap-wr determines whether to use mmap or lseek+write to create the initial CIF header. By default, this value is off. You should not change this value. Possible values are:

- **on** means that mmap and lseek+ will create the initial CIF header.
- **off** means that mmap and lseek+ will not create the initial CIF header.

mmap-rdwr determines whether to use memory mapped I/O when sending data from the cache and updating cache files. By default, this value is on.

- **on** means that memory mapped I/O will be used to send data from the cache or to update cache files.
- **off** means that memory mapped I/O will not be used to send data from the cache or to update cache files.

shmem-io determines whether shared memory is enabled. The child processes communicate with the cache monitor and cache manager via either an interprocess pipe/socket, or shared memory. The shared memory based I/O is faster. By default, shared memory I/O is enabled.

- **on** means that shared memory is enabled.
- **off** means that shared memory is disabled.

notify-num-changes is used only when shared memory I/O is enabled. This parameter determines the number of changes after which the cache monitor is notified about changes made by that child process. More frequent notifications keep the cache monitor up-to-date on the status and size of the cache, but create more work for the cache monitor. The valid range for notify num changes is 1 to 500, and the default value is 10.

notify-blk-limit is used only when shared memory I/O is enabled. This parameter determines the number of blocks cache size usage has changed by a single child process until the cache monitor is notified. Smaller notification limits keep the cache monitor up-to-date on the status and size of the cache, but create more work for the cache monitor. The valid range for notify num changes is 1 to 10000 (0.5K to 5MB), and the default value is 100 (50K).

cmon-tick-interval should not be changed.

mmap-max is the maximum number of kilobytes memory-mapped at once by a single process. Files larger than the max mmap size will be memory-mapped in portions of this size. By default, the max mmap size is 256K.

The max mmap size must be at least 16, and it must be a multiple of page size, which is often four.

min-sync-interval specifies the amount of time the cache manager waits before traversing the entire cache to find out the actual cache size. The cache monitor attempts to maintain current information about the size of each cache section. However, the information may get skewed by unexpected software shutdowns, system crashes, power failures, etc. Another reason for skew is that a server shutdown will lose unnotified changes to the cache in child processes. Because of these skews, the cache manager periodically traverses the entire cache to determine the actual cache size.

The valid range for min-sync-interval is 1800 seconds(1/2 hour) to 604800 seconds (1 week), and the default is 86400 seconds (24 hours).

notify-block-chunk-per-proc controls the threshold which triggers each child process to report their cache usage to the cache monitor. Each server child process notifies the cache monitor about how much new cache space it has taken up (or released) by creating new cache files or updating old ones. To avoid message overflow, these notifications are not sent for every cache operation, but rather each process waits until it has reached a certain threshold before notifying the cache monitor about its activities. The notify-blk-chunk-per-proc parameter controls this threshold. The units are in “blocks”. A block is always 512 bytes (0.5 KB), regardless of the operating system filesystem block size.

The notify-block-chunk-per-proc value is multiplied by the value of the MaxProcs directive. Therefore, the effect of having a higher MaxProcs setting is that child processes will buffer more cache change messages before they notify the cache monitor about them. This effect avoids message overflow on systems with a high load (large MaxProcs), and allows small caches to maintain more accurate size information (small MaxProcs).

cache-fs-full-retry-after controls how many cache write requests should be skipped before the proxy attempts a write operation. When a server child process fails to write to disk and the error code is ENOSPC (No space left on device), the process stops writing data to that cache partition, allowing time for the garbage collector to correct the situation. The fs full retry after variable controls the number of cache write requests that are skipped before the proxy attempts a write operation again.

The valid range for fs full retry after is 1 (retry immediately) to 1024 (practically don’t retry), and the default value is 50.

update-after-percent controls the percentage of used cache space that triggers a partition table update. The cache monitor continuously receives messages from server child processes about how much new cache space they have used (see the parameter, notify-block-chunk-per-proc). To conserve CPU time, the cache monitor does not update its partition table or evaluate garbage collection needs after every such message. Instead, it waits until there is big enough percentage change in size. The update after percent value controls that percentage.

The valid range for update after percent is 0 (every time) to 100 (after size doubles), and the default value is 1.

sync-dump-ticks should not be changed.

Example

```
Init fn=tune-cache
   cch-status=DO-NOT-CACHE
   mmap-wr=off
   mmap-rdwr=on
   shmem-io=off
   notify-num-changes=10
   notify-blk-limit=100
   cmon-tick-interval=(do not change)
   mmap-max=256
   min-sync-interval=86400
   notify-block-chunk-per-proc=67
   cache-fs-full-retry-after=50
   update-after-percent=1
   sync-dump-ticks=(do not change)
```

tune-gc (tuning garbage collector performance)

The **tune-gc** function allows you to tune the performance of your proxy server's garbage collector.You should not change the default settings unless directed to do so by iPlanet Technical Support.

Syntax

```
Init fn=tune-gc
   gc-urldb-interval=seconds
   gc-nap-length=seconds
   hard-gc-nap-count=number
   soft-gc-nap-count=number
   hard-gc-chunk-entries=number
   gc-dir-chunk=number
   gc-hi-margin-percent=percent
   gc-lo-margin-percent=percent
   gc-extra-margin-percent=percent
   gc-leave-fs-full-percent=percent
```

Parameters

gc-urldb-interval controls how often the URL database is cleaned of old URLs and how often it is checked for consistency.

The valid range for the gc URL db interval is 1800 sec (30 minutes) to 604800 (1 week), and the default value is 79200 seconds (22 hours).

gc-nap-length is the amount of time that the garbage collector sleeps during one "nap." The garbage collector takes "naps" every so often so that it does not hog all the CPU from other processes. The frequency of these naps is controlled by two variables: *hard gc nap count* and *soft gc nap count*.

The valid range for gc nap length is 0 (no naps) to 120 (2 minutes), and the default is 1 second

hard-gc-nap-count specifies how many naps hard garbage collector takes during the garbage collection cycle of a single cache section. There are two types of garbage collection: hard and soft. The hard garbage collector traverses the entire cache section and finds all the files individually, picking the ones to delete, and then generates lists of files in the cache ordered by their relative value. These lists are then used by the soft garbage collector to delete files without traversing the entire cache structure.

Hard garbage collection is more resource intensive than soft garbage collection, but it is required every time soft garbage collection runs out of the lists generated by hard garbage collection.

There are 64 subdirectories on each cache section, and naps can be taken between directories only. This means that there can be at most 64 naps. The number of naps should be a power of two.

The valid values for hard gc nap count are: 0 (no sleeps), 1, 2, 4, 8, 16, 32, and 64 (max). The default value is 16 (sleep after every 4 directories).

soft-gc-nap-count specifies how many naps the soft garbage collector takes during the garbage collection cycle. The soft garbage collector simply reads a list of cache files from a set of files produced by the hard garbage collector. It still looks up the file using the stat() system call, to find out if there have been subsequent accesses to the cache file during the last garbage collection, which suggests that the cache file is being (or may be) actively used, and should not be removed.

The valid range for soft gc nap count is 0 (no naps) to 1000 naps, and the default value is 20 naps.

hard-gc-chunk-entries identifies the number of cache entry slots allocated during one pass for garbage collection. One "pass" in garbage collection is defined by the variable *gc dir chunk*, which is the number of subdirectories within a cache section that will be processed in one pass.

This number should be of the same order as the number of files in a typical chunk of directories. As an example, a typical subdirectory in a cache section has 100-200 files. If the gc dir chunk variable is set to 8, then the "hard gc max entries" value should be set to around 800-1600.

A single cache entry is about 32 bytes, so every 1000 entries in this variable means 32KB pool allocation. A valid range for hard gc max entries is 100(3K) to 5000 (160K), and the default value is 500 (16K).

gc-dir-chunk controls the sample size for the LRU algorithm. Basically, this means it controls how many subdirectories under each cache section are processed by the garbage collector in one pass. Because a larger gc dir chunk value causes the proxy to process more files simultaneously, more memory is required. Larger gc dir chunk values also cause the garbage collector to be slower and more CPU intensive. The smaller the gc dir chunk value is, the lighter-weight garbage collection becomes. However, the sample size for the LRU algorithm becomes smaller.

The gc dir chunk value must be a power of two.

Valid values for gc dir chunk are: 1 (single directory), 2, 4, 8, 16, 32, and 64 (all directories at once). The default value is 8 directories.

gc-hi-margin-percent controls the percentage of the maximum cache size that, when reached, triggers garbage collection. This value must be higher than the value for gc lo margin percent.

The valid range for gc hi margin percent is 10 to 100 percent (trigger garbage collection when full). The default value is 80 percent (trigger gc when 80% full).

gc-lo-margin-percent controls the percentage of the maximum cache size that the garbage collector targets. This value must be lower than the value for gc-hi-margin-percent.

The valid range for gc lo margin percent is 5 to 100 percent. The default value is 70 percent (target at 70% full cache after gc).

gc-extra-margin-percent specifies the fraction of the cache the garbage collector will remove. If the garbage collection is triggered by a reason other than the partition's size getting close to the maximum allowed size (see gc-hi-margin-percent), the garbage collector will use the percentage set by the *gc extra margin percent* variable to determine the fraction of the cache to remove.

The valid range for gc extra margin percent is 0 to 100 percent, and the default value is 30 percent (remove 30% of existing cache files).

gc-leave-fs-full-percent determines the percentage of the cache partition size below which garbage collection will not go. This value prevents the garbage collector from removing all files from the cache if some other application is hogging the disk space.

The valid range for gc leave fs full percent is 0 (allow total removal) to 100 percent (remove nothing). The default value is 60 percent (allow the cache size to shrink to 60% of current).

Example

```
Init fn=tune-gc
    gc-urldb-interval=79200
    gc-nap-length=1
    hard-gc-nap-count=16
    soft-gc-nap-count=20
    hard-gc-chunk-entries=500
    gc-dir-chunk=8
    gc-hi-margin-percent=80
    gc-lo-margin-percent=70
    gc-extra-margin-percent=30
    gc-leave-fs-full-percent=60
```

NameTrans

NameTrans is the name translation directive, which maps URLs to mirror sites and to the local file system (for the online forms). **NameTrans** directives should appear in the root object (the “default” object), although you can put them elsewhere. If an object has more than one **NameTrans** directive, the server applies each name translation function until one succeeds and then modifies the URL to either a mirror site URL or to a full file system path.

assign name (associating templates with path)

The **assign-name** function associates the name of a configuration object with a path specified by a regular expression. It always returns REQ_NOACTION.

Syntax

```
NameTrans fn=assign-name
    from=regular expression
    name=named object
```

Parameters

from specifies a pattern, presented as a regular expression,. that specifies a path to be affected.

name is the name of the configuration object to associate with the path.

Example

```
NameTrans fn=assign-name
   name=personnel from=/httpd/docs/pers*
```

map (mapping URLs to mirror sites)

The **map** function of the **NameTrans** directive looks for a certain URL prefix in the URL that the client is requesting. If **map** finds the prefix, it replaces the prefix with the mirror site prefix. When you specify the URL, don't use trailing slashes—they cause "Not Found" errors.

Syntax

```
NameTrans fn=map
   from="site prefix"
   to="site prefix"
   name="named object"
```

Parameters

from is the prefix to be mapped to the mirror site.

to is the mirror site prefix.

name (optional) gives a named object from which to derive the configuration for this mirror site.

Example

```
# Map site http://home.netscape.com/ to mirror site
http://mirror.com
NameTrans fn=map from="http://home.netscape.com"
   to="http://mirror.com"
```

pac-map (mapping URLs to a local file)

The **pac-map** function maps proxy-relative URLs to local files that are delivered to clients who request configuration.

Syntax

```
NameTrans fn=pac-map
   from=URL
   to=prefix
   name=named object
```

Parameters

from is the proxy URL to be mapped.

to is the local file to be mapped to.

name (optional) gives a named object (template) from which to derive configuration.

Example

```
NameTrans fn=pac-map
    from=http://home.netscape.com
    to=index.html
    name=file
```

pat-map (mapping URLs to a local file)

The **pat-map** function maps proxy-relative URLs to local files that are delivered to proxies who request configuration.

Syntax

```
NameTrans fn=pat-map
    from=URL
    to=prefix
    name=named object
```

Parameters

from is the proxy URL to be mapped.

to is the local file to be mapped to.

name (optional) gives a named object (template) from which to derive configuration.

Example

```
NameTrans fn=pat-map
    from=http://home.netscape.com
    to=index.html
    name=file
```

pfx2dir (replacing path prefixes with directory names)

The **pfx2dir** function looks for a directory prefix in the path and replaces the prefix with a real directory name. Don’t use trailing slashes in either the prefix or the directory.

Syntax

```
NameTrans fn=pfx2dir
   from=prefix
   dir=directory
   name=named object
```

Parameters

from is the prefix to be mapped.

dir is the directory that the prefix is mapped to.

name (optional) gives a named object (template) from which to derive configuration for this mirror site.

Example

```
NameTrans fn=pfx2dir
   from=/icons
   dir=c:/netscape/suitespot/ns-icons
```

ObjectType

The **ObjectType** directives tag additional information to the requests, such as caching information and whether another proxy should be used.

If there is more than one **ObjectType** directive in an object, the directives are applied in the order they appear. If a directive sets an attribute and a later directive tries to set that attribute to something else, the first setting is used and the subsequent one is ignored.

cache-enable (enabling caching)

The **cache_enable** function tells the proxy that an object is cacheable, based on specific criteria. As an example, if it appears in the object `<Object ppath="http://.*">`, then all the HTTP documents are considered cacheable, as long as other conditions for an object to be cacheable are met.

Syntax

```
ObjectType fn=cache-enable
   cache-auth=0|1
   query-maxlen=number
   min-size=number
   max-size=number
   log-report=feature
   cache-local=0|1
```

Parameters

cache-enable tells the proxy that an object is cacheable. As an example, if it appears in the object `<Object ppath="http://.*">`**,** then all HTTP documents are considered cacheable (as long as other conditions for an object to be cacheable are met).

cache-auth specifies whether to cache items that require authentication. If set to 1, pages that require authentication can be cached also. If not specified, defaults to 0.

query-maxlen specifies the number of characters in the query string (the “?string” part at the end of the URL) that are still cacheable. The same queries are rarely repeated exactly in the same form by more than one user, and so caching them is often not desirable. That’s why the default is 0.

min-size is the minimum size, in kilobytes, of any document to be cached. The benefits of caching are greatest with the largest documents. For this reason, some people prefer to cache only larger documents.

max-size represents the maximum size in kilobytes of any document to be cached. This allows users to limit the maximum size of cached documents, so no single document can take up too much space.

log-report is used to control the feature that reports local cache accesses back to the origin server so that content providers get their true access logs.

cache-local is used to enable local host caching, that is, URLs without fully qualified domain names, in the proxy. If set to 1, local hosts are cached. If not specified, it defaults to 0, and local hosts are not cached.

Example

The following example of **cache-enable** allows you to enable caching of objects matching the current resource. This applies to normal, non-query, non-authenticated documents of any size. The proxy requires that the document carries either last-modified or expires headers or both, and that the content-type reported by the origin server (if present) is accurate.

```
ObjectType fn=cache-enable
```

The example below is like the first example, but it also caches documents that require user authentication, and it caches queries up to five characters long. The **cache-auth=1** indicates that an up-to-date check is always required for documents that need user authentication (this forces authentication again).

```
ObjectType fn=cache-enable
   cache-auth=1
   query-maxlen=5
```

The example below is also like the first example, except that it limits the size of cache files to a range of 2 KB to 1 MB.

```
ObjectType fn=cache-enable
   min-size=2
   max-size=1000
```

cache-setting (specifying caching parameters)

cache-setting is an **ObjectType** function that sets parameters used for cache control. Defaults for these settings are provided through the **init-cache** function, described on page 430.

This function is used to explicitly cache (or not cache) a resource, create an object for that resource, and set the caching parameters for the object.

Syntax

```
ObjectType fn=cache-setting
   max-uncheck=seconds
   lm-factor=factor
   connect-mode=always|fast-demo|never
   cover-errors=number
```

Parameters

max-uncheck (optional) is the maximum time in seconds, allowed between consecutive up-to-date checks. If set to 0 (default), a check is made every time the document is accessed, and the **lm-factor** has no effect.

lm-factor (optional) is a floating point number representing the factor used in estimating expiration time (how long a document might be up to date based on the time it was last modified). The time elapsed since the last modification is multiplied by this factor, and the result gives the estimated time the document is likely to remain unchanged. Specifying a value of 0 turns off this function, and then the caching system uses only explicit expiration information (rarely available). Only explicit Expires HTTP headers are used. This value has no effect if **max-uncheck** is set to 0.

connect-mode specifies network connectivity and can be set to these values:

- **always** (default) connects to remote servers when necessary.
- **fast-demo** connects only if the item isn’t found in the cache.
- **never** no connection to a remote server is ever made; returns an error if the document is not found in the cache.

cover-errors, if present and greater than 0, returns a document from the cache if the remote server is down and an up-to-date check cannot be made. The value specified is the maximum number of seconds since the last up-to-date check; if more time has elapsed, an error is returned. Using this feature involves the risk of getting stale data from the cache while the remote server is down. Setting this value to 0, or not specifying it (default) causes an error to be returned if the remote server is unavailable.

term-percent means to keep retrieving if more than the specified percentage of the document has already been retrieved.

Example

```
<Object ppath="http://.*">
ObjectType fn=cache-enable
ObjectType fn=cache-setting max-uncheck="7200"
ObjectType fn=cache-setting lm-factor="0.020"
ObjectType fn=cache-setting connect-mode="fast-demo"
ObjectType fn=cache-setting cover-errors="3600"
Service fn=proxy-retrieve
</Object>

# Force check every time
ObjectType fn=cache-setting max-uncheck=0
# Check every 30 minutes, or sooner if changed less than
# 6 hours ago (factor 0.1; last change 1 hour ago would
# give 6-minute maximum check interval).
ObjectType fn=cache-setting max-uncheck=1800 lm-factor=0.1
# Disable caching of the current resource
ObjectType fn=cache-setting cache-mode=nothing
```

force-type (assigning MIME types to objects)

The **force-type** function assigns a type to objects that do not already have a MIME type. This is used to specify a default object type.

Syntax

```
ObjectType fn=force-type
   type=text/plain
   enc=encoding
   lang=language
```

Parameters

type is the type to assign to matching files.

enc (optional) is the encoding given to matching files.

lang (optional) is the language assigned to matching paths.

Example

```
ObjectType fn=force-type
   type=text/plain
```

```
ObjectType fn=force-type
   lang=en_US
```

http-config (using keep-alive feature)

http-config is an **ObjectType** function that lets the proxy use the HTTP keep-alive feature between the client and the proxy server, and between the proxy server and the remote server.

Syntax

```
ObjectType fn=http-config a
   keep-alive=on|off
```

Parameters

on enables this keep-alive feature.

off disables the keep-alive feature, and is the default.

The keep-alive feature lets several requests be sent through the same connection.

Using this feature could actually degrade performance if the proxy is heavily loaded and it receives a lot of new requests every second and the network can establish connections fairly quickly. The reason for this degradation is that every connection is kept by the server for several seconds after the request processing has finished, even if the client doesn't happen to send a new request.

If connections to the proxy server take a long time to establish, or if the connection simply hangs, this feature should be disabled to reduce the total number of active connections.

Example

```
ObjectType fn=http-config keep-alive=on
```

java-ip-check (checking IP addresses)

The **java-ip-check** function allows clients to query the proxy server for the IP address used to rerouted a resource. Because DNS spoofing often occurs with Java Applets, this feature enables clients to see the true IP address of the origin server. When this features is enabled, the proxy server attaches a header containing the IP address that was used for connecting to the destination origin server.

Syntax

```
ObjectType fn=java-ip-check
   status=on|off
```

Parameters

status specifies whether Java IP address checking is enabled or not. Possible values are:

- **on** means that Java IP address checking is enabled and that IP addresses will be forwarded to the client in the form of a document header. On is the default setting.
- **off** means that Java IP address checking is disabled.

type-by-extension (determining file information)

The **type-by-extension** function uses file extensions to determine information about files. (Extensions are strings after the last period in a file name.) This matches an incoming request to extensions in the mime.types file. The MIME type is added to the "content-type" header sent back to the client. The type can be set to internal server types that have special results when combined with function you write using the server plug-in API.

Syntax

```
ObjectType fn=type-by-extension
```

Parameters

None.

Example

```
ObjectType fn=type-by-extension
```

PathCheck

The **PathCheck** directives check the URL that is returned after all of the **NameTrans** directives finish running. Local file paths (with the administration forms) are checked for elements such as **../** and **//**, and then any access restriction is applied.

If an object has more than one **PathCheck** directive, all of the directives will be applied in the order they appear.

check-acl (attaching an ACL to an object)

The **check-acl** function attaches an Access Control List to the object in which the directive appears. Regardless of the order of **PathCheck** directives in the object, **check-acl** functions are executed first, and will cause user authentication to be performed if required by the specified ACL, and will also update the access control state.

Syntax

```
PathCheck fn=check-acl
    acl="ACL name"
    bong-file=path name
```

Parameters

acl is the name of an Access Control List.

bong-file (optional) is the path name for a file that will be sent if this ACL is responsible for denying access.

Example

```
PathCheck fn=check-acl
    acl="HRonly"
```

deny-service (denying client access)

The **deny-service** function is a **PathCheck** function that sends a “Proxy Denies Access” error when a client tries to access a specific path. If this directive appears in a client region, it performs access control on the specified clients.

The proxy specifically denies clients instead of specifically allowing them access to documents (for example, you don’t configure the proxy to allow a list of clients). The “default” object is used when a client doesn’t match any client region in objects, and because the “default” object uses the **deny-service** function, no one is allowed access by default.

Syntax

```
PathCheck fn=deny-service path=.*someexpression.*
```

Parameters

path is a regular expression representing the path to check. Not specifying this parameter is equivalent to specifying `*`. URLs matching the expression are denied access to the proxy server.

Example

```
<Object ppath="http://netscape/.*">
# Deny servicing proxy requests for fun GIFs
PathCheck fn=deny-service path=.*fun.*.gif
# Make sure nobody except Netscape employees can use the object
# inside which this is placed.
<Client dns=*~.*.netscape.com>
PathCheck fn=deny-service
</Client>
</Object>
```

require-proxy-auth (requiring proxy authentication)

The **require-proxy-auth** function is a **PathCheck** function that makes sure that users are authenticated and triggers a password pop-up window.

Syntax

```
PathCheck fn=require-proxy-auth
    auth-type=basic
    realm=name
    auth-group=group
    auth-users=name
```

Parameters

auth-type specifies the type of authorization to be used. The type should be “basic” unless you are running a UNIX proxy and are going to use your own function to perform authentication.

realm is a string (enclosed in double-quotation marks) sent to the client application so users can see what object they need authorization for.

auth-user (optional) specifies a list of users who get access. The list should be enclosed in parentheses with each user name separated by the pipe | symbol.

auth-group (optional) specifies a list of groups that get access. Groups are listed in the password-type file.

Example

```
PathCheck fn=require-auth
    auth-type=basic
    realm="Marketing Plans"
    auth-group=mktg
    auth-users=(jdoe|johnd|janed)
```

url-check (checking URL syntax)

The url-check function checks the validity of URL syntax.

Route

The Route directive specifies information about where the proxy server should route requests.

icp-route (routing with ICP)

The **icp-route** function tells the proxy server to use ICP to determine the best source for a requested object whenever the local proxy does not have the object.

Syntax

```
Route fn=icp-route
   redirect=yes|no
```

Parameters

redirect specifies whether the proxy server will send a redirect message back to the client telling it where to get the object.

- **yes** means the proxy will send a redirect message back to the client to tell it where to retrieve the requested object.
- **no** means the proxy will not send a redirect message to the client. Instead it will use the information from ICP to get the object.

pa-enforce-internal-routing (enforcing internal distributed routing)

The **pa-enforce-internal-routing** function enables internal routing through a proxy array. Internal routing occurs when a non PAC-enabled client routes requests through a proxy array.

Syntax

```
Route fn="pa_enforce_internal_routing"
   redirect="yes|no"
```

Parameters

redirect specifies whether or not client's requests will be redirected. Redirecting means that if a member of a proxy array receives a request that it should not service, it tells the client which proxy to contact for that request.

CAUTION Redirect is not currently supported by any clients, so you should not use the feature at this time.

pa-set-parent-route (setting a hierarchical route)

The **pa-set-parent-route** function sets a route to a parent array.

Syntax

```
Route fn="pa_set_parent_route"
```

set-proxy-server (using another proxy to retrieve a resource)

The **set-proxy-server** function directs the proxy server to connect to another proxy for retrieving the current resource. It also sets the address and port number of the proxy server to be used.

Syntax

```
Route fn=set-proxy-server
   host name=otherhost name
   port=number
```

Parameters

host name is the name of the host on which the other proxy is running.

port is the port number of the remote proxy.

Example

```
Route fn=set-proxy-server
   host name=proxy.netscape.com
   port=8080
```

set-socks-server (using a SOCKS server to retrieve a resource)

The **set-socks-server** directs the proxy server to connect to a SOCKS server for retrieving the current resource. It also sets the address and port number of the SOCKS server to be used.

Syntax

```
Route fn=set-socks-server
   host name=sockshost name
   port=number
```

Parameters

host name is the name of the host on which the SOCKS server runs.

port is the port on which the SOCKS server listens.

Example

```
ObjectType fn=set-socks-server
   host name=socks.netscape.com
   port=1080
```

unset-proxy-server (unsetting a proxy route)

The **unset-proxy-server** function tells the proxy server not to connect to another proxy server to retrieve the current resource. This function nullifies the settings of any less specific set-proxy-server functions.

Syntax

```
Route fn=unset-proxy-server
```

unset-socks-server (unsetting a SOCKS route)

The **unset-socks-server** function tells the proxy server not to connect to a SOCKS server to retrieve the current resource. This function nullifies the settings of any less specific set-socks-server functions.

Syntax

```
Route fn=unset-socks-server
```

Service

Once the other directives have done all the necessary checks and translations, the functions of the **Service** directive send the data (first receiving it from a remote server when necessary) and complete the request. Most of the time, the **Service** directive connects to a remote server, making the request for the client and then passing the results back to the client.

Parameters

Service directives support these optional parameters to help determine if the directive is used:

method specifies a regular expression that indicates which HTTP methods the client must be using to have the directive applied. Valid HTTP methods include GET, HEAD, POST, and INDEX (CONNECT through SSL tunneling is also available). Multiple values are enclosed in parentheses and separated by the pipe (|) symbol.

type (not with **proxy-retrieve**) specifies a regular expression that indicates the MIME types to which to apply the directive. The proxy server defines several MIME types internally that are used only to select a **Service** function that translates the internal type into a form presentable to the client.

If an object has more than one **Service** directive, the first applicable directive is used and the rest are ignored.

proxy-retrieve (retrieving documents with the proxy)

The **proxy-retrieve** function retrieves a document from a remote server and returns it to the client. It manages caching if it is enabled. The **proxy-retrieve** function also lets you configure the proxy to allow or block arbitrary methods.

Syntax

```
Service fn=proxy-retrieve
   method=GET|HEAD|POST|INDEX|CONNECT...

   allow|block=<List-of-comma-separated-methods>
```

Parameters

method lets you specify a retrieval method.

allow configures the proxy to allow specified arbitrary methods.

block configures the proxy to block specified arbitrary methods.

NOTE **allow** takes precedence over **block**.

Examples

```
# Normal proxy retrieve
Service fn=proxy-retrieve
# Proxy retrieve with POST method disabled
Service fn=proxy-retrieve
   method=(POST)
# Proxy retrieve allows methods FOO and BAR to pass through
Service fn=proxy-retrieve
```

```
   allow="FOO,BAR"
# Proxy retrieve blocks methods MKCOL,DELETE,LOCK,UNLOCK
Service fn=proxy-retrieve
   block="MKCOL,DELETE,LOCK,UNLOCK"
```

send-file (sending text file contents to client)

The **send-file** function sends the contents of a plain text file to the client. If this function finds any extra path information, it doesn't send the text file to the client.

Syntax

```
Service fn=send-file
   method=GET|HEAD|POST|INDEX|CONNECT...
   type=MIME type
```

Parameters

method lets you specify a retrieval method. By default, all methods are allowed unless the **method** parameter is given.

type specifies a regular expression that indicates the MIME types to which to apply the directive.

Example

```
Service fn=send-file
   method=(GET|HEAD)
   type=*~magnus-internal/*
```

deny-service (denying access to a resource)

The **deny-service** function is the only function that belongs to two classes: **PathCheck** and **Service**. It prevents access to the requested resource.

The socks5.conf File

The proxy uses the file `/etc/socks5.conf` to control access to the SOCKS proxy server SOCKD and its services. Each line defines what the proxy does when it gets a request that matches the line.

When SOCKD receives a request, it checks the request against the lines in `/etc/socks5.conf`. When it finds a line that matches the request, the request is permitted or denied based on the first word in the line (permit or deny). Once it finds a matching line, the daemon ignores the remaining lines in the file. If there are no matching lines, the request is denied. You can also specify actions to take if the client’s identd or user ID is incorrect by using #NO_IDENTD: or #BAD_ID as the first word of the line. Each line can be up to 1023 characters long.

There are five sections in the `socks5.conf` file. These sections do not have to appear in the following order. However, because the daemon uses only the first line that matches a request, the order of the lines within each section is extremely important. The five sections of the `socks5.conf` file are:

- ban host/authentication - identifies the hosts from which the SOCKS deamon should not accept connections and which types of authentication the SOCKS daemon should use to authenticate these hosts
- routing - identifies which interface the SOCKS deamon should use for particular IP addresses
- variables and flags - identifies which logging and informational messages the SOCKS daemon should use
- proxies - identifies the IP addresses that are accessible through another SOCKS server and whether that SOCKS server connects directly to the host
- access control - specifies whether the SOCKS daemon should permit or deny a request

When the SOCKS daemon receives a request, it sequentially reads the lines in each of these five sections to check for a match to the request. When it finds a line that matches the request, it reads the line to determine whether to permit or deny the request. If there are no matching lines, the request is denied.

Each line in this file can be up to 1023 characters long and in order for a line to match a request, each entry in the line must match.

Authentication/Ban Host Entries

There are two lines in authentication/ban host entries. The first is the authentication line.

Syntax

`auth` *source-hostmask source-portrange auth-methods*

Parameters

source-hostmask identifies which hosts the SOCKS server will authenticate.

source-portrange identifies which ports the SOCKS server will authenticate.

auth-methods are the methods to be used for authentication. You can list multiple authentication methods in order of your preference. In other words, if the client does not support the first authentication method listed, the second method will be used instead. If the client does not support any of the authentication methods listed, the SOCKS server will disconnect without accepting a request. If you have more than one authentication method listed, they should be separated by commas with no spaces in between. Possible authentication methods are:

- `n` (no authentication required)
- `u` (user name and password required)
- `-` (any type of authentication)

The second line in the authentication/ban host entry is the ban host line.

Syntax

`ban` *source-hostmask source-portrange*

Parameters

source-hostmask identifies which hosts are banned from the SOCKS server.

source-portrange identifies from which ports the SOCKS server will not accept requests.

Example

```
auth 127.27.27.127 1024 u,-
ban 127.27.27.127 1024
```

Routing Entries

Syntax

`route` *dest-hostmask dest-portrange interface/address*

Parameters

dest-hostmask indicates the hosts for which incoming and outgoing connections must go through the specified interface.

dest-portrange indicates the ports for which incoming and outgoing connections must go through the specified interface.

interface/address indicates the IP address or name of the interface through which incoming and outgoing connections must pass. IP addresses are preferred.

Example

```
route 127.27.27.127 1024 le0
```

Variables and Flags

Syntax

`set` *variable value*

Parameters

variable indicates the name of the variable to be initialized.

value is the value to set the variable to.

Example

```
set SOCKS5_BINDPORT 1080
```

Available Settings

The following settings are those that can be inserted into the variables and flags section of the `SOCKS5.conf` file. These settings will be taken from the administration forms, but they can be added, changed, or removed manually as well.

SOCKS5_BINDPORT

The S**OCKS5_BINDPORT** setting sets the port at which the SOCKS server will listen. This setting cannot be changed during rehash.

Syntax

`set SOCKS5_BINDPORT` *port number*

Parameters

port number is the port at which the SOCKS server will listen.

Example

```
set SOCKS5_BINDPORT 1080
```

SOCKS5_PWDFILE

The **SOCKS5_PWDFILE** setting is used to look up user name/password pairs for user name/password authentication. This setting only applies to situations in which ns-sockd is running separately from the administration server.

Syntax

`set SOCKS5_PWDFILE` *full pathname*

Parameters

full pathname is the location and name of the user name/password file.

Example

```
set SOCKS5_PWDFILE /etc/socks5.passwd
```

SOCKS5_CONFFILE

The **SOCKS5_CONFFILE** setting is used to determine the location of the SOCKS5 configuration file.

Syntax

`set SOCKS5_CONFFILE` *full pathname*

Parameters

full pathname is the location and name of the SOCKS configuration file.

Example

```
set SOCKS5_CONFFILE /etc/socks5.conf
```

SOCKS5_LOGFILE

The **SOCKS5_LOGFILE** setting is used to determine where to write log entries.

Syntax

`set SOCKS5_LOGFILE` *full pathname*

Parameters

full pathname is the location and name of the SOCKS logfile.

Example

```
set SOCKS-5_LOGFILE /var/log/socks5.log
```

SOCKS5_NOIDENT

THe **SOCKS5_NOIDENT** setting disables Ident so that SOCKS does not try to determine the user name of clients. Most servers should use this setting unless they will be acting mostly as a SOCKS4 server. (SOCKS4 used ident as authentication.)

Syntax

```
set SOCKS5_NOIDENT
```

Parameters

None.

SOCSK5_DEMAND_IDENT

The **SOCKS5_DEMAND_IDENT** setting sets the Ident level to "require an ident response for every request". Using Ident in this way will dramatically slow down your SOCKS server. If neither SOCKS5_NOIDENT or SOCKS5_DEMAND_IDENT is set, then the SOCKS server will make an Ident check for each request, but it will fulfill requests regardless of whether an Ident response is received.

Syntax

```
set SOCSK5_DEMAND_IDENT
```

Parameters

None.

SOCKS5_DEBUG

The **SOCKS5_DEBUG** setting causes the SOCKS server to log debug messages. You can specify the type of logging your SOCKS server will use.

If it's not a debug build of the SOCKS server, only number 1 will work.

Syntax

`set SOCSK5_DEBUG` *number*

Parameters

number determines the number of the type of logging your server will use. Possible values are:

- **1 -** log normal debugging messages. This is the default.
- **2 -** log extensive debugging (especially related to configuration file settings).
- **3 -** log all network traffic.

Example

```
set SOCKS5_DEBUG 2
```

SOCKS5_USER

The **SOCKS5_USER** setting sets the user name to use when authenticating to another SOCKS server.

Syntax

`set SOCKS5_USER` *user name*

Parameters

user name is the user name the SOCKS server will use when authenticating to another SOCKS server.

Example

```
set SOCKS5_USER mozilla
```

SOCKS5_PASSWD

The **SOCKS5_PASSWD** setting sets the password to use when authenticating to another SOCKS server. It is possible for a SOCKS server to go through another SOCKS server on its way to the Internet. In this case, if you define SOCKS5_USER, ns-sockd will advertise to other SOCKS servers that it can authenticate itself with a user name and password.

Syntax

`set SOCKS5_PASSWD` *password*

Parameters

password is the password the SOCKS server will use when authenticating to another SOCKS server.

Example

```
set SOCKS5_PASSWD m!2@
```

SOCKS5_NOREVERSEMAP

The **SOCKS5_NOREVERSEMAP** setting tells ns-sockd not to use reverse DNS. Reverse DNS translates IP addresses into host names. Using this setting can increase the speed of the SOCKS server.

If you use domain masks in the configuration file, the SOCKS server will have to use reverse DNS, so this setting will have no effect.

Syntax

```
set SOCKS5_NOREVERSEMAP
```

Parameters

None.

SOCKS5_HONORBINDPORT

The **SOCKS5_HONORBINDPORT** setting allows the client to specify the port in a BIND request. If this setting is not specified, the SOCKS server ignores the client's requested port and assigns a random port.

Syntax

```
set SOCKS5_HONORBINDPORT
```

Parameters

None.

SOCKS5_ALLOWBLANKETBIND

The **SOCKS5_ALLOWBLANKETBIND** setting allows the client to specify an IP address of all zeros (0.0.0.0) in a BIND request. If this setting is not specified, the client must specify the IP address that will be connecting to the bind port, and an IP of all zeros is interpreted to mean that any IP address can connect.

Syntax

```
set SOCKS5_ALLOWBLANKETBIND
```

Parameters

None.

SOCKS5_STATSFILE

The **SOCKS5_STATSFILE** setting identifies a different file for storing running statistics about the SOCKS server.

Syntax

`set SOCKS5_STATSFILE` *full pathname*

Parameters

full pathname is the location and name of the statistics file.

Example

```
set SOCKS5_STATSFILE /tmp/socksstat.any.1080
```

SOCSK5_QUENCH_UPDATES

The **SOCKS5_QUENCH_UPDATES** setting tells the SOCKS server *not* to write a line to the logfile every hour. This line, if written, provides a brief summary of statistics. The following is a sample line:

```
[04/aug/1997:21:00:00] 000 info: 78 requests,
78 successful: connect 77, bind 1, udp 0
```

Syntax

```
set SOCKS5_QUENCH_UPDATES
```

Parameters

None.

SOCKS5_WORKERS

The **SOCKS5_WORKERS** setting tunes the performance of the SOCKS server by adjusting the number of worker threads. Worker threads perform authentication and access control for new SOCKS connections. If the SOCKS server is too slow, you should increase the number of worker threads. If it is unstable, decrease the number of worker threads.

The default number of worker threads is 40, and the typical number of worker threads falls between 10 and 150.

Syntax

`set SOCKS5_WORKERS` *number*

Parameters

number is the number of worker threads the SOCKS server will use.

Example

```
set SOCKS5_WORKERS 40
```

SOCKS5_ACCEPTS

The **SOCKS5_ACCEPTS** setting tunes the performance of the SOCKS server by adjusting the number of accept threads. Accept threads sit on the SOCKS port listening for new SOCKS requests. If the SOCKS server is dropping connections, you should increase the number of accept threads. If it is unstable, decrease the number of accept threads.

The default number of accept threads is 1, and the typical number of accept threads falls between 1 and 10.

Syntax

`set SOCKS5_ACCEPTS` *number*

Parameters

number is the number of accepts threads the SOCKS server will use.

Example

```
set SOCKS5_ACCEPTS 1
```

LDAP_URL

The **LDAP-URL** setting sets the URL for the LDAP server.

Syntax

`set LDAP-URL` *URL*

Parameters

URL is the URL for the LDAP server used by SOCKS.

Example

```
set LDAP-URL ldap://name:8180/0=Netscape,c=US
```

LDAP_USER

The **LDAP-USER** setting sets the user name that the SOCKS server will use when accessing the LDAP server.

Syntax

`set LDAP-USER` *user name*

Parameters

user name is the user name SOCKS will use when accessing the LDAP server.

Example

```
set LDAP-USER admin
```

LDAP_PASSWD

The **LDAP-PASSWD** setting sets the password that the SOCKS server will use when accessing the LDAP server.

Syntax

`set LDAP-PASSWD` *password*

Parameters

password is the password SOCKS will use when accessing the LDAP server.

Example

```
set LDAP-PASSWD T$09
```

SOCKS5_TIMEOUT

The **SOCKS5-TIMEOUT** setting specifies the idle period that the SOCKS server will keep a connection alive between a client and a remote server before dropping the connection.

Syntax

`set SOCKS5_TIMEOUT` *time*

Parameters

time is the time, in minutes, SOCKS will wait before timing out. The default value is 10. The value can range from 10 to 60, including both these values.

Example

```
set SOCKS5_TIMEOUT 30
```

Proxy Entries

Syntax

proxy-type dest-hostmask dest-portrange proxy-list

Parameters

proxy-type indicates the type of proxy server. This value can be:

- socks5 - SOCKS version 5
- socks4 - SOCKS version 4
- noproxy - a direct connection

dest-hostmask indicates the hosts for which the proxy entry applies.

dest-portrange indicates the ports for which the proxy entry applies.

proxy-list contains the names of the proxy servers to use.

Example

```
socks5 127.27.27.127 1080 proxy1
```

Access Control Entries

Syntax

`permit|deny` *auth-type connection-type source-hostmask dest-hostmask source-portrange dest-portrange* [*LDAP-group*]

Parameters

auth-type indicates the authentication method for which this access control line applies.

connection-type indicates the type of command the line matches. Possible command types are:

- c (connect)
- b (bind; open a listen socket)
- u (UDP relay)
- - (any command)

source-hostmask - indicates the hosts for which the access control entry applies.

dest-hostmask indicates the hosts for which the access control entry applies.

source-portrange indicates the ports for which the access control entry applies.

dest-portrange is the port number of the destination.

LDAP-group is the group to deny or permit access to. This value is optional. If no LDAP group is identified, the access control entry applies to everyone.

Example

```
permit u c - - - [0-1023] group1
```

Specifying Ports

You will need to specify ports for many entries in your `socks5.conf` file. Ports can be identified by a name, number, or range. Ranges that are inclusive should be surrounded by brackets (i.e. [ ]). Ranges that are not inclusive should be in parentheses.

The bu.conf File

The optional `bu.conf` file contains batch update directives. You can use these directives to update many documents at once. You can time these updates to occur during off-peak hours to minimize the effect on the efficiency of the server. The format of this file is described in this section.

Accept

A valid URL **Accept** filter consists of any POSIX regular expression. It is used as a filter to test URLs for retrieval in the case of internal updates, and determines whether branching occurs for external updates.

This directive may occur any number of times, as separate **Accept** lines or as comma or white space delimited entries on a single **Accept** line and is applied sequentially. Default behavior is .*, letting all URLs pass.

Syntax

```
Accept regular expression
```

Connections

For the **Connections** directive, *n* is the number of simultaneous connections to be used while retrieving. This is a general method for limiting the load on your machine and, more importantly, the remote servers being contacted.

This directive can occur multiple times in a valid configuration, but only the smallest value is used.

Syntax

```
Connections n
```

Count

The argument *n* of the **Count** directive specifies the total maximum number of URLs to be updated via this process. This is a simple safeguard for limiting the process and defaults to a value of 300. This directive can occur multiple times in a valid configuration, but only the smallest value is used.

Syntax

```
Count n
```

Days

The **Days** directive specifies on which days you want to allow the starting of batch updates. You can specify this by naming the days of the week (Sunday,..., Saturday), and you can use three-letter abbreviations (Sun, Mon, Tue, Wed, Thu, Fri, Sat).

This directive can occur multiple times in a valid configuration, but only the first value is used. The default is seven-day operation.

Syntax

```
Days day1 day2...
```

Depth

The **Depth** directive lets you ensure that, while enumerating, all collected objects are no more than a specified number of links away from the initial URL. The default is 1.

Syntax

```
Depth depth
```

Object boundaries

The **Object** wrapper signifies the boundaries between individual configurations in the `bupdate.conf` file. It can occur any number of times, though each occurrence requires a unique name.

All other directives are only valid when inside **Object** boundaries.

Syntax

```
<Object name=name>
...
</Object>
```

Reject

A valid URL **Reject** filter consists of any POSIX regular expression. It is used as a filter to test URLs for retrieval in the case of internal updates, and determines whether branching occurs for external updates.

This directive may occur any number of times, as separate **Reject** lines or as comma or white space delimited entries on a single **Reject** line, and is applied sequentially. Default behavior is no reject for internal updates and .* (no branching, get single URL) for recursive updates.

Syntax

`Reject` *`regular expression`*

Source

In the **Source** directive, if the argument is the keyword **internal**, it specifies batch updates are to be done only on objects currently in the cache (and a directive of **Depth 1** is assumed); otherwise, you specify the name of a URL for recursive enumeration.

This directive can occur only once in a valid configuration.

Syntax

`Source internal`

`Source` *`URL`*

Time

The **Time** directive specifies the time to start and stop updates. Valid values range from 00:00 to 24:00 (24-hour “military” time).

This directive can occur multiple times in a valid configuration, but only the first value will be used.

Syntax

`Time` *`start - end`*

Type

This function lets you control the updating of mime types that the proxy caches. This directive can occur any number of times, in any order.

Syntax

`Type ignore`

```
Type inline
```

```
Type mime_type
```

Parameters

ignore means that updates will act on all MIME types that the proxy currently caches. This is the default behavior and supersedes all other **Type** directives if specified.

inline means that in-lined data is updated as a special type, regardless of any later MIME type exclusions, and are meaningful only when doing recursive updates.

mime-type is assumed to be a valid entry from the system `mime-types` file, and is included in the list of MIME types to be updated. If the proxy doesn't currently cache the given MIME type, the object may be retrieved but is not cached.

The icp.conf File

This file is used to configure the Internet Cache Protocol (ICP) feature of your server. There are three functions in the `icp.conf` file, and each can be called as many times as necessary. Each function should be on a separate line. The three functions are **add_parent**, **add_sibling**, and **server**.

add_parent (adding parent servers to an ICP neighborhood)

The **add_parent** function identifies and configures a parent server in an ICP neighborhood.

Syntax

```
add_parent name=name icp_port=port number
proxy_port=port number mcast_address=IP address ttl=number round=1|2
```

NOTE	The above text should all be on one line in the `icp.conf` file.

Parameters

name specifies the name of the parent server. It can be a dns name or an IP address.

icp_port specifies the port on which the parent listens for ICP messages.

proxy_port specifies the port for the proxy on the parent.

mcast_address specifies the multicast address the parent listens to. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors listening to that multicast address can receive, therefore eliminating the need to send a query to each neighbor separately.

ttl specifies the time to live for a message sent to the multicast address. ttl controls the number of subnets a multicast message will be forwarded to. If the ttl is set to 1, the multicast message will only be forwarded to the local subnet. If the ttl is 2, the message will go to all subnets that are one hop away.

round specifies in which polling round the parent will be queried. A polling round is an ICP query cycle. Possible values are:

- **1** means that the parent will be queried in the first query cycle with all other round one neighbors.
- **2** means that the parent will only be queried if none of the neighbors in polling round one return a "HIT."

Example

```
add_parent name=proxy1 icp_port=5151 proxy_port=3333
mcast_address=189.98.3.33 ttl=3 round=2
```

add_sibling (adding sibling servers to an ICP neighborhood)

The **add_sibling** function identifies and configures a sibling server in an ICP neighborhood.

Syntax

`add_sibling name=`*name* `icp_port=`*port number* `proxy_port=`*port number*
`mcast_address=`*IP address* `ttl=`*number* `round=1|2`

NOTE The above text will all be on one line in the `icp.conf` file.

Parameters

name specifies the name of the sibling server. It can be a dns name or an IP address.

icp_port specifies the port on which the sibling listens for ICP messages.

proxy_port specifies the port for the proxy on the sibling.

mcast_address specifies the multicast address the sibling listens to. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors listening to that multicast address can receive, therefore eliminating the need to send a query to each neighbor separately.

ttl specifies the time to live for a message sent to the multicast address. ttl controls the number of subnets a multicast message will be forwarded to. If the ttl is set to 1, the multicast message will only be forwarded to the local subnet. If the ttl is 2, the message will go to all subnets that are one hop away.

round specifies in which polling round the sibling will be queried. A polling round is an ICP query cycle. Possible values are:

- **1** means that the sibling will be queried in the first query cycle with all other round one neighbors. This is the default polling round value.
- **2** means that the sibling will only be queried if none of the neighbors in polling round one return a "HIT."

Example

```
add_sibling name=proxy2 icp_port=5151 proxy_port=3333
mcast_address=190.99.2.11 ttl=2 round=1
```

NOTE The above text will all be on one line in the `icp.conf` file.

server (configuring the local proxy in an ICP neighborhood)

The **server** function identifies and configures the local proxy in an ICP neighborhood.

Syntax

```
server bind_address=IP address mcast=IP address num_servers=number
icp_port=port number default_route=name default_route_port=port number
no_hit_behavior=fastest_parent|default timeout=seconds
```

NOTE The above text will all be on one line in the `icp.conf` file.

Parameters

bind_address specifies the IP address to which the server will bind. For machines with more than one IP address, this parameter can be used to determine which address the ICP server will bind to.

mcast the multicast address to which the neighbor listens. A multicast address is an IP address to which multiple servers can listen. Using a multicast address allows a proxy to send one query to the network that all neighbors who are listening to that multicast address can see, therefore eliminating the need to send a query to each neighbor separately.

If both a multicast address and bind address are specified for the neighbor, the neighbor uses the bind address to communicate with other neighbors. If neither a bind address nor a multicast address is specified, the communication subsystem will decide which address to use to send the data.

num_servers specifies the number of processes that will service ICP requests.

icp_port specifies the port number to which the server will listen.

default_route tells the proxy server where to route a request when none of the neighboring caches respond. If default_route and default_route_port are set to "origin," the proxy server will route defaulted requests to the origin server. The meaning of default_route is different depending on the value of no_hit_behavior. If no_hit_behavior is set to default, the default_route is used when none of the proxy array members return a hit. If no_hit behavior is set to fastest_parent, the default_route value is used only if no parent responds.

default_route_port specifies the port number of the machine specified as the default_route. If default_route and default_route_port are set to "origin," the proxy server will route defaulted requests to the origin server.

no_hit_behavior specifies the proxy's behavior whenever none of the neighbors returns a "HIT" for the requested document. Possible values are:

- **fastest_parent** means the request is routed through the first parent that returned a "MISS."
- **default** means the request is routed to the machine specified as the default route.

timeout specifies the maximum number of milliseconds the proxy will wait for an ICP response.

Example

```
server bind_address=198.4.66.78 mcast=no num_servers=5 icp_port=5151
default_route=proxy1 default_route_port=8080
no_hit_behavior=fastest_parent timeout=2000
```

NOTE The above text will all be on one line in the `icp.conf` file.

Glossary

Administration Server The HTTP server used to configure any Netscape 2.0 servers, such as iPlanet Web Proxy Server, installed on your machine.

Cache A storage area that contains copies of original data stored locally so that the data doesn’t have to be retrieved from a remote server each time it is requested.

Cache build The creation of the cache hierarchy.

Cache capacity How much data the cache can hold and still be efficient and effective. Cache capacity is related to the cache hierarchy in the cache directories. The larger the hierarchy, the bigger the capacity. The cache capacity should be configured to be equal to or greater than the cache size.

Cache directory hierarchy The proxy’s directory structure for storing cache files.

Cache Manager A periodic clean-up process to remove old files to make room for new ones.

Cache Manager daemon A process that monitors the cache size and spawns the Cache Manager when necessary.

Cache Monitor A process daemon for determining the status of the cache directory structure.

Cache refresh Replacing a cached document with a new copy from the content server.

Cache repair A process to repair a cache damaged by a software failure, system crash, disk breakdown, or full file system.

Cache root A directory on the proxy server machine that contains all cached files. The proxy controls which documents are copied to the cache root, and the Cache Manager daemon purges this directory structure to control the amount of data stored.

Cache partition You can divide the cache into multiple directories or disk partitions.

Cache size The total amount of disk space available for the proxy cache directory structure, which can be specified during initial proxy configuration and can later be changed through the online forms or the obj.conf configuration file. For efficiency, the cache size should not exceed the cache capacity.

Cache section Section of the iPlanet Web Proxy Server cache. The number of cache sections can be from 1 to 256, and must be a power of two (1, 2, 4, 8, 16, ..., 256). Each cache section can hold 100-250 megabytes of data; the optimum size is around 125 MB per section.

Cache up-to-date check A check to determine if the copy in the cache is still valid, and if not, refresh it.

CERN The European Laboratory for Particle Physics (CERN) invented the World Wide Web to share information among research groups. This is where the CERN proxy prototype was produced.

client An individual user or the web browser they are using (such as Netscape Navigator).

Common logfile format The format used by the server for entering information into the access logs. The format is the same among all of the major servers.

Content server A server that contains the original documents that are requested by clients directly or through a proxy server.

DMZ Demilitarized Zone. Taken from the military term for a safety zone between battle lines, this refers to an area within the firewall. Often this is a single machine with access to the internal site and the outside network. See also *firewall.*

DNS Domain Name Service. The system used by machines on a network to associate standard IP addresses (such as 198.95.251.) with host names (such as www.netscape.com). Machines typically get this translated information from a DNS server, or look it up in tables maintained on their systems.

DNS alias A host name that points to another host name—specifically a DNS CNAME record. Machines always have only one real name, but they can have more than one alias. For example, `www.[yourdomain].[domain]` might be an alias that points to a real machine called `realthing.[yourdomain].[domain]` where the server currently exists.

EMACS A UNIX text editor that can also be used to read e-mail and news.

Expire To label a document as "expired," or too old to serve to a client. The proxy will retrieve a current copy directly from the content server the next time a client requests the document. If the content server is unavailable, the expired document can still be served to the client with a message stating that it isn't current.

Expires header A header that contains the expiration time of the returned document, as specified by the remote server.

Extended logfile format Similar to the common logfile format, but it contains additional information.

File extension The last section of a file name that typically defines the type of file (for example, .GIF and .HTML). For example, in the filename `index.html` the file extension is `html`.

File type The format of a given file. For example, a graphics file doesn't have the same file type as a text file. File types are usually identified by the file extension (.GIF or .HTML).

Firewall A network configuration, usually both hardware and software, that forms a fortress between networked computers within an organization and those outside the organization. It is commonly used to protect information such as a network's e-mail and data files within a physical building or organization site. The area within the firewall is called the *demilitarized zone*, or DMZ. Often, a single machine in the DMZ is allowed access to both internal and external computers. The computer in the DMZ is directly interacting with the Internet, so strict security measures on it are required.

GIF The Graphics Interchange Format A cross-platform image format originally created by CompuServe. GIF files are usually much smaller than other graphic file types (.BMP, .TIFF). GIF is one of the most common interchange formats. GIF images are readily viewable on UNIX, Microsoft Windows, and Apple Macintosh systems.

Hard restart Terminating the process, and starting it up again.

Host name A name for a machine of the form machine.subdomain.domain, which is translated into an IP address. For example, www.netscape.com is the machine www in the subdomain netscape and com domain.

HTML Hypertext Markup Language is a formatting language used for documents on the World Wide Web. HTML files are plain text files with formatting Codes that tell browsers such as the Netscape Navigator how to display text, position graphics and form items, and display links to other pages.

HTTP Hypertext Transfer Protocol is the method for exchanging information between HTTP servers and clients.

HTTPD HTTP daemon, a program that serves information using the HTTP protocol. The Netscape Communications Server is often called an httpd.

HTTPS A secure version of HTTP, implemented using the secure sockets layer, SSL.

IANA The Internet Assigned Numbers Authority, an organization that assigns port numbers to specific types of communications.

inittab A file that lists programs that need to be restarted if they stop for any reason (this ensures a program continually runs). It is also called `/etc/inittab` because of its location. This isn’t available on all UNIX systems.

IP address Internet Protocol address—a set of numbers, separated by dots, that specifies the actual location of a machine on the Internet.

Jail A state in which a proxy’s access is limited to a given directory. The chroot directive lets the UNIX system administrator place a proxy server into a “jail” where it has access only to files in a given directory. This helps limit damage if the server’s security is compromised, because the intruder can access only the files in the one directory.

Last-modified header The last modification time of the document file, returned in the HTTP response from the server.

MD5 A message digest algorithm by RSA Data Security, Inc., which can be used to produce a short digest of data of any size, which has a high probability of being unique. It is mathematically extremely difficult to reproduce the same message digest.

MD5 signature A message digest produced by the MD5 algorithm.

MIME Multi-Purpose Internet Mail Extensions. This is an emerging standard for multimedia e-mail and messaging.

NIS Network Information Service—a system of programs and data files that UNIX machines use to collect, collate, and share specific information about machines, users, file systems, and network parameters throughout a network of computers.

NCSA The National Center for Supercomputing Applications is a research organization at the University of Illinois at Urbana-Champaign.

Password file A file on UNIX machines that stores UNIX user login names, passwords, and user ID numbers. It is also known as **/etc/passwd**, because of where it is kept. The proxy also has its own password files for user authentication; these are *not* connected with UNIX users.

pid Process identification. The name of a process.

proxy Server software, typically installed in the firewall DMZ, that allows access to the Internet across the firewall. A proxy is a special server that typically runs in conjunction with firewall software. The proxy server waits for a request from inside the firewall, forwards the request to the remote server outside the firewall, reads the response, then sends the response back to the client. The iPlanet Web Proxy Server also provides caching of documents for improved performance, extensive logging, and fine-grain access control.

RAM Random Access Memory. The physical semiconductor-based memory in a computer.

rc.local A file that describes programs that are run when the machine starts. It is also called **/etc/rc.local** because of its location.

Redirection A system by which clients accessing a particular URL are sent to a different location, either on the same server or on a different server. This is useful if a resource has moved and you want the clients to use the new location transparently. It’s also used to maintain the integrity of relative links when directories are accessed without a trailing slash.

Regular expression A form of expression that is used in Proxy for wildcard patterns for access control.

Resource Any document (URL), directory, or program that the server can access and send to a client.

Root The most privileged user available on UNIX machines (also called superuser). The root user has complete access privileges to all files on the machine.

Server daemon A process that, once running, listens for and accepts requests from clients.

Server root A directory on the server machine dedicated to holding the server program, configuration, maintenance, and information files.

SOCKS Firewall software that establishes a connection from inside a firewall to the outside when direct connection would otherwise be prevented by the firewall software or hardware (for example, the router configuration).

Soft restart A process that causes the server to internally restart, that is, reread its configuration files, by sending the **-HUP** signal (signal number one) to the process. The process itself does not die, as it does in a hard restart.

SSL Secure Sockets Layer. A software library establishing a secure connection between two parties (client and server) used to implement HTTPS, the secure version of HTTP.

Superuser The most privileged user available on UNIX machines (also called root). The superuser has complete access privileges to all files on the machine.

telnet A protocol where two machines on the network are connected to each other and support terminal emulation for remote login.

Timeout A specified time after which the server should give up trying to finish a service routine that appears hung.

top A program on some UNIX systems that shows the current state of system resource usage.

Top-level domain authority The highest category of host name classification, usually signifying either the type of organization the domain is (**.com** is a company, **.edu** is an educational institution) or the country of its origin (**.us** is the United States, **.jp** is Japan, **.au** is Australia, **.fi** is Finland).

uid User identification. A unique number associated with each UNIX user on a machine.

URL Uniform Resource Locator. The addressing system used by the server and the client to request documents. It is often called a location. The format of a URL is `[protocol]://[machine:port]/[document]`

An example of a URL is **http://www.netscape.com/index.html**.

URL list A list in the cache that contains all the URLs found in the cache, and links them to the cache files. This file can be browsed using the Cache Manager.

URL list repair A process that repairs and updates a URL list that has been damaged by a software failure, a system crash, a disk breakdown, or a full file system.

white space Any keystroke that leaves space on the screen, such as space bar, cursor return, line feed, horizontal tab, or vertical tab. In the `obj.conf` file, you can continue a directive line by adding white space at the beginning of the next line.

Index